Supporting Variable Bit-Rate Streams in a Scalable Continuous Media Server

by

Stergios V. Anastasiadis

A thesis submitted in conformity with the
requirements for the Degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Supporting Variable Bit-Rate Streams in a Scalable Continuous Media Server

Stergios V. Anastasiadis

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

2001

Variable bit-rate encoding of motion video has been shown to generate streams of considerably smaller size than constant bit-rate encoding of equivalent quality. Therefore, efficient support for bit-rate variability in continuous media servers has the potential to significantly reduce the requirements for disk storage space, disk bandwidth, server buffer space, and network bandwidth. Nonetheless, such flexibility has been previously discouraged due to system design complications and excessive expectations from technological progress.

In this thesis, we describe the design of a distributed media server architecture and the implementation details of a prototype. Variable bit-rate streams are striped efficiently across multiple disks with deterministic quality of service guarantees. In contrast to results of earlier studies, the number of concurrent playbacks supported is shown to increase almost linearly with the number of disks. Several factors contribute to this conclusion, including the performance evaluation method and the resource management policies that we use. New approaches are introduced for scheduling the playback requests and disk transfers, organizing the memory buffers, allocating the storage space, and structuring the stream metadata. We justify several of our decisions with comparative performance measurements using both synthetic benchmarks and actual experiments with variable bit-rate streams.

High variability in stream resource requirements can lead to reduced utilization of system resources. Previous smoothing techniques tried to address this problem by prefetching stream data into buffer space of the client device. Thus, the maximum transfer bandwidth can be reduced depending on the memory configuration of the individual client. Instead, we introduce a new smoothing algorithm that can decrease the maximum required disk bandwidth by prefetching stream data into server buffers. We show that the algorithm has optimal smoothing

effect under the specified constraints, and can be successfully applied to streams striped across either homogeneous or heterogeneous disks. Experiments with our prototype server demonstrate considerable improvement in the disk bandwidth utilization, and the number of streams supported at different system scales.

Acknowledgments

I am mostly grateful to Ken Sevcik, whose invaluable guidance made possible the successful completion of this thesis. He instilled me with insight about computer systems performance evaluation, while in numerous hours of discussions, he devoted endless effort to clarify and formalize several of the problems and approaches presented here.

Michael Stumm's co-supervision helped make a prototype implementation indispensable part of this work. His pragmatic concerns encouraged me to spend significant part of my time on writing systems software that would verify several aspects of the proposed design. Under his direction, the Tornado OS group had the patience to introduce me to the art of handling large and complex systems building.

Charlie Clarke offered precious help from the very beginning of this work. Attending his course in information retrieval inspired me to do work on data storage. Demetri Terzopoulos and Peter Marbach were valuable members of my committee that provided useful feedback at several checkpoints of my thesis.

Garth Gibson honored this work through his participation in the final exam as external appraiser. His insightful comments helped several ideas to become clearer.

My research internship at the Storage Systems Program of HP Labs, under the mentorship of Arif Merchant and management of John Wilkes, deepened my understanding of disk storage issues and renewed my interest for completing this thesis.

Kathy Yen, Joan Allen and Linda Chow with their smile and keen administrative assistance helped overcome several problems in time-constrained situations.

Many thanks to my friends in Toronto and other parts of the world for all the times we shared.

My family in Greece never stopped encouraging and supporting me throughout the difficulties that this thesis involved.

Contents

Chapter 1

Introduction

1.1 Problem Context

Efficient spatial and temporal compression schemes, that reduce redundant information within individual or between consecutive video frames, make practical the manipulation of digital video with acceptable quality. Standard encoding specifications (e.g. Moving Pictures Experts Group) facilitate widespread distribution and use of compressed video content in a range of applications from studio post-production editing to home entertainment (e.g. Digital Versatile Disks). Nevertheless, online access to video streams remains a challenging task. Predicted advances in optical and wireless telecommunication technology are only expected to increase the need for network servers with the capacity to support large numbers of users accessing collections of stored digital video (Gilder 1997; Gray 2000).

Although video streams can optionally be encoded with constant bit rates, it has been shown that equally acceptable quality can be achieved using variable bit-rate encoding at lower average rates (Tan et al. 1991; Gringeri et al. 1998; Lakshman et al. 1998). Supporting bit-rate variability can actually reduce the resource requirements in the entire path from the disks to the client buffer. However, it also requires tight resource reservations over time which comes at the cost of increased system complexity (Makaroff et al. 1997). For the sake of simplicity, most existing video servers reserve resources according to the maximum bit rate required for the retrieval of each stream. Therefore, it would be interesting to investigate alternative more

efficient methods for supporting variable bit-rate streams, and to evaluate the potential benefits that can be achieved in large scale systems.

Instead of storing an entire stream on a single disk, it is possible to distribute the data of each stream across multiple disks. Thus, the utilization of the resources can be better balanced, and also the system operation can become more reliable. Assuming that a media server supports requests for several different streams, appropriate data distribution makes it possible to scale the number of concurrent playbacks to the limit of the server resources. This can be achieved independently of the particular streams being requested by the clients. It becomes possible by retrieving different parts of each stream from different disks, thus restricting the degree of load imbalance among them. Replication of data across different devices also allows tolerance of hardware failures without interrupting the operation of the system.

Multiplexing the resource requests of concurrently served streams can lead to reduced aggregate resource requirements in the system. In the case of supporting variable bit-rate streams, additional performance improvement can be achieved by applying prefetching techniques. Peaks in the required transfer bandwidth of individual streams can be removed if the data are transferred in advance of when they are actually required. Prefetched data are temporarily kept in buffer memory, located in the path from the disks to the client, and the bandwidth utilization of the disk and network links is improved.

Existing experience with streaming media and other related services suggests several quality of service parameters related to the expectations of the users, in addition to the uninterrupted stream decoding at the client. Whether accessing streams over the Internet or through privately owned installations, the playback initiation latency should be predictable and limited. Also the probability of a user request being rejected due to system overloads should be minimized. Possible control operations ultimately should include the flexibility found in traditional video playback devices (VCRs).

In web and other online services, service providers strive to maximize the utilization of the available resources, and prefer reliable systems with minimal downtime and revenue loss. Software for simplifying system management is expected to keep the system operation cost-effective with minimal human intervention. Finally, it is advantageous to scale the system

installation according to increasing user demands, and to be able to efficiently expand currently available resources with new state-of-the-art hardware equipment.

1.2 Thesis Content

Our central goal in this thesis is to investigate the cost-effective support of variable bit-rate streams in continuous media servers. We set as basic objective to maximize the sustained number of stream playbacks, while keeping limited the waiting time of the users and the rejection ratio of playback requests. In addition, we would like to allow the system size to scale efficiently over time as service demand increases.

We present a high-level view of a distributed video server architecture. Then, we examine several important issues from our experience with the design and implementation of a prototype system. The prototype supports admission control of playback requests, and striping of variable bit-rate streams over multiple disks attached to a single computer node. We describe design alternatives regarding the buffer and disk storage space contiguity, the structuring of the metadata corresponding to each disk, and the scheduling of playback requests and disk transfers. We use performance measurements with synthetic benchmarks and actual variable bit rate MPEG-2 streams over SCSI disks in order to demonstrate the performance advantages of our decisions.

Previous analytical and simulation studies have already demonstrated potential advantages from striping variable bit-rate streams over multiple disks with statistical quality of service guarantees. However, we are not aware of any actual implementation that demonstrates support of the above features, especially with deterministic quality of service guarantees. As we see in later chapters, designing for efficient access of variable bit-rate streams affects the structure of the system in fundamental ways, and cannot be considered as an add-on feature that can be included later.

Previous studies focusing on particular striping policies found that both disk load imbalance and disk overhead cause the stream striping to be efficient only on disk arrays of limited size. We experiment with several striping policies that differ in the amount and variability of the data

stored on each disk. We show an almost linear scaling of the sustained number of supported streams as a function of the number of disks. Our conclusions differ from earlier ones in part due to the performance evaluation method that we use and the resource management policies that we introduce. We also find differences in the striping policies to have a significant effect to the throughput of the system.

Recent emergence of mass-produced specialized devices for data access applications means that we cannot always assume abundant computing resources at the client side. In this thesis we show that disk striping becomes more efficient with a priori knowledge of the retrieval process. Optimizing for disk bandwidth is critical because disk bandwidth increases an order of magnitude slower than network link bandwidth (Gray and Shenoy 2000). For all these reasons, it is important to investigate stream data prefetching techniques that can improve disk bandwidth utilization, depend on a known server configuration, and make minimal assumptions about the client resources.

We introduce a stream smoothing algorithm that prefetches data into server buffers in order to reduce peaks in the disk bandwidth requirements of individual streams. It accepts as input a specification of the data amount that should be sent to the client over time. In order to prevent excessive smoothing from exhausting the available buffer space, the smoothing process is automatically adjusted according to the total resources available in the server configuration. Server-based smoothing deals with disk bandwidth and does not alleviate potential network utilization problems. Thus, its operation can be complemented with network smoothing techniques. Such techniques prefetch stream data into client buffers, where sufficient client buffer space can be assumed, in order to reduce peaks in both network and disk bandwidth requirements.

Traditionally, load-balancing and reliability problems restricted the size of disk arrays across which stream data could be striped efficiently. More recently, disk striping scalability of stream data has been demonstrated for both constant and variable bit-rate streams (Bolosky et al. 1996; Anastasiadis et al. 2001a). In addition, fast technological improvements make attractive the incremental expansion and support of heterogeneity in a system during load increases. In the present study, we use our smoothing algorithm for striping variable bit-rate streams across heterogeneous disks, treating bandwidth as the key disk resource that should be fully utilized.

1.3 Contributions

The most important contributions of this thesis are the following:

- We demonstrate the feasibility of building media servers that efficiently support variable bit-rate streams striped across multiple disks with deterministic quality of service guarantees.
- We describe efficient policies for allocating the disk space, organizing the memory buffers, structuring the stream metadata, and scheduling the playback requests and disk transfers.
- We introduce a general method for evaluating media server performance. Its flexible definition makes it equally applicable to different system scales and stream characteristics. It proved an indispensable tool for conducting our experiments and unifying our results.
- We formally describe previous and new disk striping policies for variable bit-rate streams. We experimentally compare them and find significant performance differences, depending on the variability and the amount of data stored on each disk. We show that the system throughput increases almost linearly as a function of the numbers of disks.
- We introduce a new smoothing algorithm for variable bit-rate streams that uses buffer space available at the server side. We show that it works correctly under the specified assumptions.
- Application of the smoothing algorithm to MPEG-2 streams demonstrates significant performance benefits when using homogeneous disks. A straightforward extension is shown to achieve high disk bandwidth utilization over heterogeneous sets of disks as well.

1.4 Organization

The remainder of this thesis is organized in the following way:

In **Chapter 2**, we review previous research related to building media server systems, striping variable bit-rate streams across multiple disks, and smoothing stream transfers for improved network and disk bandwidth utilization.

In **Chapter 3**, we describe the basic components of the distributed media server architecture that we propose, define alternative disk striping policies for variable bit-rate streams, and introduce a new disk space allocation technique.

In **Chapter 4**, we present the most important modules of our prototype and describe design alternatives related to several resource management policies.

In **Chapter 5**, we specify the features of our experimentation platform, and the characteristics of our stream workload. We also evaluate the effect that important parameters have on the performance and the resource requirements of the system.

In **Chapter 6**, we investigate important parameters of the admission control process. We compare the efficiency of different disk striping policies, and study the scalability of their performance with respect to the number of disks.

In **Chapter 7**, we introduce the server smoothing algorithm, and evaluate its potential benefit on homogeneous and heterogeneous disks. We demonstrate that it successfully balances the load of the system across different rounds and disks, while keeping under control the buffer space required.

In **Chapter 8**, we discuss system operation issues related to tolerating hardware component failures, resequencing stream data at the client, restriping data after system upgrades, and reducing the cost of admission control.

In **Chapter 9**, we summarize the basic conclusions of this thesis, and outline open issues for further investigation.

Chapter 2

Related Research

A significant amount of research effort has been recently devoted to the design and development of media streaming servers. However, most of the existing experimental and commercial systems can only support constant bit-rate streams (Berson et al. 1994; Buddhikot and Parulkar 1995; Bolosky et al. 1996; Ozden et al. 1996). Alternatively, they retrieve variable bit-rate streams using peak-rate resource reservations that may reduce resource utilization, but not increase server capacity (Martin et al. 1996; Shenoy et al. 1998). Another possibility is to support variable bit-rate streams with statistical quality-of-service guarantees, that allow the system to occasionally be overloaded and discard transferred data (Muntz et al. 1998). On the other hand, the approach of retrieving variable bit-rate streams using constant bit rates cannot solve the general efficiency problem either, due to arbitrarily large playback initiation latency or client buffer space that it can require with higher quality streams (Sen et al. 1997).

In the rest of this chapter, we examine basic features of media server architectures that have been proposed until now. We explain previous policies and comparative studies for striping variable bit-rate streams across multiple disks. We also outline earlier data prefetching techniques for improving disk or network bandwidth utilization. Finally, we summarize past efforts for obtaining balanced load when accessing data from disks with different performance specifications.

2.1 Media Server System Design

The Tiger fileserver is a distributed video server architecture that supports constant bit-rate streams on commodity computer hardware (Bolosky et al. 1996). Stream data are striped across multiple disks attached to different computer nodes. Data block replicas are declustered on one or more disks in order to provide uninterrupted streaming operation even in the case that disks or entire nodes fail. Data retrieval scheduling information is distributed among the different nodes.

In the Fellini storage system by Martin et al. stream data are stored on multiple disks (Martin et al. 1996). Accepted clients have to submit explicit requests over time for accessing the retrieved data, according to the client-pull model. The resource reservation is based on the worst case requirements of each stream. Careful data replication keeps the load balanced across different disks even in the case of disk failures. Data caching with efficient page replacement is implemented for minimizing disk accesses during sequential playback.

In the Continuous Media File Server proposed by Neufeld et al., detailed resource reservation is done over time for each accepted stream (Neufeld et al. 1996). The experimental study is limited to the case that an entire stream is stored on a single disk. Techniques of data prefetching are considered for preventing system overloads and improving throughput. Prematurely retrieved data are discarded when extra buffer space is required for admitting new playback requests (Makaroff et al. 1997).

The Symphony multimedia file system by Shenoy et al. emphasizes integration on the same platform of different data types with real-time or best-effort requirements (Shenoy et al. 1998). In order to achieve that, appropriate methods are described for disk scheduling, failure recovery, data placement and caching. Experimental evaluation shows good response time for text data requests combined with real-time operation for video data requests.

The RIO storage system by Muntz et al. is designed to handle several different data types, including video streams (Muntz et al. 1998). The admission control is based on statistical estimation of the workload requirements, and the stream blocks are randomly distributed across different disks. Stream data replication is shown to improve load balancing in the system. An

early design of distributed data striping is described by Cabrera and Long (Cabrera and Long 1991). Their data striping and resource reservation policies do not take into account special requirements of variable bit-rate streams though. In addition, 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 types by Birrel and Needham (Birrel and Needham 1980). In describing the design of the Fast File System, McKusick et al. argue that chaining together kernel buffers would allow accessing contiguous blocks in a single disk transaction, and more than double the disk throughput (McKusick et al. 1984). At that time, throughput was limited by processor speed though, and changes of the device drivers were also necessary for adding this feature. We show later how contiguously allocated buffers can lead to performance improvement in a media storage system.

In extent-based file systems, disk space is allocated in large, physically contiguous chunks, called extents. Most I/O is done in units of an extent. Alternative versions of this approach are possible that differ depending on whether users are exposed to the choice of the extent size (McVoy and Kleiman 1991). Shenoy et al. proposed disk space allocation that supports different block sizes for I/O transfers. A range of minimum and maximum block sizes must be specified during file system creation (Shenoy et al. 1998). Another allocation scheme retrieved a variable number of noncontiguous fixed-size blocks for each data request (Chang and Zakhor 1996). Separate disk-head movements were required for each individual block transfer, however.

2.2 Disk Striping of Stream Data

In a simulation study of constant rate streams, Paek et al. consider striping the stream data of one round across different numbers of disks (Paek et al. 1995). They show that disk efficiency is maximized when only one disk is accessed for a stream during each round, but at the cost of reduced system responsiveness. The above conclusion also holds for streams with multi-layer encoding, where subsets of a stream can be retrieved for lower resolution decoding. In a later paper, they describe a technique for reducing server buffer requirements with variable bit-rate

streams (Paek and Chang 1996).

Chang and Zakhor, in their study for multi-layer encoded streams, describe *non-periodic interleaving*, where an entire stream is split into parts equal to the number of disks and each part is stored on a separate disk (Chang and Zakhor 1994). They also describe *periodic interleaving*, where stream data for a round are stored on a single disk that changes round-robin every round. They find that although periodic interleaving does not improve the number of streams supported, it can lead to better system responsiveness. In later work, they show that periodic interleaving improves both the number of users and the system responsiveness when compared to a striping scheme where stream data for a round are striped across a fixed number (greater than one) of disks (Chang and Zakhor 1997). In a different study for variable bit-rate streams, Chang and Zakhor 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 (Chang and Zakhor 1996).

Shenoy and Vin study the fixed-size block striping of stream data, with both analytical and simulation methods (Shenoy and Vin 1997). 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. From their paper it is not clear 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 study the fixed-block striping technique called *Constant Data Length* (CDL), and the *Block-constrained Constant Time Length* (BCTL) technique, where stream data of each round are stored on a single disk (that changes round-robin every round) in multiples of a fixed (possibly large) block size (Reddy and Wijayaratne 1999). They compare the throughput of the two techniques using eight disks and conclude that the improvement of

BCTL (at large block sizes) is insignificant, which is due to the particular block constraint they assumed.

In a study of single disk systems storing constant bit-rate streams, Triantafillou and Harizopoulos propose a technique called *Group Periodic Multi-Round Prefetching*, where multiple rounds worth of data are retrieved in a single round (Triantafillou and Harizopoulos 1999). Instead, Biersack et al. introduce the *Generalized Constant Data Length* for single disk systems storing variable bit-rate streams. With this scheme, the retrieval of a fixed amount of data is completed in a fixed number of rounds (Biersack et al. 1996).

2.3 Smoothing of Variable Bit-Rate Streams

Salehi et al. describe a prefetching algorithm that minimizes the variability of network bandwidth requirements of stored video streams during their playback (Salehi et al. 1996). The algorithm is shown to have optimal smoothing effect assuming a fixed-size client buffer. It is applied to individual streams with known data retrieval sequence. Experiments with concurrently served streams demonstrate improved network bandwidth utilization.

Feng and Rexford compare the previous algorithm with alternative techniques that minimize the total number of network bandwidth increases or decreases (Feng and Rexford 1997). They find interesting tradeoffs that would make the different approaches favorable depending on the network architecture assumed. McManus and Ross introduce a general dynamic programming methodology for applying several optimization objectives in scheduling network transfers of stored video (McManus and Ross 1998). Zhao and Tripathi describe a class of algorithms that minimize the maximum required network bandwidth, when multiplexing stream network transfers to multiple clients (Zhao and Tripathi 1999).

Rexford et al. use prefetching of data at the client for smoothing live video streams (Rexford et al. 2000). They consider the additional restriction that the stream resource requirements are only known for a limited period instead of the entire playback period. Peak network bandwidth is reduced when limited playback delay can be tolerated and moderate client buffer space is available.

Mansour et al. examine optimal tradeoffs between buffer space, playback delay and link bandwidth for lossy smoothing of live video (Mansour et al. 2000). They consider online algorithms for determining which part of the stream to drop when different data have different importance during decoding. Sen et al. combine ideas from live video smoothing with caching of file prefices for smoothing streams in network proxies (Sen et al. 1999). They assume that proxy servers can have extra knowledge of quality of service parameters about the client not usually available at the server.

Patterson et al. apply a cost-benefit analysis in order to control the disk access versus data buffering tradeoff in general applications (Patterson et al. 1995). Applications can disclose knowledge of future accesses, while the system estimates the relative values of caching and prefetching disk blocks. A global algorithm is applied for maximizing the global usefulness of every buffer. Experiments with executing several different applications show reduced elapsed time and improved throughput.

Paek and Chang propose an approach that, given a set of streams, optimizes a "general objective function" by allocating the appropriate maximum disk bandwidth and buffer space to each stream (Paek and Chang 1996). They make online decisions without taking into account constraints from data distribution across multiple disks.

Reddy and Wijayaratne have experimented with the effect of client-based smoothing on alternative disk striping methods. They point out the need for also studying techniques of prefetching data into server buffers in their future work (Reddy and Wijayaratne 1999). Other proposed solutions for reducing the maximum needed bandwidth of a variable bit-rate stream require that an arbitrary amount of data be retrieved in the server buffer before playback initiation. This can lead to reduced system responsiveness, however (Biersack and Hamdi 1998; Lee and Yeom 1999).

Sen at al. consider the case of transmitting variable bit-rate video using constant bit rate (Sen et al. 1997). They describe an algorithm for computing the minimum required client buffer size and the associated constant rate and playback initiation latency. They evaluate the actual buffer space and latency requirements for reducing the transfer rate using MPEG-1 streams.

Although Kim et al. (Kim et al. 1999) outline some ideas on how to control the tradeoff be-

tween buffer and disk bandwidth utilization in stored video streaming, they specify no concrete algorithm for the problem. Their approach divides the stream into arbitrary length segments according to an "empirical threshold" alpha. Prefetching is limited to within a segment, and it is controlled by an "empirical threshold" beta. Their disk bandwidth definition ignores the disk arm movement delays and the round duration, and their simulation study is limited to single disk systems only.

2.4 Load Balancing across Heterogeneous Disks

Dan and Sitaram suggest that multiple heterogeneous storage devices may coexist in a video server environment (Dan and Sitaram 1995). Considering the complexity of striping data across all the devices, they propose clustering homogeneous devices into groups. Subsequently, they describe a dynamic data placement policy to keep the bandwidth and storage space utilization high. For that purpose, entire video objects are appropriately replicated depending on their popularity.

Chou et al. also propose techniques for dynamic video object replication across different groups of disks. Based on a mathematical model of user behavior, the system adapts to frequent changes in user preferences for particular objects (Chou et al. 1999).

Santos and Muntz use randomized data replication techniques for load balancing of heterogeneous disk arrays, assuming that streams are striped across different disks (Santos and Muntz 1998). Other studies try to achieve high utilization of heterogeneous disks by appropriately grouping them and adjusting the amount of data stored on each of them (Zimmermann and Ghandeharizadeh 1997; Wang and Du 1997). All these methods are applicable (or have been demonstrated to work) only with constant bit-rate streams.

Chapter 3

Architectural Definitions

In the present chapter, we describe the media server architecture that we propose. In addition, we define alternative policies for disk striping of stream data, and introduce a new disk space allocation technique.

3.1 System Overview

We describe a distributed media server architecture that stores video streams on multiple disks. The streams are compressed according to the MPEG-2 specification, or any other encoding scheme that supports constant quality quantization parameters and variable bit rates. Clients with appropriate stream decoding capability send playback requests and receive stream data via a high-speed network, as shown in Fig. 3.1.

We assume that the system operates using the server-push model (Shenoy et al. 1995). 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. This is different from the client-pull model of traditional file servers, where each individual data transfer is explicitly requested by the client. The server-push model reduces the control traffic from the client to the server, and facilitates resource reservation at the server side, thus ensuring timely transfer of the data to the client.

We also assume that data transfers occur in rounds of fixed duration T_{round}. In each

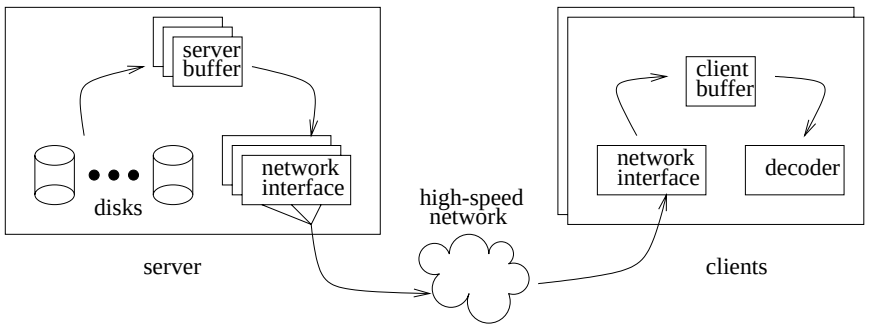


Figure 3.1: Compressed video streams are stored across multiple disks of the media server. Multiple clients can connect and start playback sessions via a high-speed network.

round an appropriate amount of data is retrieved from the disks into the server buffers for each active client. Concurrently, data already residing in the server buffers are sent to the client through the network interfaces. Round-based operation is used in media servers in order to keep the reservation of the resources and the scheduling-related bookkeeping of the data transfers manageable.

Disk transfer requests are submitted synchronously at the beginning of each round, and are kept sorted according to the location of their initial block. By serving requests in this order (according to a Circular SCAN policy), seek and rotation overhead during actual service is reduced. The round-based operation can be considered as a special case of the more traditional real-time operation with periodically scheduled tasks. The data transfers are the periodic tasks, while the beginning and end of each round correspond to the starting time and deadline of the tasks.

The large amount of network bandwidth required for this kind of service requires that the server components be connected to the high-speed network through different network interfaces. Transfer of data to the client essentially requires a multipoint-to-point network connection between the multiple interfaces of the server and the one of the client. This is basically a network signaling problem already reported by previous related studies (Bolosky et al. 1996).

The amount of stream data periodically sent to the client is determined by the decoding frame rate of the stream and the resource management policy of the network. One reasonable policy would send to the client during each round the amount of data that will be needed for

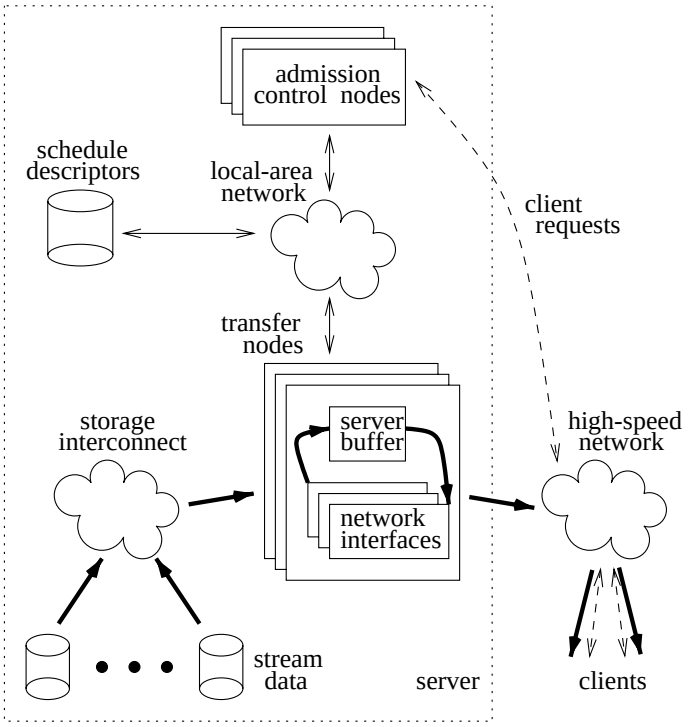


Figure 3.2: In the *Exedra* media server architecture, stream data are retrieved from the disks and sent to the clients through the Transfer Nodes. Both the admission control and the data transfers make use of stream scheduling information maintained as a set of Schedule Descriptors.

the decoding process during the next round; any other policy that does not violate the timing requirements and buffering constraints of the decoding client would also be acceptable. As we shall see later, in some cases there are advantages to using policies other than this basic one.

3.2 The *Exedra* Media Server Architecture

In this section, we describe the basic features of the media storage server that we propose.[1] The design is based on standard off-the-self components for data storage and transfer, currently used in server systems. One basic innovation of the design lies in the scalable resource management of variable bit-rate streams, as is demonstrated in subsequent chapters.

[1]Exedra: architectural term meaning semicircular or rectangular niche (orig. Greek).

3.2.1 Transfer Node

The stream data are stored across multiple disks. As shown in Figure 3.2, every disk is connected to a particular *Transfer Node*, through the *Storage Interconnect*, which could be either i) a standard I/O channel e.g. Small Computer System Interface, ii) standard network storage equipment e.g. Fibre-Channel (Clark 1999), or iii) a general purpose network, as with Network-Attached Secure Disks (Gibson et al. 1998). Recent research has demonstrated that it is possible to offload file server functionality to network-attached disks (Gibson et al. 1998). Although our design could be extended in a similar way, we leave the study of this issue for future work.

The Transfer Nodes are computers responsible for scheduling and initiating all data accesses from the attached disks. Data arriving from the disks are temporarily staged in the *Server Buffer* memory of the Transfer Node before being sent to the client through the *High-Speed Network*. We assume that the system bus bandwidth is a critical resource within each Transfer Node, which determines the number and capacity of attached network or I/O channel interfaces. For example, the Peripheral Component Interconnect (PCI) is becoming the dominant system bus architecture in both desktop and server systems. Current specifications can achieve burst transfer rates between 132Mbytes/sec and 524Mbytes/sec depending on the clock rate (33 or 66MHz) and the data bus width (32 or 64 bits) (PCI Special Interest Group 1995). The corresponding sustained rates can be as low as half of the maxima though, depending on how the hardware implementation of the interfaces takes advantage of the burst transfer features of the bus specification.

In addition, Asynchronous Transfer Mode (ATM) network equipment can be used for stream data transfer with quality of service guarantees from the server to the clients (Bolosky et al. 1996). Although, the ATM network interfaces most commonly used today support 155Mbit/sec, interfaces with transfer capacity 622Mbit/sec have become recently available for the 33/66MHz PCI bus (FORE Systems, Inc. 1999).

3.2.2 Admission Control Node

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

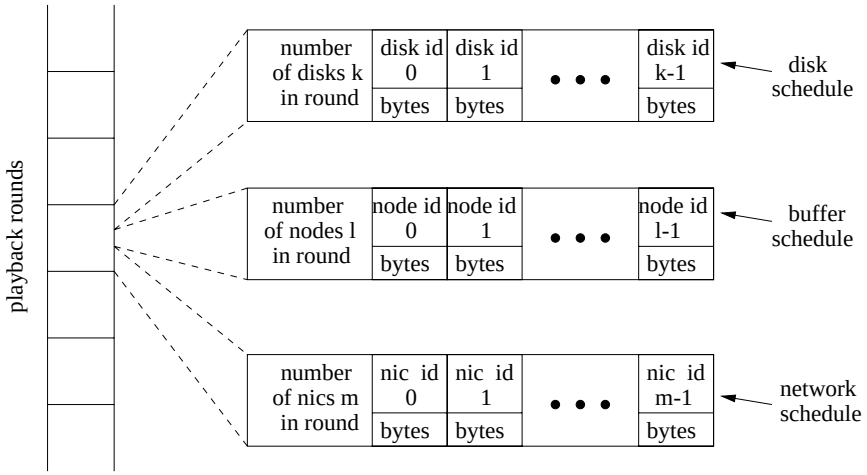


Figure 3.3: Stream schedule descriptor for a stream stored on system with multiple transfer nodes. For each playback round, the number of disks accessed is specified, along with the identifier of each disk and the corresponding amount of data that have to be transferred from each disk. In addition, the descriptor includes the buffer space reserved on each node, and the amount of data that has to be sent to the client through each network interface that is used in the current round.

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).

We make the practical assumption that the acceptable initiation latency is limited, and use a simpler scheduling algorithm with complexity linear in the number of rounds of each stream and the number of reserved resources. Depending on the expected load and the required detail of resource reservation, the admission control process might still become a bottleneck. In that case, the admission control could be distributed across multiple nodes as shown in Figure 3.2, taking into account non-trivial concurrency control issues that arise.

If a new playback request is accepted, commands are sent to the Transfer Nodes to begin the appropriate data accesses and transfers. We assume that local-area network technology is sufficient for handling the moderate control traffic between Admission Control and Transfer Nodes.

3.2.3 Schedule Descriptors

In traditional storage systems, data access patterns are relatively hard to predict, making it difficult to determine optimal disk striping parameters, customized to the needs of a constantly changing workload and system configuration (Borosky et al. 1997). However, with read-only sequential access being the common case in video streaming, it is possible to predict to some extent the expected system load requirements during retrieval. Then appropriate disk striping parameters can be determined a priori for the storage and retrieval of the data (Shenoy et al. 1995). In a later section, we see how different striping policies exploit this characteristic of stored video streams.

The amount of stream data that needs to be retrieved during each round from each disk is stored in a *Schedule Descriptor* (Figure 3.3). The descriptor also specifies the buffer space required and the amount of data sent to the client by the Transfer Nodes during each round. It is possible that two or more schedule descriptors with distinct requirements are available for a stream. The scheduling information is generated before a stream is first stored and is used for both admission control and for controlling data transfers during playback. Since this information changes infrequently, it can be replicated to avoid potential bottlenecks.

3.3 Stride-Based Disk Space Allocation

In our experiments, we use a new form of disk space allocation, called *stride-based allocation*. Thus, disk space is allocated in large, fixed-sized chunks called *strides*, which are chosen larger than the maximum stream request size per disk during a round. Stored streams are accessed sequentially according to a predefined (albeit variable) rate; therefore, the maximum amount of data accessed from a disk during a round for a stream is known a priori. 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 (see Fig. 3.4).

Although stride-based allocation seems similar to extent-based (McVoy and Kleiman 1991) and other allocation methods (Shenoy et al. 1998), one basic difference is that strides have

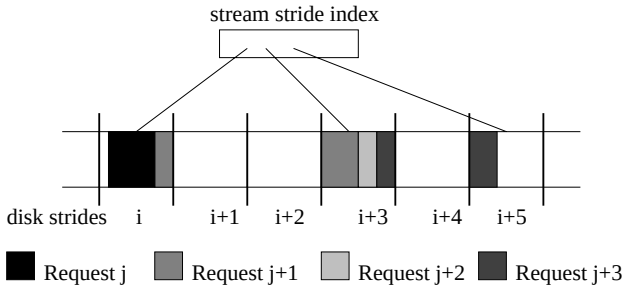


Figure 3.4: The stride-based allocation of disk space shown on one disk. A stream is stored in a sequence of generally non-consecutive fixed-size strides with a stride possibly containing data of more than one round. Sequential requests of one round are smaller than the stride size and thus require at most two partial stride accesses.

fixed size. More importantly, when a stream is retrieved, only the requested amount of data is fetched to memory, and not the entire stride, which is sequentially allocated on the disk surfaces. 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 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, instead of two at most. Contiguous disk space allocation was the favorite approach in file systems of early microcomputers. Due to external fragmentation, it requires periodic compaction of the allocated disk space, which is a time-consuming process. In our case, we assume that variable disk data transfers can be specified in a granularity of a few tens of KBytes, while the total used disk storage space can reach hundreds of GBytes. Therefore, the time and space cost involved in bookkeeping for contiguous disk space allocation in highly utilized disks could become significant. With stride-based allocation we get most of the benefits of contiguous allocation, while avoiding its extra overhead.

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. Moreover, we don't expect that this would improve the throughput of the system significantly. Due to the large size of the disk requests, most of the busy time of each disk is spent on useful data transfers rather than mechanical movement overhead. We quantitatively investigate the efficiency of the stride-based allocation in the next chapter.

3.4 Reservation of Server Resources

We consider a system consisting of N network interfaces, D disks, and Q transfer nodes. It is possible that two or more replicas are available for each stream file, with different resource reservation sequences.

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

We assume that data are stored 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. After taking into account logical block quantization issues, the disk sequence $\mathbf{S_d}$ can be derived from the network sequence $\mathbf{S_n}$ as follows: If

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

specifies the cumulative number of blocks B_l retrieved through round i, then

$$S_d(i) = (K^d(i) - K^d(i-1)) \cdot B_l.$$

The *Disk Striping Sequence* $\mathbf{S_{md}}$ of length L_d determines the amount of data $S_{md}(i,k)$, that are retrieved from the disk k, $0 \leq k \leq D-1$, in round i, $0 \leq i \leq L_d - 1$. It can be generated from the Disk Sequence $\mathbf{S_d}$, 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}. The stride-based disk space allocation policy enforces an upper bound of at most two disk arm movements per disk for each client per round. The total seek distance can also be limited using a CSCAN (circular scan) disk scheduling policy. Let M_i be the number of active streams during round i of the system operation. We assume that the playback of stream j, $1 \leq j \leq M_i$, has been initiated at round l_j of system operation. Then, the total access time on disk k in round i of the system operation will have an upper-bound of:

$$T_{disk}(i,k) = 2T^k_{fullSeek} + 2M_i \cdot (T^k_{trackSeek} + T^k_{avgrot}) + \sum_{j=1}^{M_i} S^j_{md}(i - l_j, k)/R^k_{disk} \tag{3.1}$$

where $\mathbf{S^j_{md}}$ is the disk striping sequence of client j. $T^k_{fullSeek}$ is counted twice due to the disk arm movement from the CSCAN policy, while the factor two of the second term is due to the stride-based method. The first term should be accounted for only once in the disk time reservation structure of each disk. Then, each client j can be assumed to incur an additional maximum access time of

$$T^j_{disk}(i,k) = 2 \cdot (T^k_{trackSeek} + T^k_{avgRot}) + S^j_{md}(i - l_j, k)/R^k_{disk}$$

on disk k during round i, when $S^j_{md}(i - l_j, k) > 0$, and zero otherwise.

If R^u_{net} is the bandwidth available at network interface u, then the corresponding network transmission time reserved for client j in round i becomes $T^j_{net}(i,u) = S^j_{mn}(i - l_j, u)/R^u_{net}$, where $\mathbf{S_{mn}}$ is the Network Striping Sequence of client j. The buffer space at transfer node q reserved for client j in round i becomes $B^j(i,q) = S^j_{mb}(i - l_j, q)$, where $\mathbf{S_b}$ is the Buffer Striping Sequence of client j.

The Disk, Network and Buffer Striping Sequence of each stream are available through the corresponding schedule descriptor (Figure 3.3).

3.5 Definition of Striping Techniques

3.5.1 Fixed-Grain Striping

In the method called *Fixed-Grain Striping*, data are striped round-robin across the disks in blocks of a fixed size B_f, a multiple of the logical block size B_l defined previously. During each round a number of these blocks are accessed from each disk as needed. This is shown in Fig. 3.5.(a). We define as stripe any sequence of D consecutive blocks each of size B_f, with the first block of the sequence stored on disk 0 of the array. The $_{modD}$ notation denotes the remainder of the division by D, and the $_{divD}$ symbolizes the quotient of the division by D. We assume that

$$K^f(i) = \lceil \frac{\sum_{0 \le j \le i} S_d(j)}{B_f} \rceil,$$

specifies the cumulative number of blocks B_f retrieved through round i for a specific client. When $K^f_{divD}(i) - K^f_{divD}(i-1) = 0$, all blocks accessed for the client during round i lie on the same stripe of blocks. Then, the striping sequence $\mathbf{S^f_{md}}$ is equal to:

$$S^f_{md}(i,k) = D^f_0(i,k) \cdot B_f \tag{3.2}$$

where

$$D^f_0(i,k) = \begin{cases} 1, & \text{if } K^f_{modD}(i-1) < k_{modD} \le K^f_{modD}(i) \\ 0, & \text{otherwise,} \end{cases}$$

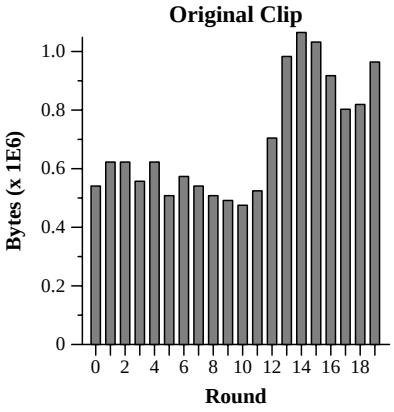
specifies the particular disks that need to be accessed at most once for the stream.

Respectively, when $K^f_{divD}(i) - K^f_{divD}(i-1) > 0$, the blocks accessed for the client during round i lie on more than one stripe. Then, the striping sequence becomes

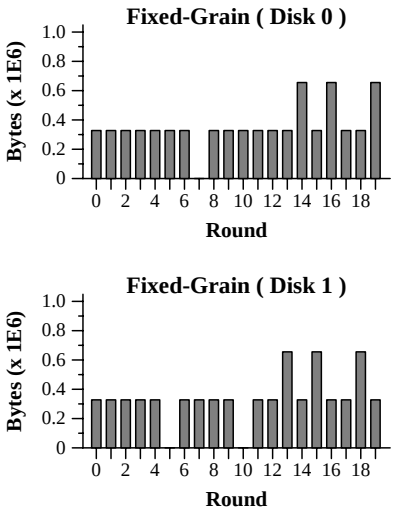
$$S^f_{md}(i,k) = (K^f_{divD}(i) - K^f_{divD}(i-1) - 1) \cdot B_f + D^f_{>0}(i,k) \cdot B_f, \tag{3.3}$$

where

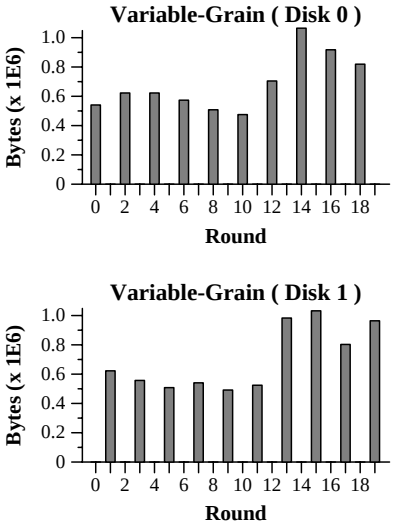
$$D^f_{>0}(i,k) = \begin{cases} 2, & \text{if } K^f_{modD}(i-1) < k_{modD} \le K^f_{modD}(i) \\ 1, & \text{if } k_{modD} > max(K^f_{modD}(i-1), K^f_{modD}(i)\ ) \\ 1, & \text{if } k_{modD} \le min(K^f_{modD}(i-1),\ K^f_{modD}(i)) \\ 0, & \text{otherwise.} \end{cases}$$



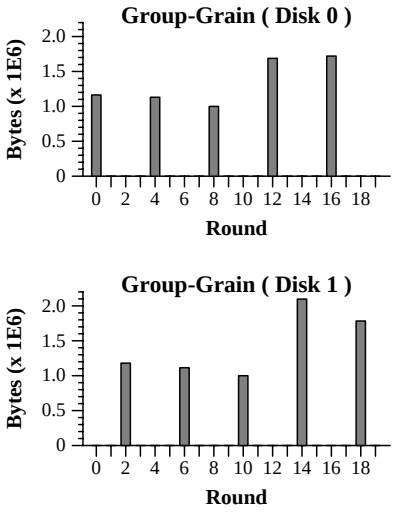
(a) Required Data Transfers



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



(c) Variable-Grain Striping



(d) Group-Grain (G=2)

Figure 3.5: Access of stream data using alternative striping techniques over two disks. Figure (a) shows the data requirements of twenty consecutive rounds in an MPEG-2 clip. With Fixed-Grain Striping, the needed blocks of size B_f are retrieved round-robin from the two disks every round. In Variable-Grain Striping, a different disk is accessed in each round, according to the byte requirements of the original clip. In Group-Grain Striping with G=2, stream data worth of two rounds are accessed from a different disk every two rounds.

Intuitively, blocks in stripes that are fully accessed in round i are taken into account by the first term in Eq. 3.3, while blocks of stripes partially accessed in round i are covered by the second term.

3.5.2 Variable-Grain Striping

In the method that we call *Variable-Grain Striping*, the data retrieved during a round for a client are always accessed from a single disk round-robin, as shown in Fig. 3.5.(b). The corresponding striping sequence becomes:

$$S^v_{md}(i,k) = (K^v(i) - K^v(i-1)) \cdot B_l$$

when $i \pmod D = k$, with

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

Therefore, the Network Sequence through the Disk Sequence determines the particular single disk accessed and the exact amount of data retrieved during each round.

3.5.3 Group-Grain Striping

Variable-Grain Striping is a special case (with $G = 1$) of a method that we call *Group-Grain Striping*, where the amount of data required by a client over G rounds is retrieved every Gth round from one disk that changes round-robin (see Fig. 3.5.(c), noting that the y-axis uses a different scale). The parameter G, $G \ge 1$, is called *Group Size*. The striping sequence for Group-Grain Striping is equal to:

$$S^g_{md}(i,k) = (K^v(i+G-1) - K^v(i-1)) \cdot B_l$$

when $i \; mod \; G = 0 \; AND \; (i \; div \; G) \; mod \; D = k$, and $S^g_{md}(i,k) = 0$ otherwise.

As G increases, fewer disk accesses are required, which leads to reduced disk overhead. On the other hand, the fixed round spacing between subsequent requests for a stream basically divides the server into G virtual servers. The fixed group size G guarantees that two streams started from the same disk at rounds i and j with $i \neq j (mod \; G)$, do not have any disk transfers

in a common round. This is different from increasing B_f in Fixed-Grain Striping, where accesses from different streams can randomly collide on the same disk in the same round, resulting in the system saturating with fewer streams. We show later in more detail how increasing G for a particular round time is advantageous with future expected changes in disk technology.

Although aggregation of disk transfers could also be achieved with an appropriate increase of round time, this could directly affect the responsiveness of the system by potentially increasing the initiation latency of each playback. Longer round time also increases proportionally the required buffer space.

3.6 Summary

In this chapter, we described the basic components of the *Exedra* distributed media server architecture. We introduced the stride-based disk space allocation technique, and specified the scheme that we use for reserving disk time, buffer space and network transfer time in our system. Finally, we formally defined alternative methods for striping stream data over multiple disks.

Chapter 4

Prototype Implementation

We have designed and built a media server prototype, in order to evaluate the resource requirements of alternative stream scheduling techniques. The modules are implemented in about 12,000 lines of C++/ Pthreads code on AIX4.1. The code can be linked to the University of Michigan DiskSim disk simulation package (Ganger et al. 1999), which incorporates advanced features of modern disks, such as on-disk cache and zones, for obtaining disk access time measurements. Alternatively, hardware disks can be accessed directly through their raw interface for full data transfers. The stream indexing metadata are stored in the Unix file system as regular 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 stream blocks address translation, buffer management, and disk and network transfer scheduling (Figure 4.1). With appropriate configuration parameters, the system can operate in several modes involving different levels of detail. In *Admission Control* mode, the system receives playback requests, does admission control and resource reservation, but no actual data transfers take place. In *Simulated Disk* mode, all the modules become functional, and disk request processing takes place using the specified DiskSim disk array (Ganger et al. 1999). In *Full Operation* mode, the system accesses hardware disks and transfers data to clients.

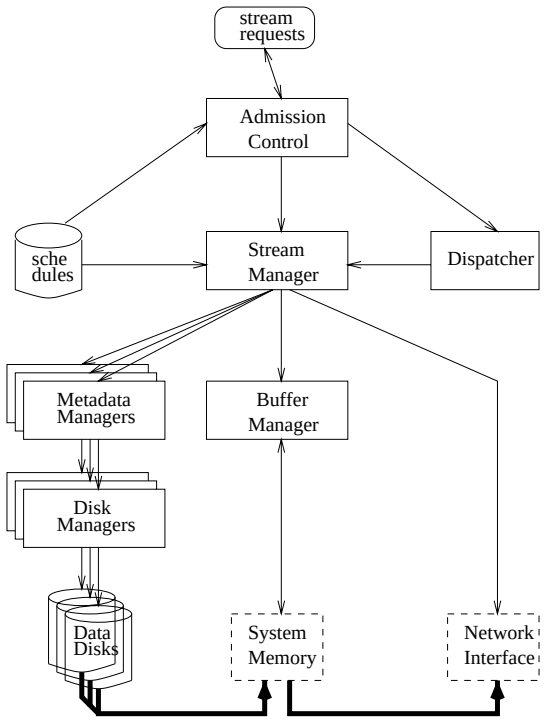


Figure 4.1: System modules in the *Exedra* prototype implementation.

4.1 Admission Control and Dispatching

The admission control module uses circular vectors of sufficient length to represent the allocated disk time, network time, and buffer space, respectively. On system startup, the disk time vectors are initialized to $2 \cdot T_{fullSeek}$, while the network time and buffer space are initially set to zero. When a new stream request arrives, the admission control is performed by checking the requirements of the stream against currently available resources. In particular, the total service time of each disk in any round may not exceed the round duration, the total network service time on each network interface may not exceed the round duration, and 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. In addition, notification records for the accepted request are inserted into dispatch queues (generally residing on each transfer node) at the appropriate offset from the current

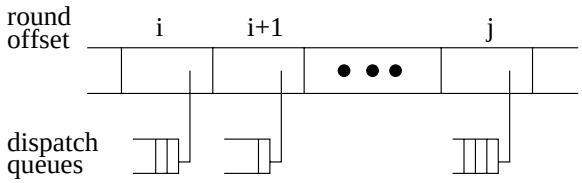


Figure 4.2: A circular vector of dispatch queues keeps track of admitted streams yet to be activated. The dispatch queue consists of notification records for activating the streams in the corresponding rounds.

round. When an upcoming round becomes current, the notification records are used for activating the stream and starting its data transfers (Figure 4.2).

4.2 Stream Scheduling

The stream management module is responsible for initiating all the buffer reservation, disk transfer and network transfer requests for each active stream during each round. The schedule descriptors provide the necessary information about the amount of data and the particular disks that should be accessed for each stream. The required amount of buffer space is allocated, and disk transfer requests are prepared specifying the buffer location, the stream file offset and the length of the transfer. Each request is passed to the lower layers for the actual transfer to occur, while a copy of the request (descriptor) is kept in a fifo queue, to be used for the corresponding network transfers.

The buffer space of each disk transfer is allocated contiguously in the virtual memory, as a sequence of fixed-size blocks. Appropriate control information attached to each individual block is used for completion notification of the initiated data transfers. Subsequently, data can be sent to each client through the network, and the corresponding buffer space can be released. Although the buffer space is allocated in large chunks specified by the disk transfer requests, it is deallocated in the granularity of individual buffer blocks. This provides the necessary flexibility for keeping disk and network transfers relatively independent, a feature that we use when prefetching data into the server memory.

There is also a partial stream playback service, that we use in the Admission Control mode of the system. For this service, resources required by each active stream are reserved, without

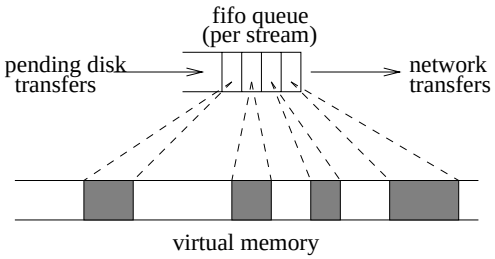


Figure 4.3: Pending disk transfers are gathered in a fifo queue before they complete and allow the corresponding network transfers to begin.

any actual disk request processing taking place. Similarly, there is a partial recording service that does all the necessary disk space allocation and metadata updating, without any actual disk data transfer processing taking place. This has been used for expedited stream "recording".

4.3 Metadata Management

Stream metadata management is organized in a layer above disk scheduling. 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, in the general case, while 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 is used for mapping the identifier of each recorded stream into a direct index of the corresponding allocated strides. A separate directory of this form 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. With the help of the stream index, each such request is translated to a sequence of contiguous disk transfers, each specifying the starting physical block location and the number of blocks. From the stride-based disk space allocation, it follows that each logical request will be translated into at most two physical contiguous disk transfers.

The decision to create a separate metadata manager for each disk was basically motivated by

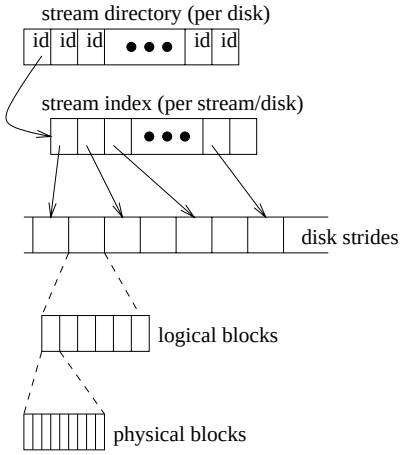


Figure 4.4: Management of metadata within each disk.

our intention to experiment with general disk array organizations, including those consisting of heterogeneous disks. Although handling of heterogeneous devices was less necessary in limited size traditional storage systems, it might prove crucial for the incremental growth and economic survival of large scalable media storage installations. In our prototype implementation, this feature is fully implemented in a relatively straightforward way. However, this functionality does not alleviate the need for developing striping policies that assure high utilization of the server resources.

In order to keep system performance predictable and unbiased by particular disk geometry features, we decided to exercise some control over the disk space allocation pattern. In particular, disk zoning could possibly lead to excessively optimistic or pessimistic data access delays, if we mostly allocated the outer or inner cylinders of the disks. Similarly, contiguous allocation could lead to lower than expected delays in some special cases (such as when streams are stored on a single disk with a very large on-disk cache). However, low-level disk geometry is generally not disclosed by the disk manufacturers, and the above features are not explicitly considered by the system in any sophisticated way. Therefore, when we allocate strides for a stream within each disk, we try to distribute them across all the zones of the disk.

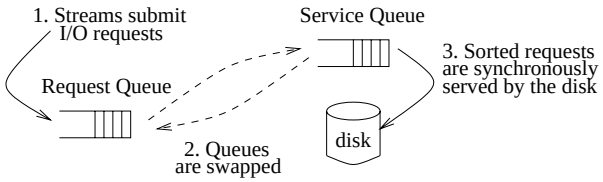


Figure 4.5: Sequence of steps during the disk service of I/O requests using dual-queue CSCAN.

4.4 Disk Scheduling

The disk management layer is responsible for passing data transfer requests to the disks, after the necessary translation from logical stream offsets to physical block locations in the above layers.

In the *dual-queue CSCAN* disk scheduling that we introduce, the operation of each disk is managed by a separate pair of priority queues, called *Request Queue* and *Service Queue*, respectively. The two queues, although structurally equivalent, play different roles during each round. At the beginning of each round, data transfer requests for the current round are added asynchronously into the request queue of each disk, 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 4.5). Subsequently, requests from the service queue are synchronously submitted to the raw disk interface for the corresponding data transfers to occur. There is at most one request pending within one disk at any time.

When swapping the two queues, the service queue becomes request queue and remains empty until the beginning of the next round. Although a single priority queue for each disk would seem sufficient, there is a rare (yet possible) case where the disk busy time in a round slightly exceeds the round duration. Then, with a naive design using a single queue, new incoming requests could postpone (potentially indefinitely) the service of requests from the previous round starting at the innermost edge of a disk. Instead, the two-queue scheme prevents new requests from getting service before those of the previous round complete, thus keeping the system operation more stable.

For the special case of simulated disks, no actual data transfers occur and the requests are active only for disk access time estimation. Requests corresponding to different disks are submitted, processed, and terminate concurrently at a simulated time granularity enforced by the simulator. Whenever a disk request completes, the next request waiting to be served in the current round is synchronously submitted, similar to the hardware disk case.

4.5 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 earlier. The server memory is allocated in groups of blocks contiguous in virtual memory. From experiments with raw interface disk accesses, we found that non-contiguity of the memory buffers could penalize disk bandwidth significantly in the AIX4.2 implementation. Although this might be attributed to the way that scatter/gather features of the disk controller are used by the system, we found the allocation contiguity easy to enforce.

For the allocation of buffer blocks we use a bitmap structure with an interface that can support block group requests. Deallocations are allowed on a block basis, as opposed to entire block groups obtained during allocation. This last feature increases independence between disk accesses and network transfers.

In our design, we do not cache previously accessed data, as is done in traditional file and database systems. Although related research has developed data caching algorithms for constant rate streams, 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).

Paging of buffer space is prevented by locking the corresponding pages in main memory. Although several Unix versions (e.g. HP-UX, Irix, Solaris) and Linux make available for this purpose the *mlock* system call, AIX does not. Instead, we exported the *pinu* kernel service through a loadable kernel module, and used that.

4.6 Schedule Management

The configuration input provides the basic parameters of each device that will be used. This includes the disk and metadata file name, the maximum and track seek time, the average rotation latency, and the minimum internal transfer rate. Other configuration parameters include the available network bandwidth along with the server and client buffer space, as required by the different policies.

The schedule management module is responsible for generating data transfer schedules, where each schedule specifies the amount of data accessed from the disks, stored in server buffer and transmitted over the network during each round. The schedule manager accepts as input actual MPEG-2 Unix files (or their frame size traces), along with the prefetching and striping scheme that should be used. The prefetching schemes can make use of the buffer space that is available at the server or client side, depending on the policy. The striping schemes specify the disks and the amount of data that should be accessed during each round.

4.7 Summary

In this chapter we described basic alternatives and decisions in our prototype implementation. We introduced system design details regarding the admission control and dispatching of playback requests, and the structure of the disk metadata. We also explained our approach for scheduling the disk transfers, organizing the buffers and allocating the disk storage space.

Chapter 5

Experimentation Environment and Basic Parameters

In the present chapter, we introduce the hardware configuration that we used in our performance measurements, and the set of streams that composed our benchmark. We define a novel method for media server performance evaluation that can be easily applied to different system scales and stream characteristics. In addition, we examine potential performance implications of the round duration and the contiguity of the buffer or disk storage space. Finally, we evaluate the effectiveness of our resource reservation scheme.

5.1 Experimentation Setup

Several of our experiments have been made using the system in Admission Control mode. We also used the Full Operation mode for low-level performance measurements and for examining the efficiency of our resource reservation scheme. The Simulated Disk mode allowed additional configuration flexibility for validating results on homogeneous and heterogeneous disk arrays.

Our performance measurements in Full Operation mode were made on an IBM RS/6000 two-way SMP workstation with 233 MHz PowerPC processors running AIX4.2. The system was configured with 256 MB physical memory, and a fast wide SCSI controller to which a single 2GB disk was attached, used for the system and paging partitions. The stream data are stored

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 Zone to Outer Zone Burst	122 to 177 Mbit/s
Buffer Size	0.5 MB	Inner Zone to Outer Zone Sustained	11.3 to 16.8 MB/s

Table 5.1: Features of the SCSI disks used in our experiments.

on two 4.5GB Seagate Cheetah ST-34501W disks (Table 5.1) attached to a separate ultra wide SCSI controller. 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). Although storage capacity can take much larger values in the latest models, the remaining performance numbers of the above two disks are typical of today's high-end drives.

Experiments in Admission Control and Simulated Disk modes also assume arrays of disks with the above specification (with the exception of the heterogeneous disk case described in more detail later). The logical block size B_l is set equal to 16 KB, while the physical sector size B_p is 512 bytes. Unless otherwise stated, the stride size B_s in the disk space allocation is 2 MB. Depending on the case, a total space between 32 MB and 256 MB for every extra disk is used for memory buffer, and is organized in fixed size blocks of 16 KB.

In our experiments, data retrieved from the disks are discarded (copied to the *null* device), leaving protocol processing and contention for the network outside the scope of the present study. However, from our experience with UDP/IP transfers on alternative configurations with sufficient network bandwidth, we do not expect the network overhead to affect in any fundamental way the results presented here. The round time is set equal to one second.

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 720x480 and 24 bit color depth, 30 frames per second frequency, and a $IB^2PB^2PB^2PB^2PB^2$ 15 frame Group of Pictures structure. The encoding hardware that we use allows the generated bit rate to take values between 1Mbit/s

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

Table 5.2: 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.

and 9.6Mbit/s. Although the Main Profile Main Level MPEG-2 specification allows bit rates up to 15Mbit/sec, there is a typical point of diminishing returns (no visual difference between original and compressed video) at 9Mbit/sec. The digital versatile disk (DVD) specification sets a maximum allowed MPEG-2 bit rate of 9.8Mbit/sec.

Statistical characteristics of the clips are given in Table 5.2, where the coefficients of variation of bytes per round lie between 0.028 and 0.383, depending on the content type. We used the MPEG-2 decoder from the MPEG Software Simulation Group for stream frame size identification (MPEG Software Simulation Group 1996). In our *mixed* basic benchmark, the six different streams are submitted round-robin. Where necessary, experimental results from individual stream types are also shown.

5.2 Performance Evaluation Method

Although media server architectures have been investigated for more than a decade, performance parameters are usually evaluated in ad-hoc ways, and not according to generally accepted methodologies. Important decisions about the interarrival process used, the stream scheduling techniques employed, and several user-oriented constraints that keep the system operation practical usually vary inconsistently across different related studies.

In general, we expect that a fair performance evaluation method:

- demonstrates the capacity of the system for the performance parameters used,
- is applicable to different hardware configurations,
- is not biased towards any particular policies, and
- evaluates the expected practical operation of the system.

We assume that playback initiation requests arrive independently of one another, according to a Poisson process. The system load can be controlled through the corresponding arrival rate λ. If disk bandwidth is the bottleneck resource, we consider the perfectly efficient 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:

$$\mu = \frac{\sum_{k=0}^{D-1} R^k_{disk} \cdot T_{round}}{S_{tot}}. \tag{5.1}$$

Then the maximum arrival rate $\lambda = \lambda_{max}$ that can be handled by the system is equal to the above service rate. This should create enough load to show the performance benefit of arbitrarily efficient data striping policies. The system load ρ is equal to:

$$\rho = \frac{\lambda}{\mu} \leq 1, \tag{5.2}$$

where $\lambda \leq \lambda_{max} = \mu$.

When a playback request arrives, it is checked whether adequate resources are available for every round during playback. The test should consider the exact data transfers of the requested playback for every round and also the corresponding available disk transfer time, network transfer time and buffer space in the system. If the request cannot be initiated in the next round, the test is repeated for successive rounds until the first future round is found, where the requested playback can be started with guaranteed sufficiency of resources. We also tried alternative scheduling techniques previously proposed, where request initiations are attempted at fixed or random offsets from the current round (Reddy and Wijayaratne 1999). We did not observe significant performance improvements from this approach. This is due to the Poisson arrival process that we assume (as opposed to all requests arriving at the beginning (Reddy

and Wijayaratne 1999), which naturally distributes transfers from different clients across the disks of the array to different rounds.

We define as *lookahead distance* H_l the number of future rounds that are considered as candidate rounds for initiating a stream playback. The number of rounds between the arrival of a request and the beginning of the playback is equal to the playback initiation latency experienced by the user. 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. This latency cannot be unlimited in order for the service to be acceptable by the users. On the other hand, setting the lookahead distance too small can prevent the system from attaining full capacity. This system design tradefoff between throughput and latency is investigated in more detail in the next chapter.

We set the *basic lookahead distance* H_l^{basic} to be equal to the mean number of rounds between request arrivals $H_l^{basic} = \lceil \frac{1}{\lambda} \rceil$. Intuitively, setting $H_l = H_l^{basic}$ allows the system to consider for admission control the number of upcoming rounds that will take (on average) for another request to arrive. More generally, we define as *lookahead factor* F_l the fraction $F_l = \frac{H_l}{H_l^{basic}}$. All the experiments presented in the thesis 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.

The playback initiation latency that can be tolerated by users depends on the video streaming application. For example, as the playback duration becomes longer, the maximum acceptable latency may be larger. The maximum number of rounds considered for initiating a stream is determined by the throughput of the system and the system load. The corresponding latency can be made equal to an arbitrarily small integral number of rounds by using a configuration with large enough system throughput, or operating the system at sufficiently low load. This follows from the expression:

$$H_l = F_l \times H_l^{basic} = 1 \times \lceil \frac{1}{\lambda} \rceil = \lceil \frac{1}{\rho \cdot \lambda_{max}} \rceil \tag{5.3}$$

Here we assume lookahead factor $F_l = 1$, which is justified in the next chapter. In most practical

situations, the rejection ratio should also be kept low, for example close to 1%. This can be also achieved by operating the system at loads lower than the maximum 100%, as is shown in the next chapter for different striping policies.

5.3 Summary of Differences from Previous Studies

Before studying the results from our experiments, it is worthwhile to summarize important differences in our assumptions from those of previous related studies.

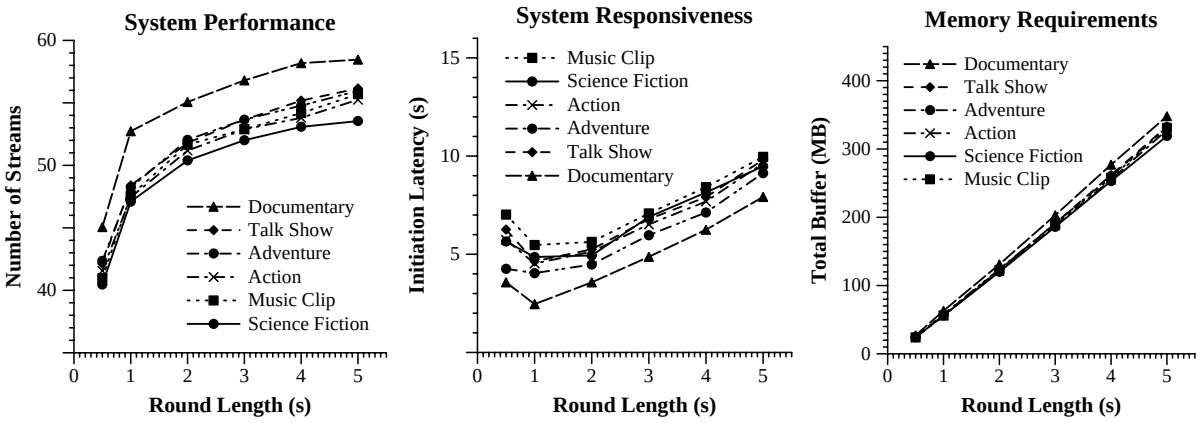
1. We assume that playback requests arrive according to a Poisson process, which closely resembles system operation in practice. Instead, several previous studies made the worst case assumption that all the arrivals occur at the beginning of the experiments (Salehi et al. 1996; Shenoy and Vin 1997; Reddy and Wijayaratne 1999).

2. We adjust the system load according to the available resources in the system, which makes comparison of different system configurations fair. Other studies assumed an infinite number of requests (Shenoy and Vin 1997; Reddy and Wijayaratne 1999), or requests arriving according to an arbitrarily specified rate (Makaroff et al. 1997; Douceur and Bolosky 1999).

3. For the admission control process, 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 limiting the playback initiation latency that a user has to tolerate, 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, instead of considering only one round.

5. In subsequent chapters, we also keep track of the rejection ratio of playback requests. We make sure that it remains limited (around 20%), with the most efficient policies at highest loads that we examine.



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

Figure 5.1: Interesting effects of round length on 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.

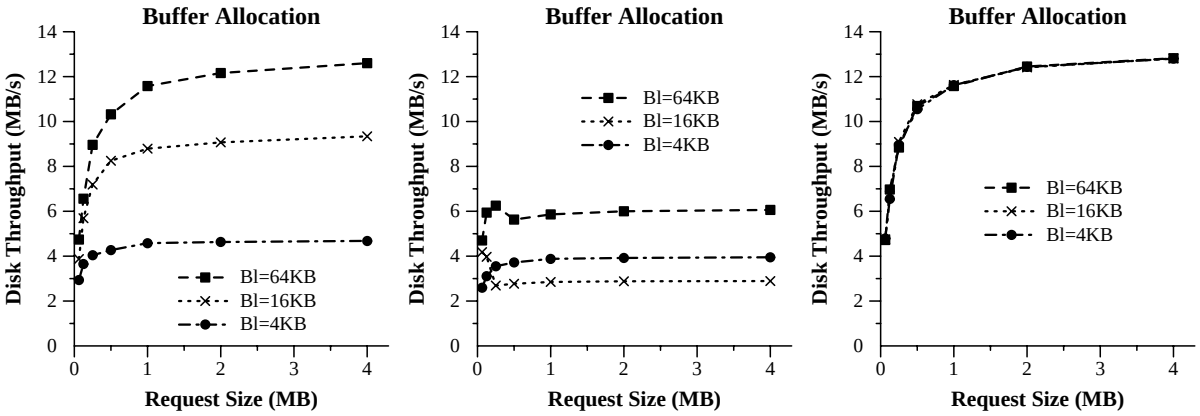
5.4 Study of Basic Parameters

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

5.4.1 Choosing the Right Round Length

We start with an investigation of how the round length affects the performance and the resource requirements of the system. We use the Admission Control mode for our experiments, and assume that the disk bandwidth forms the system bottleneck. A four-disk array is assumed, with load $\rho = 80\%$, lookahead factor $F_l = 1$ and Variable-Grain Striping. Experiments with other parameter values led to similar conclusions.

Longer rounds increase the data size of the disk transfers and can improve the disk operation efficiency. There is a diminishing returns effect though, as shown in Figure 5.1(a). The performance benefit from longer rounds depends on the ratio between useful transfer time and mechanical overhead within each disk access. A round length of one second achieves most of



(a) multiple read's (b) multiple readv's (c) single read

Figure 5.2: 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()` cuts 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.

the system capacity, with current disk technology and stream data requirements.

In Figure 5.1(b), we measure the average initiation latency of accepted streams, and notice that it increases almost linearly with rounds longer than one second. In fact, latency depends on both the number and the length of the rounds tried during stream admission control. The plots of Figure 5.1(b) indicate that system responsiveness can be controlled by limiting the round length appropriately. From Figure 5.1(c), it also 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 conclusion, the choice of the round length is crucial for several aspects of the system operation. A value of one second achieves high throughput, with reasonable initiation latency and buffer requirements. This is also a typical value used in other studies, which facilitates the comparison of our results with those of previous related research.

5.4.2 Contiguity of Buffer Allocation

In this section, we examine how disk performance depends on the buffer allocation policy, using a synthetic benchmark that we developed for that purpose. We measure the disk throughput at different sizes of I/O requests and degrees of contiguity in the buffer space allocated for each request. Disk requests of a specific size are initiated at different locations approximately uniformly distributed across the disk space. Disk data are transferred to pinned memory, organized in blocks of fixed size B_l, similar to our prototype.

In Figure 5.2(a), we depict the average throughput, when a separate `read()` call is invoked for each buffer block corresponding to a request. We vary both the block size and the size of the request. When the block size increases from 4KB to 64KB, disk throughput changes by a factor of three across different request sizes. When the request size varies between 64 KB and 4 MB for a particular block size, the throughput increases by more than a factor of two.

In Figure 5.2(b) we repeat the previous measurements by using the `readv()` system call instead. It takes as parameters the pointer to an array of address-length buffer descriptors, along with the size of the array. In most systems, the array size is limited to a small number of buffer descriptors (e.g. $IOV\_MAX = 16$ in AIX, Solaris). For each I/O request, the required number of `readv()` calls is used, with the array entries initialized to the address and length of each buffer block. Although we expected improved performance due to the increased amount of information supplied with each `readv()` call to the host system, the measured throughput was less than half of what we measured with `read()`. Proper explanation for this would probably require internal knowledge of the AIX device drivers that we don't have. An additional limit is also imposed on the I/O performance due to the small value of the IOV_MAX.

In Figure 5.2(c), we repeated the previous experiments by using only a single `read()` call for each request, similarly to the way I/O requests are served in our prototype. This policy requires contiguity at the virtual address space, without any control on the physical addresses of the underlying pages. Still, it is remarkable how the sensitivity to the block size B_l disappears. Note that the achieved performance is only slightly higher than that of figure 5.2(a) with large blocks. However, the block size itself cannot be arbitrarily large; otherwise the benefit from multiplexing requests of different sizes drops, which eventually reduces the number of accepted

streams (see also Chapter 6). Since the average size of the disk transfers is about 625,000 bytes in our MPEG-2 clips, from these experiments we can expect the disks to operate at average throughput higher than 11 MB/s, which is consistent with the minimum sustained rate 11.3 MB/s advertised in the disk specification.

We conclude (for this system) that contiguity in the buffer space allows a relatively small block size to be chosen that guarantees both high number of streams and efficient disk transfers. This simplifies the performance tuning of the system. One disadvantage is the complexity introduced by having to manage buffer ranges instead of fixed buffers. In addition, buffer space external fragmentation requires a number of buffers to remain unused (no more than 10-15% of the total buffer space, in our experience).

5.4.3 Contiguity of Disk Space Allocation

Arguably, disk access efficiency would improve if the disk space corresponding to each request were allocated contiguously, requiring a single disk head movement instead of a maximum of two incurred by stride-based allocation. We investigate this issue by measuring the disk bandwidth utilization when retrieving streams allocated on a disk using different stride sizes, while still keeping the stride size larger than the stream requests in a round (as per our original constraint). As was explained before, the stream strides are approximately uniformly distributed across the cylinders of the disks in order to prevent disk geometry biases.

Figure 5.3 shows the measured bandwidth utilization on a single disk configuration when retrieving different streams. The achieved stream throughput is based on the resource reservations of Section 3.4, and remains the same across different stride sizes. The system load was set equal to $\rho = 80\%$, and the statistics were gathered over a period of 2,000 rounds after an initial warmup of 500 rounds. One important observation from these plots is that disk utilization drops as the stride size is increased from 2 MB to 16 MB. This is not surprising, since a larger stride size reduces disk head movements, and improves disk efficiency overall.

However, Figure 5.3 shows that the total improvement in disk utilization does not exceed 2-3%. This percentage does not justify using larger strides, and increasing the unused storage space at the last stride of each stream. More importantly, it indicates that stream disk accesses are dominated by useful data transfers rather than mechanical overhead. Generally,

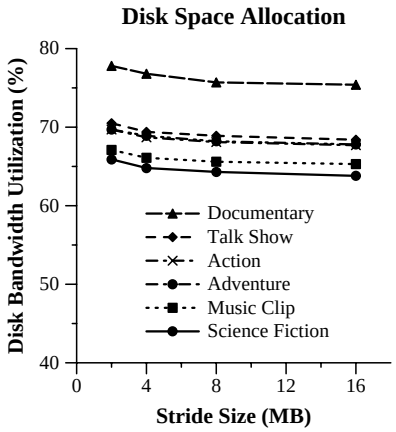


Figure 5.3: Increasing the stride size from 2 MB to 16 MB reduces only marginally (2-3%) the disk bandwidth utilization across different stream types. Therefore, the expected benefit from either large strides, or contiguous disk space allocation, would be limited. A single-disk configuration was used with load $\rho = 80\%$, lookahead factor $F_l = 1$ and a degenerate version of Variable Grain Striping for one disk.

in an environment of multiple streams striped across several disks, the expected benefit from contiguous disk space allocation would be limited. Reduction in disk actuator overhead as a result of technology advances will only make this argument stronger.

5.4.4 Resource Reservation Efficiency

After the previous study of buffer and disk space allocation parameters, we fix the buffer block size to $B_l = 16KB$ and the stride size to $B_s = 2MB$. In a system with 2 disks and 64 MB buffer memory, we compare the reserved and measured resource utilizations across different stream types. We set the system load to 80%, use the Variable Grain Striping, and gather statistics for a period of 2,000 rounds after a warmup of 500 rounds. Further increasing the load would not increase the system utilization, unless stream smoothing were used as we show later. The average number of active streams in the above measurement period was roughly between 20 and 25 depending on the stream type.

The measured busy time in most rounds was less than or within a few milliseconds of the total disk time reserved. In only a small percentage of rounds (less than 1%) the discrepancy would be higher, and this is hard to avoid completely, due to mechanical and other kinds of unexpected overhead. However, all the discrepancies were limited enough to be hidden by

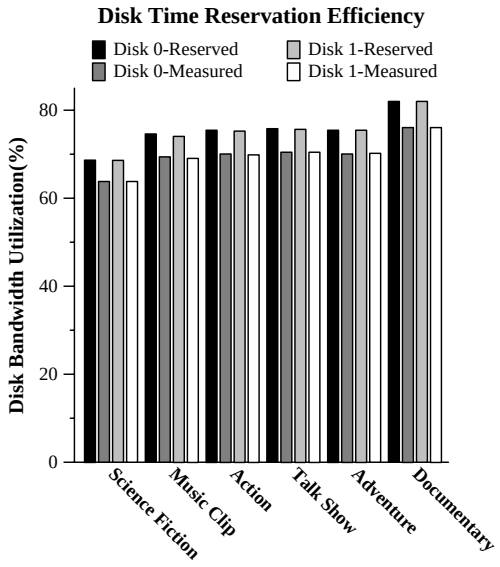


Figure 5.4: In a two-disk configuration with load $\rho = 80\%$, the measured disk utilization is balanced between the two disks. On each disk, the reserved disk utilization bounds relatively tightly (is only higher by about 5%) the measured disk utilization.

the client with an extra round added to the playback initiation latency. Other than that, we got stable prolonged system operation at high loads without any observed problems. The corresponding processor utilization hardly exceeded 5% on our SMP machine. (We expect the processor utilization to be higher when network protocol processing is included.)

In Figure 5.4, we illustrate the fraction of the measurement period, during which each of the two disks was busy, and the corresponding fraction of reserved time. We notice that the load is equally balanced across the two disks. (This observation remained valid when striping streams across larger disk arrays as well, which has important scalability implications.) In addition, the reserved busy fraction does not exceed by more than 5% the corresponding measured. This allows the admission control procedure to successfully offer quality of service guarantees, without disk bandwidth underutilization.

The reservation of disk access time assumes that two disk head movements are required for serving each stream request to a disk. In fact, a more accurate approximation of the required number of seeks and rotations would take into account the ratio of the average stream request size over the stride size. However, we prefer using the more pessimistic two head movements,

because it allows tolerance of unpredictable mechanical delays. This makes the system operation more stable, without significant waste of disk bandwidth.

Each of the buffer blocks allocated for a data transfer is marked busy at the beginning of the round when the disk access occurs. It is not released until its last byte is sent over the network, in some subsequent round. On the other hand, resource reservation during admission control reserves the length of each buffer block for the entire duration of the rounds that it spans. In the particular experimental setup used here, no network transfers occur, essentially simulating an infinite bandwidth network. Fast network transfers allow early release of buffer blocks at the beginning of the round of the last-byte network transfer. Slower network transfers can delay block releases until the end of the round of the last-byte network transfer. In general, depending on the speed of the network subsystem and the network scheduling policy, we expect the measured buffer utilization to lie somewhere between the half and the total reserved buffer space.

5.5 Summary

In this chapter, we described the hardware features of our experimentation platform and the set of streams that we used in our measurements. In addition, we described our method for evaluating the performance of media servers. We proceeded with a study of basic system parameters that control the round length, the contiguity of the buffer space, the contiguity of the disk storage space, and resource reservation efficiency.

Chapter 6

Comparative Study of Disk Striping

In the present chapter, we examine the effects of the system load and the lookahead distance parameters on the system performance, before deciding on particular values of these parameters that we should use in the rest of our study. 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 the Fixed-Grain Striping. An important feature of this method is the ability to control the disk access efficiency through the choice of block size, B_f. As the block size is increased, a larger part of each access is devoted to data transfer rather than mechanical movement overhead. When a stream requests more than one block from a particular disk during a round, a maximum of two contiguous accesses are sufficient with the stride-based disk space allocation we used.

As shown in Figure 6.1, the number of active streams with sixteen disks and the mixed workload increases linearly as the load, ρ, increases from 10% to 50%. At loads higher than 50%, the number of streams that can be supported no longer increases. Not surprisingly, the

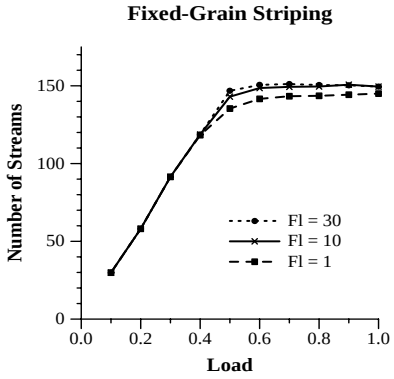


Figure 6.1: The sustained number of active streams with Fixed-Grain Striping flattens out at loads higher than 50% using a block size $B_f = 327,680$ with sixteen disks and the mixed workload. Changing the lookahead factor F_l from 1 to 30 increases the number of streams by less than 5%.

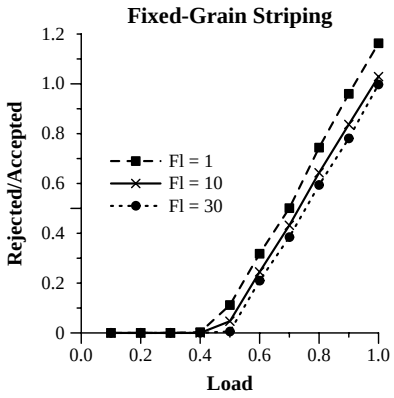


Figure 6.2: For Fixed-Grain Striping with $B_f = 327,680$, sixteen disks and the mixed workload, the total number of rejected streams over the total number of accepted during the measuring period. The ratio increases linearly as the load exceeds 50%, and changes by less than 20% as the lookahead factor F_l is increased from 1 to 30.

additional load beyond 50% translates into a corresponding increase in the number of rejected streams (Fig. 6.2). Since increasing the lookahead factor from 1 to 30 improves the number of streams that can be supported only marginally, for the rest of our experiments we set the lookahead factor F_l to 1.

This corresponds to a lookahead distance of less than 10 rounds, for a system of sixteen disks operating at load $\rho = 80\%$, and half-hour clips of about 1GByte each. The total disk bandwidth available in the system is equal to $11,300,000 \times 16$ bytes/s. If we divide that by the

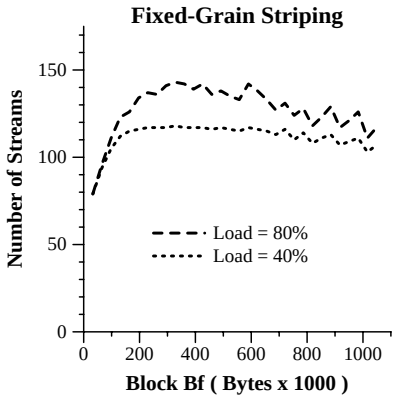


Figure 6.3: The number of active streams with Fixed-Grain Striping at different values of B_f. At both load values 40% and 80%, a maximum number of streams is achieved at $B_f = 327,680$. The experiments have been done over a range of B_f between $32,768$ and $1,048,576$ at steps of $32,768$. Sixteen disks have been used with the mixed workload.

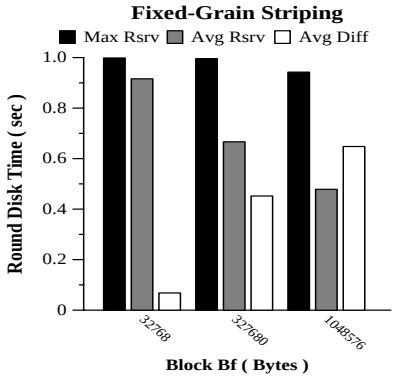


Figure 6.4: The maximum (Max Rsrv) and average (Avg Rsrv) disk access times reserved for a specific disk (Disk 0) in each round during the measuring period. Also, the maximum difference (Avg Diff) is shown between the reserved access times across all the disks in each round, averaged over the measuring period. Three different block sizes are tried with sixteen disks and the mixed workload at load $\rho = 80\%$.

average stream size $1.1 \cdot 10^9$ bytes, we get the maximum arrival rate that can be handled by the system $\lambda_{max} = 0.164$ streams/s. The corresponding lambda at load $\rho = 80\%$ becomes equal to $\lambda = \rho \times \lambda_{max} = 0.13$ streams/s. Therefore, with $F_l = 1$, the lookahead distance is equal to $H_l = F_l \times H_l^{basic} = \lceil \frac{1}{\lambda} \rceil = 8$ rounds.

For load values $\rho = 40\%$ and $\rho = 80\%$, we measure the number of active streams as the block size increases from $B_f = 32KB$ to $B_f = 1,024KB$ bytes at steps of 32 KB. As can be seen from

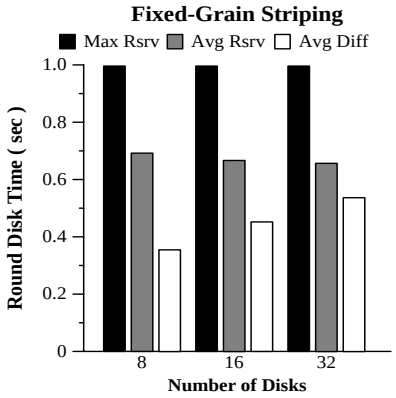


Figure 6.5: As the number of disks takes values 8, 16 and 32, the maximum and average reserved access time for disk 0 (these values are respectively similar for the rest of the disks) remain almost the same. This is despite the corresponding increase in the maximum difference between reserved access times across the disks. The Fixed-Grain Striping policy is used on sixteen disks with mixed workload at load 80%.

Fig. 6.3, at load 80% the number of streams increases until B_f becomes approximately equal to $320KB$. With larger block sizes it drops. A similar behavior is noticed at 40%, although the variation in the number of streams is much smaller across different block sizes.

The Admission Control mode that is used for the above experiments allows also gathering of statistics on system resources reserved during the admission control process. In particular, Fig. 6.4 depicts the maximum and average access time $T_{disk}(i,k)$ reserved during the measuring period $3,000 \leq i < 9,000$ for a particular disk ($k = 0$). A sixteen disk configuration is used with load $\rho = 80\%$. While the maximum time remains close to 100% across different block sizes, the average time drops from about 90% at $B_f = 32KB$ to less than 50% at $B_f = 1,024KB$.

With the round time set to 1 sec, the average time (normalized by the round time) corresponds to the expected disk utilization and varies depending on the number of disks accessed for a stream every round. Part of it is actuator overhead and decreases as the block size becomes larger. On the other hand, the maximum difference in reserved access times among different disks in a round (Avg Diff in Fig. 6.4) increases on average from almost zero to above 60% with increasing block size B_f.

Conclusions similar to the above affect the choice of the block size that maximizes the number of streams, and confirm results from previous studies (Shenoy and Vin 1997). However,

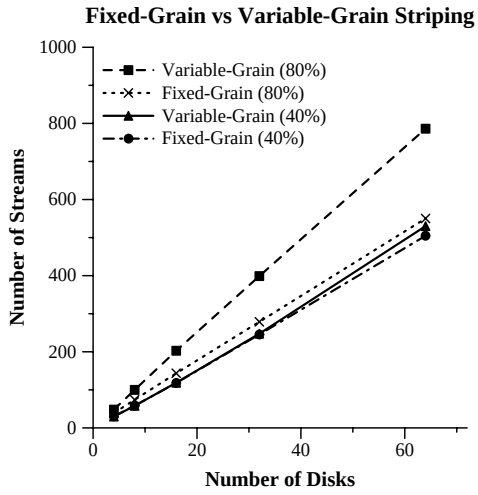


Figure 6.6: The sustained number of streams is measured using the Fixed-Grain Striping policy with a block size B_f that maximizes the number of streams. At load $\rho = 80\%$, the streams increase almost linearly from 39 to 550 as the number of disks increases from 4 to 64. For $\rho = 40\%$, there is a corresponding increase from 30 to 505. With Variable-Grain Striping instead, the number of streams increases from 48 to 786 at $\rho = 80\%$, and from 30 to 530 at $\rho = 40\%$.

we also found that the average reserved time (shown in Fig. 6.4 only for disk 0) remains about the same (within 2%) across a disk array. Thus, the access load, on average, is equally distributed across the disks, despite variations from round to round. Furthermore, as the number of disks increases, the average time drops only slightly from 69% with 8 disks to 67% with 16 and 66% with 32 disks (Fig. 6.5). This implies that the useful capacity of the system increases almost linearly as more disks are added.

We repeated the above measurements varying the number of disks from 4 to 64 (Fig. 6.6). The block size B_f, that maximized the number of streams, was found to remain at $B_f = 320KB$. At 80% load, the number of streams that could be supported increased 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. With the load at 40%, the number of streams increased from 30 with 4 disks to 505 with 64 disk, thus reflecting the improved capacity of the system with increased number of disks at low loads. At load $\rho = 40\%$, we see a slightly superlinear increase of about 4%, which is less than the relative statistical error $\gamma = 5\%$ that we allowed in our experiments. One possible explanation for this behavior is the statistical error, which was higher at low loads and small

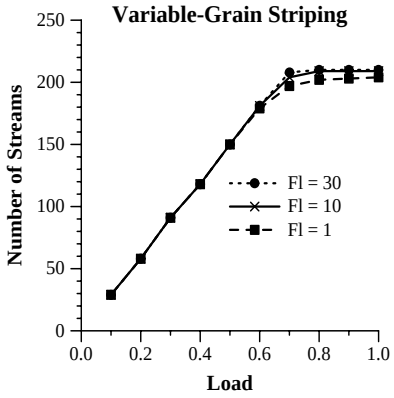


Figure 6.7: The sustained number of active streams with Variable-Grain Striping flattens out at loads higher than 70%. Changing the lookahead factor F_l from 1 to 30 increases the number of streams less than 5%. Sixteen disks are used with the mixed workload.

number of disks. Another possible reason is the increased statistical benefit of multiplexing requests from a larger number of streams across a larger number of disks.

6.2 Comparison with Variable-Grain Striping

In the previous section, we studied the behavior of Fixed-Grain Striping. We found that with the mixed workload the number of streams is maximized at $B_f = 320KB$ across different number of disks and system load values. In the present section, we study Variable-Grain Striping with respect to scalability. This is done for first time to the best of our knowledge (Anastasiadis et al. 2001a).

In Fig. 6.7, we show the performance of Variable-Grain Striping on sixteen disks as the load increases from 10% to 100%. The number of streams grows linearly as the load increases up to 70%. This is significantly higher than the 50% load, where Fixed-Grain Striping flattened out (Fig. 6.1). Loads higher than 70% with Variable-Grain Striping only increase the number of rejected streams as shown in Fig. 6.8. As before, a lookahead factor value of $F_l = 1$ attains more than 95% of the system throughput, and that is the value that we will use.

In Figure 6.9, we depict the reserved disk access time per round. As the number of disks increases, the average reserved time increases from 83% with 8 disks, to 84% with 16 disks, and 85% with 32 disks. We also measured the maximum number of sustained streams from 4 to 64

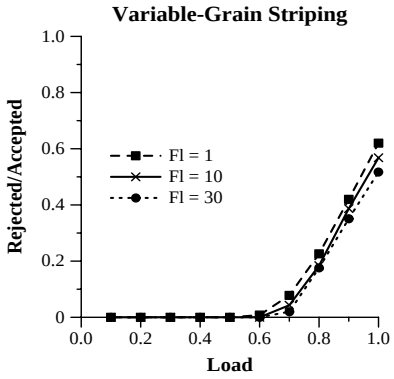


Figure 6.8: The total rejected streams over the total accepted during the measuring period. The ratio increases linearly as the load exceeds the 70% value, and changes by no more than 10%, as the lookahead factor F_l varies between 1 and 30. Sixteen disks are used with the mixed workload.

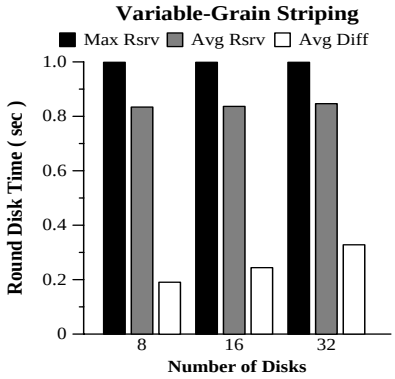


Figure 6.9: As the number of disks takes values 8, 16 and 32, the maximum and average reserved access time in Disk 0 (the values are respectively similar for the rest of the disks) remain almost the same. This is despite the corresponding increase in the maximum difference between reserved access times across the disks. The Variable-Grain Striping policy is used on sixteen disks with the mixed workload at load 80%.

disks (Fig. 6.6). At a load of 80%, the number of streams increases from 48 with 4 disks, to 203 with 16 disks and 786 with 64 disks. Thus, as the number of disks increases, the number of streams remains within 3% of what perfectly linear scalability would achieve. In addition, the advantage of Variable-Grain Striping over Fixed-Grain Striping increases from 23% with 4 disks to 43% with 64 disks. To the best of our knowledge, this is the first scalability analysis of the Variable-Grain Striping policy.

A straightforward way of calculating the maximum theoretical capacity of the system is to divide the total disk bandwidth of the server by the average transfer bandwidth required by

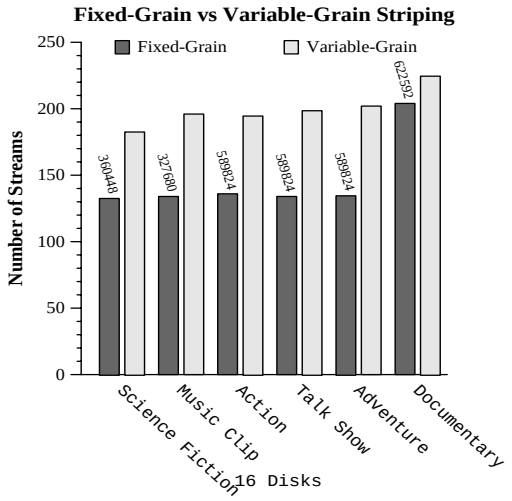


Figure 6.10: The advantage of Variable-Grain Striping over Fixed-Grain Striping varies among 38% in Science Fiction, 46% in Music Clip, 43% in Action, 49% in the Talk Show, 50% in the Adventure, and 11% in the Documentary. The block size shown for Fixed-Grain Striping was found to maximize the number of streams, over a range of block sizes between 32 KB and 1,024 KB with step of 32 KB. The load was always set equal to 80%.

a stream. It should be made clear that this is not a very tight upper bound since it ignores both the disk access overhead and the transfer size variability of variable bit-rate streams. The actual disk bandwidth utilization that more reallistically represents the efficiency of the system is measured using simulated disks in the next section. Possible approaches for improving it are investigated in the next chapter. Disk bandwidth utilization measurements with hardware disks have been presented previously (Chapter 5).

From table 5.1 the transfer bandwidth of an individual disk can be taken equal to 11.3MB/s, while from Table 5.2 the average transfer bandwidth of a stream is equal to 0.624MB/s. Therefore the maximum capacity of the server cannot exceed 18 concurrent streams with one disk, 72 streams with 4 disks and 1158 streams with 64 disks. By dividing the measured throughout of the system with the maximum capacity that we calculated, we find that Fixed-Grain Striping does not achieve more than 44% of the maximum capacity with 64 disks, while Variable-Grain Striping gets close to 68%. They are both less than 100% due to transfer size variabilities in the streams and the simplifications in the calculation of the upper bound.

In Fig. 6.10, we consider individual stream types in more detail. As the content type changes

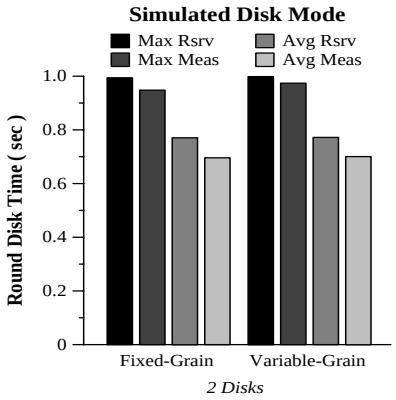


Figure 6.11: The reserved disk time statistics (Disk 0) are no more than 8% higher than the measured access time statistics using the detailed Seagate Cheetah ST34501N model of Ganger et al ($\rho = 80\%$).

from Science Fiction to Documentary and the variation in data transfers correspondingly drops, the block size needs to be larger in order to maximize the performance of Fixed-Grain Striping. However, the performance remains about the same for the five stream types, and increases only with the Documentary stream. In contrast, Variable-Grain Striping manages to transform even minor decreases in data transfer variation into improved performance. Overall, Variable-Grain Striping maintains an advantage over Fixed-Grain Striping between 11% and 50%.

The explanation for the performance advantage of Variable-Grain Striping over Fixed-Grain Striping is two-fold. 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. Although improved disk access efficiency can also be achieved by using Fixed-Grain Striping with large blocks, it will not be as efficient as Variable-Grain Striping because the multiplexing benefit is lacking.

6.3 Validation in Simulated Disk Mode

In order to keep the computation time reasonable, the previous experiments were conducted with our system in Admission Control mode, where playback requests arrive leading to corresponding resource reservations, but without actual time measurement of the individual disk

transfers. In the present section, we compare the statistics of the disk time resource reservations with the statistics gathered over the access times of all individual data transfers involved, using the DiskSim representation of the Seagate Cheetah disk (Ganger et al. 1999). A two-disk array model is used with each disk attached to a separate 20MB/sec SCSI bus, and no contention assumed on the host system bus connecting the two SCSI buses. The statistics are gathered during 6,000 rounds after a warmup period of 3,000 rounds, as before. The mixed workload is used with average number of active streams 21 and 23 for Fixed-Grain and Variable-Grain Striping, respectively, corresponding to 80% load.

As can be seen from Fig. 6.11, in both the average and maximum case, the reserved disk time is no more than 8% higher than the corresponding measurements using the detailed disk model of Ganger et al. The difference can be attributed to the fact that the reservation assumes a minimum disk transfer rate and ignores on-disk caching. Practically, the achieved accuracy in the predicted disk access times during resource reservation is adequate. In fact, any extra disk geometry information that would be required to improve it is not readily available in commercial disk drives (Worthington et al. 1995).

6.4 Effect of Technology Trends

To project disk technology improvements for the foreseeable future, we extend compound growth rates from the past linearly into the future (Table 6.1). In particular, we assume 30% increase in internal disk transfer rate per year, and 23% decrease in seek distance (Ng 1998). The full seek time depends linearly on seek distance, so the decrease is also 23%. However, we assumed decrease of 12% per year for the track seek time, which is dependent on the square root of the seek distance (among other factors more complex to project including settle time). Finally, we assume a rotation speed increase of 12% per year (Ruemmler and Wilkes 1994). We presume that stream types and sizes will remain the same. This is a conservative projection, ignoring potential demand for higher resolution and more content rich streams.

In our experiments so far, we compared Fixed-Grain Striping to Variable-Grain Striping, which is a special case of Group-Grain Striping at $G = 1$. With current disk technology,

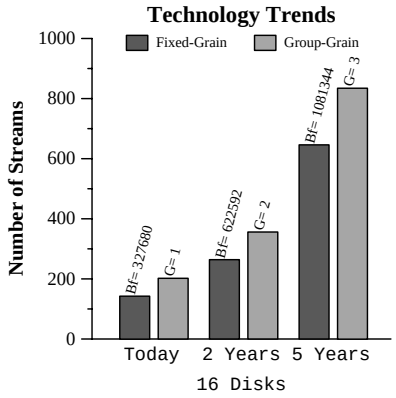


Figure 6.12: With reasonable technology projections, two years into the future Group-Grain Striping (generalized Variable-Grain Striping) maintains an advantage of 35% over Fixed-Grain Striping (from 41% today). The corresponding benefit in five years is no less than 29%. The shown values of B_f and G were found to maximize the throughput of the two policies respectively.

Disk Parameter	Today	2 Years	5 Years
Min Transfer Rate (MB/sec)	11.3	19.10	41.92
Max Seek Time (msec)	18.2	10.74	4.91
Track Seek Time (msec)	0.98	0.76	0.51
Avg Rotation Latency (msec)	2.99	2.38	1.70

Table 6.1: Projection of disk parameter changes in two and five years into the future.

having $G = 1$ maximizes the number of streams. 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 head overhead during each round, without negatively affecting the initiation latency or the buffer requirements of the streams, as increase of the round length would do (see Section 5.4.1).

Specifically, when using the mixed workload, we found that two years into the future, the number of streams that could be 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 at 29%. Thus, under reasonable assumptions about technological improvements, there are significant performance improvements when using Group-Grain Striping instead of Fixed-Grain Striping.

6.5 Summary

In this chapter, we demonstrated that the supported number of streams scaled almost linearly as the number of disks increased with Fixed-Grain Striping. The disk striping scalability is farther improved with Variable-Grain Striping, which outperforms Fixed-Grain Striping by 41% for mixed stream workload and up to 50% for individual stream types on sixteen disks. With a reasonable technological projection based on previous trends, we also show a significant advantage of Group-Grain Striping (generalized Variable-Grain Striping) over Fixed-Grain Striping two and five years into the future.

Chapter 7

Performance Study of Server-Based Smoothing

In this chapter, we introduce a stream smoothing algorithm that for its operation relies only on the buffer space available at the server side. The algorithm accepts as input a specification of the data amount that should be sent to the client over time. In order to prevent excessive smoothing from exhausting the available buffer space, data prefetching is applied as long as the proportion of server buffer required by each stream does not exceed the corresponding decreased proportion of the required disk bandwidth. Thus, the smoothing process is adjusted according to the total resources available in the server configuration.

7.1 The Server Smoothing Algorithm

7.1.1 Outline

Previous studies have pointed out the potentially low disk utilization and system throughput achieved when retrieving variable bit-rate streams, and the need for appropriately prefetching data into the server buffers (Makaroff et al. 1997; Reddy and Wijayaratne 1999). A similar utilization problem was also studied in the context of network links carrying variable bit-rate streams, where it was proposed to smooth bit-rate peaks along a stream by prefetching data into client buffers (Feng and Rexford 1997; Salehi et al. 1996). It was shown that such an

approach can improve bandwidth utilization in both disks and network links, but is dependent on the memory configuration of individual clients.

The low utilization problem occurs because the aggregate bandwidth requirements of concurrently served streams can be significantly higher at particular time instances than on average, due to variability. However, the admission control process bases its decisions on peak aggregate demand when considering new stream requests. Of course, the problem would disappear if the resource requirements from different streams were constant over time. Smoothing techniques attempt to achieve a similar effect using data prefetching. Thus, the maximum resource requirements become equal to (or close to) the average, and the number of accepted streams is maximized.

Here, we consider smoothing out disk bandwidth peaks by prefetching stream data into server buffers. One crucial issue with disk data prefetching is how to maintain an appropriate balance between disk bandwidth and server buffer space usage. Too aggressive prefetching can limit the number of concurrent streams that can be supported because of excessive server buffer usage Makaroff et al. 1997. Existing client-based smoothing algorithms do not have this problem, due to their implicit assumption of a fixed available client buffer size. The client buffer space need not be multiplexed among different streams as is the case when buffering is done at the server. Server-side prefetching addresses disk bandwidth and not network utilization. However, our scheme accepts as input a specification of the quantity of data that should be sent to the client over time. Thus, its operation can be complemented with client-based smoothing techniques for cases where clients have sufficient buffer resources.

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

in order to prevent exhaustion of the server buffer. Both conditions are applied to individual streams, and we experimentally study their effect when serving multiple streams concurrently.

Knowing the data amount that needs to be retrieved from the disks during stream playback is important information that can be used during stream storage. Appropriate striping methods that take advantage of this information have been previously shown to achieve significantly increased system throughput with respect to striping methods of fixed-size block (Anastasiadis et al. 2001a). On the other hand, a retrieval sequence that is fixed a priori might ignore the exact resource tradeoffs that occur during system operation, when different stream playbacks are multiplexed. We evaluate later in detail the resource utilizations that are achieved with our approach of using the retrieval sequence to determine the striping method.

7.1.2 Limitations of Previous Approaches

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

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

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

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

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

7.1.3 Basic Definitions

We use a "smoothness" criterion that is based on *Majorization Theory* (Salehi et al. 1996; Marshall and Olkin 1979). For any $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, let the square bracket subscripts denote the elements of $\mathbf{x}$ in decreasing order $x_{[1]} \geq \cdots \geq x_{[n]}$. For $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x}$ is *majorized* by $\mathbf{y}$, $\mathbf{x} \prec \mathbf{y}$, if:

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

and

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

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

From section 3.4, the disk time reservation for a disk transfer of X bytes is equal to:

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

Before further explaining the algorithm, we introduce some definitions. B_{disk} is the total server buffer divided by the number of disks D. We assume that all buffers are equally accessible by all the disks attached to each server node.

Definition 1 We define the *Disk Time Proportion* of X bytes, as the fraction of the round time T_{round} that the disk time reservation $T_{disk}(X)$ occupies: $P_d(X) = T_{disk}(X)/T_{round}$. We define the *Buffer Space Proportion* of X bytes as the fraction of the B_{disk} buffer space that X bytes occupy in a round: $P_b(X) = X/B_{disk}$. The *Maximum Resource Proportion* in round i, is the maximum of the corresponding disk time and buffer space proportions in that round $max(P_d(S_d(i)), P_b(S_b(i)))$.

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

```
0.  proc serverSmoothing
1.  input : L_n, S_n[] ( =0 outside [1..L_d] ), B_l
2.  output : L_d, S_d[], L_b, S_b[]
3.  begin
4.      blockQuantize(L_n, S_n[], B_l) (* see Figure 7.2 *)
5.      for t_rnd : 0..L_n-1
6.          if ( P_buf(S_b(t_rnd)) < P_dsk(S_d(t_rnd)) )
7.              repeat
8.                  t_min := t_rnd
9.                  P_min := max( P_dsk(S_d(t_rnd)), P_buf(S_b(t_rnd)) )
10.                 t_prv := t_rnd - 1, prefFailed := false
11.                 while ( prefFailed = false AND t_prv >= 0 )
12.                     P_prf = max( P_dsk(S_d(t_prv) + B_l),
13.                                  P_buf(S_b(t_prv) + B_l) )
14.                     P_shf = max( P_dsk(S_d(t_prv)),
15.                                  P_buf(S_b(t_prv) + B_l) )
16.                     (*check for max proportion decrease*)
17.                     if (P_prf < P_min)
18.                         t_min = t_prv, P_min = P_prf
19.                     else if (P_min < P_shf)
20.                         prefFailed := true
21.                     end-if
22.                     t_prv := t_prv - 1
23.                 end-while
24.                 if (t_min < t_rnd) (* update vectors *)
25.                     S_d(t_min) := S_d(t_min) + B_l,
26.                     S_b(t_min) := S_b(t_min) + B_l
27.                     for t_prv := t_min + 1 .. t_rnd - 1
28.                         S_b(t_prv) := S_b(t_prv) + B_l
29.                     end-for
30.                     S_d(t_rnd) := S_d(t_rnd) - B_l
31.                 end-if
32.             until (t_min >= t_rnd) (*prefetch search failed*)
33.         end-if
34.     end-for
35. end
```

Figure 7.1: The Server Smoothing algorithm generates majorization minimal disk sequence with the disk bandwidth proportion bounding above the corresponding server buffer proportion.

```
0.    proc blockQuantize
1.    input : L_n , S_n[] ( =0 outside [1..L_n] ), B_l
2.    output : L_d , S_d[], L_b , S_b[]
3.    begin
4.        S_d[] := 0, S_b[] := 0
5.        L_d := L_n , L_b := L_n + 1
6.        totSn := 0
7.        for t_rnd : 0..L_n
8.            prvSn := totSn, totSn := totSn + S_n(t_rnd + 1)
9.            (* we use function ceil() for the ⌈ ⌉ operation *)
10.           S_d(t_rnd) := B_l · ( ceil(totSn/B_l) − ceil(prvSn/B_l) )
11.           S_b(t_rnd) := S_b(t_rnd − 1) + S_d(t_rnd) - S_n(t_rnd − 1)
12.       end-for
13.   end
```

Figure 7.2: The `blockQuantize` procedure quantizes the disk transfers with respect to the logical block size B_l.

The above definitions only affect the number of blocks accessed in each round, since stream block accesses are done sequentially during playback.

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

7.1.4 The Algorithm

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

The Server Smoothing algorithm of Figure 7.1 initially invokes the `blockQuantize` procedure (see Figure 7.2) in order to generate disk and buffer sequences with disk data transfers

quantized with respect to the logical block B_l. Network transfers are specified in byte granularity for increased flexibility (if necessary, they could be quantized too). Then, rounds of the generated sequences are visited in increasing order starting from round zero. For every logical block of the current round, previous rounds are searched linearly in decreasing order for possible block prefetching. The search completes successfully when logical block prefetching can reduce the maximum resource proportion of the current round to values *higher* than those of the prefetch and shift rounds. This implies that the disk sequence can be smoothed out without incurring buffer utilization peaks exceeding the disk utilization of the current round. Otherwise, the block prefetching operation will not have any positive smoothing effect and the search fails. Below, we show that the algorithm works correctly.

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

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

Definition 4 If $b_1 \geq \cdots \geq b_n$ are integers and $b_i > b_j$, then the transformation $b'_i = b_i - 1$, $b'_j = b_j + 1$, $b'_k = b_k$, for all $k \neq i, j$ is called a *transfer from i to j.* [1]

Lemma 2 *(Muirhead 1903) If $\alpha_1, \ldots, \alpha_n$, $b_1, \ldots, b_n$ are integers and $\alpha \prec b$, then α can be derived from b by successive applications of a finite number of transfers.*

[1] The term *transfer* that we borrow from the original definition (rfmu03), should not be confused with regular data transfers.

Proof: See Marshall and Olkin 1979, pg. 135.□

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

Lemma 3 *The Server Smoothing algorithm produces disk sequence that satisfies the properties of Lemma 1, and has no transfer that would violate them.*

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

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

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

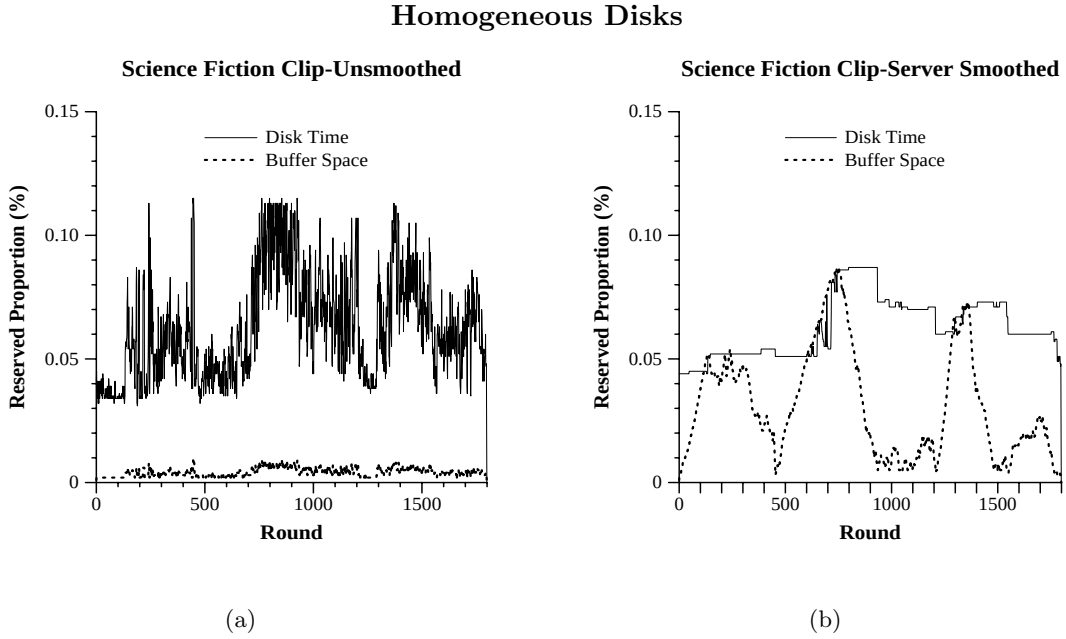


Figure 7.3: Example of applying the Server Smoothing algorithm. We depict the disk time and buffer space proportion in the system (a) before, and (b) after applying Server Smoothing. The maximum disk time proportion drops from 11.5% to 8.7%, while the maximum buffer space proportion increases from less than 1% to 8.7%. The Seagate ST34501 disk parameters are assumed and server buffer of 256MB per disk.

The computational complexity of the Server Smoothing algorithm is $O(\frac{\sum_{i=1}^{L_n} S_n(i)}{B_l} L_n^2)$, where $\mathbf{S_n}$ is the input network sequence and L_n is its length. Although it may be possible to reduce this complexity, practically the execution completes in tens of seconds in our experiments on 133MHz RISC processors with $B_l = 16,384$, $L_n = 1,800$ and $\sum_{i=1}^{L_n} S_n(i) = 1.12 \cdot 10^9$. Since the schedule generation is done off-line, the above execution time is acceptable. The higher computational complexity with respect to $O(L_n)$ of network smoothing algorithms (Salehi et al. 1996) is the extra cost for avoiding the "hardwired" fixed client buffer constraint. The Server Smoothing algorithm can generate majorization-minimal disk sequence with several buffer constraints, including the fixed buffer of network smoothing algorithms as a special case.

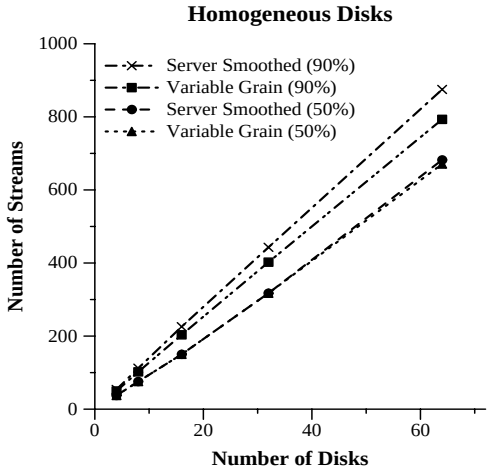


Figure 7.4: With the mixed workload and Variable-Grain Striping (with/without Server Smoothing), the sustained number of active streams increases almost linearly with the number of disks. Although at system load 50% all the submitted streams are accepted, at load 90% Server Smoothing increases the number of active streams by about 10%. This benefit is maintained across different numbers of disks.

7.2 Study of Homogeneous Disks

We start with a study of disk arrays consisting of functionally equivalent disks. In Figure 7.3, we depict the disk time and buffer space proportions in each round for a particular stream. Without smoothing, the occupied buffer space is the minimum necessary for data staging during disk and network transfers. With Server Smoothing, data are prefetched into the server buffer according to the maximum-proportion constraint. This keeps the maximum buffer space proportion constrained above by the maximum disk time proportion (8.7% in this example).

In Figure 7.4 we can see the sustained number of active streams achieved at different system loads and array configurations of size from 4 to 64 disks using the mixed workload. In all the cases shown, the stream data have been striped according to the Variable-Grain Striping method. The Server-Smoothed plots show the performance benefit of applying the Server Smoothing algorithm assuming 256MB of available server buffer space for each extra disk (we try later other server buffer sizes). Although at moderate load $\rho = 50\%$ plain Variable-Grain Striping allows all stream requests to be accepted, at a higher load $\rho = 90\%$ the Server Smoothing can improve throughput by over 10%. The corresponding rejection rate (not shown) is 25%

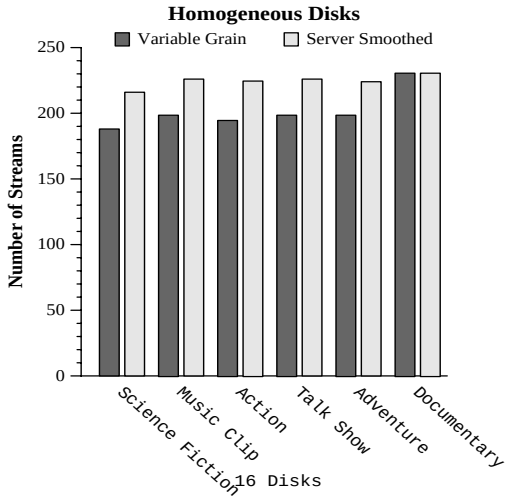


Figure 7.5: The advantage of Server Smoothing when applied to Variable Grain Striping can exceed 15% (Action) depending on the stream type. The load was set to 90% on sixteen disks and 256MB per disk were assumed.

with Server Smoothing, and 41% with plain Variable Grain Striping, at 90% load. With 64 disks, the Server Smoothing algorithm achieves 875 concurrently supported streams at 90% load. This is about 75% of the maximum theoretical capacity of the system, as defined in the previous chapter.

From results for individual stream types in Figure 7.5, we find that the benefit of Server Smoothing depends on the variability of data transfers across different rounds. Thus, although smoothing adds no benefit for streams with negligible variability (Documentary), as variation becomes higher, the increase in the number of streams can exceed 15% (Action).

In Figure 7.6, we show the average disk time $T_{disk}(i,k)$ that was reserved on a particular disk ($k = 0$) during the measurement period $3,000 \leq i < 9,000$. While in most stream types the average disk time hardly exceeds 80% of the round time with plain Variable-Grain Striping, it consistently approaches 90% and in several cases exceeds 93% (Action, Music Clip, Talk Show) when Server Smoothing is applied. This is remarkably high when compared to the 96% average time corresponding to Documentary with inherently very low variability of data transfer sizes across different rounds. The reason that we do not achieve reservation of disk busy time equal to the entire round duration (100%), is due to the large size of the individual disk transfer requests that cannot fit in the remaining fraction of the round duration. Another reason is that

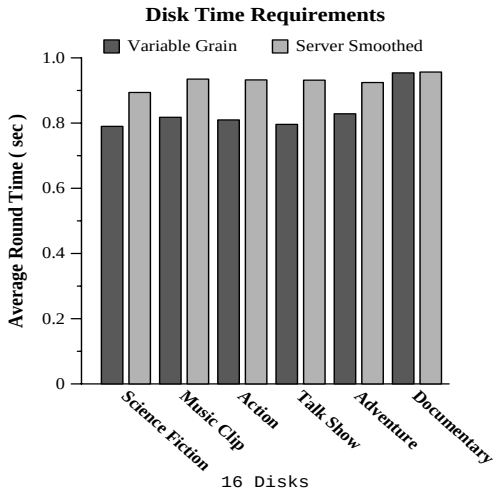


Figure 7.6: With sixteen disks and 90% system load, the average disk time reserved each round increases from less than 80% to over 90% with Server Smoothing and server buffer 256MB per disk. Although the disk time shown corresponds to one of the disks (Disk 0) it was similar (within 2%) across the disks of the array.

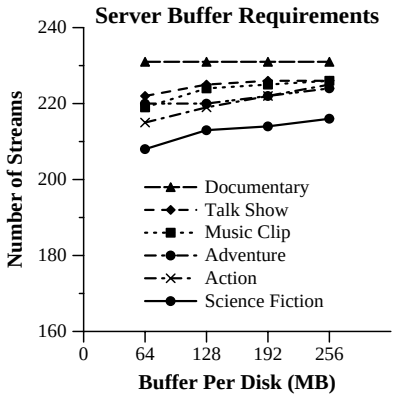


Figure 7.7: With buffer space (per disk) set to 64MB, more than half of the total benefit of Server Smoothing can be achieved (see also Figure 7.5). Increasing the buffer space to 256MB further improves the number of streams in Science Fiction and Action types although at a diminishing degree.

transfer size variability is not completely eliminated, when smoothing is applied under buffer constraints.

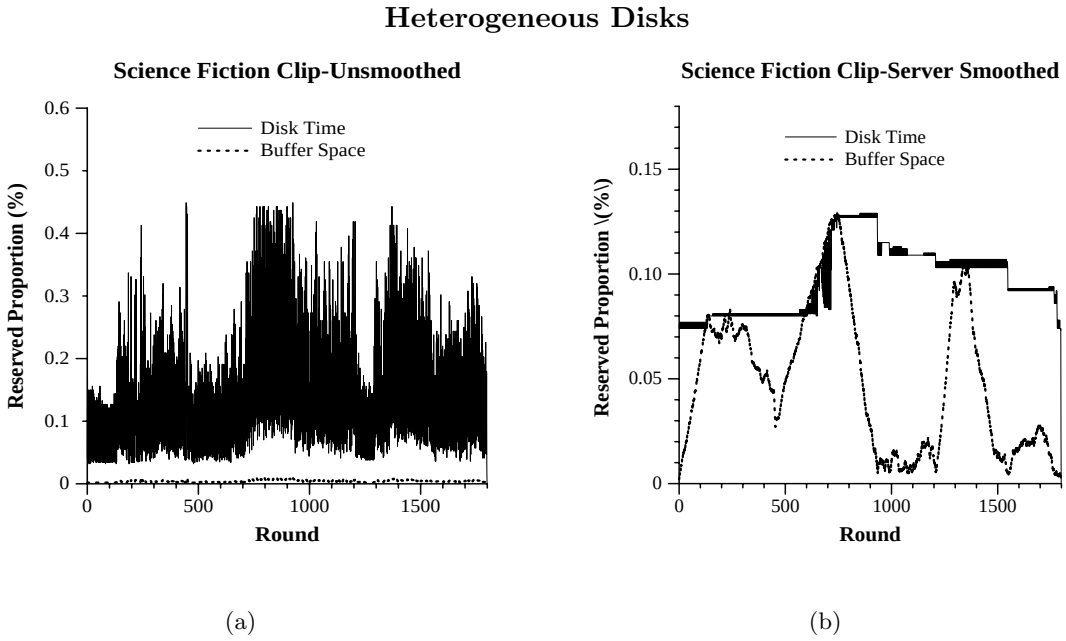
Statistics gathered across the different stream workloads showed that the average occupied proportion of the available buffer space was about 50%, and the maximum hardly exceeded 60% at 90% load. Although in individually smoothed streams the buffer space requirement is allowed to reach that of disk bandwidth (in terms of proportions), the aggregate buffer space requirement

turns out to be lower. This is a result of the way resource requirements of individual streams are multiplexed during system operation. The original constraint of preventing excessive prefetching from overflowing the available buffer space is still satisfied. Further increasing the aggregate buffer demand, without the buffer becoming a potential bottleneck in the system, would require incorporating into the algorithm information about the way stream requests are multiplexed.

We investigated this issue by allowing the buffer requirement of individual streams to exceed the corresponding disk bandwidth requirement (in terms of proportions) by a specific percentage (10%, 20% and 50%). As expected, this increased accordingly the aggregate buffer space utilization. The number of accepted streams did not increase significantly though, and remained within 1-2% of what we report here. Allowing the stream buffer requirements to increase further, would only reduce the number of accepted streams due to shortage of server buffer space.

In order to explain intuitively the low sensitivity of the smoothing effect to the available buffer space, we should look at the shape of the disk bandwidth and server buffer profiles in Figure 7.3. We notice that server smoothing generates a disk bandwidth profile consisting of flat areas, and a server buffer profile consisting of triangular peaks. Multiplexing of buffer profiles corresponding to different (or time-shifted) stream requests can hide such peaks. As a result the aggregate buffer requirements are lower than the sum of those of the individual streams. On the other hand, superimposing disk bandwidth profiles of different (or time-shifted) stream requests leads to aggregate requirements that are the sum of the requirements of individual streams. Due to the particular shape of the disk bandwidth profiles, further reduction of the maximum disk bandwidth requires significant increase in the maximum available buffer space (as shown in Figure 7.7). In other words, the peaks of the buffer will need to move considerably higher in order to keep in memory any additional prefetched blocks. This explains the relative insensitivity of the system throughput to small variations in the upper bound of the buffer space allowed to be used by individual streams.

In addition, we can conclude from Figure 7.7 that more than half of the Server Smoothing benefit is achieved with server buffer size as low as 64MB per disk. We still find our assumption of 256MB server memory (per disk) in the smoothing process to be reasonable though. The additional performance benefit from extra memory is sustained across different system sizes as



(a) (b)

Figure 7.8: Example of stream Server Smoothing, in disk array configuration with Seagate and HP disks in alternating order. The much lower bandwidth of the older HP disk model leads to disk time proportion exceeding 40% in some of the rounds. When Server Smoothing is applied, disk transfers are appropriately adjusted to smooth out the peaks and keep the maximum reservation below 13%. At the same time, the maximum server buffer proportion is constrained to never exceed the maximum disk time proportion.

has been shown in Figure 7.4, at a purchase and administration cost of server memory that is only fraction of what is required for high-end disk drives.

7.3 Study of Heterogeneous Disks

It has traditionally been assumed that disk arrays consist of homogeneous disks, presumably in order to keep the system complexity manageable. With the scalability of stream striping demonstrated recently (Bolosky et al. 1996; Anastasiadis et al. 2001a) and the sequential access of stored video making things somewhat simpler, it is interesting to investigate the possibility of combining different disk types for incrementally creating larger disk arrays. With fast disk technology advances, newer disk models are expected to achieve higher transfer rates and larger storage capacities.

The expected ratio between disk capacity and bandwidth increases and disk accesses are

HP-C3323			
Data Bytes per Drive	1,052,491,776	Track to Track Seek	< 2.5 msec
Data Sectors per Track	72 to 120	Maximum Seek	22 msec
Data Cylinders	2910	Rotational Latency	5.56 msec +/- 0.5%
Data Surfaces	7	Internal Transfer Rate	
Zones	8	Inner to Outer Zone Burst	4.0 to 6.6 Mbytes
Buffer Size	512 Kbytes	Inner to Outer Zone Sustained	2.8 to 4.7 Mbytes

Table 7.1: Features of the HP SCSI disk that is included in the experiments for heterogeneous environments.

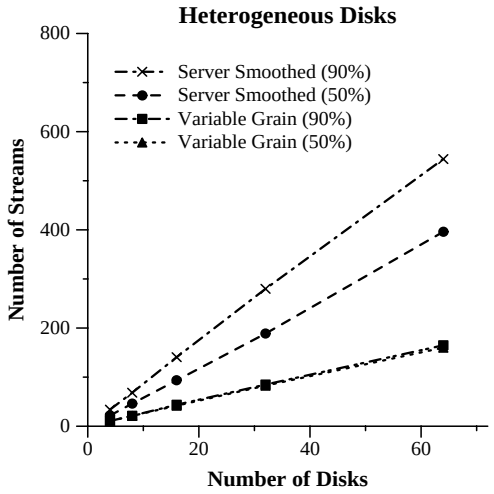


Figure 7.9: With the mixed workload and plain Variable-Grain Striping, the sustained number of active streams remains the same as the load is raised from 50% to 90%. Instead, when Variable-Grain Striping is combined with Server Smoothing the number of streams increases by a factor of 2 at 50% and more than a factor of 3 at 90% load, when compared to that achieved by plain Variable-Grain Striping. Server buffer space of 256MB per disk has been assumed.

expected to become more significant over time (Gray and Shenoy 2000). In this section, we study the case of striping stream data across heterogeneous disk arrays, with the objective of maximizing the number of active streams by increasing the disk bandwidth utilization across all the disks. This might lead to suboptimal storage capacity utilization, which, we assume, is affordable given the current technology trends.

In our experiments, we assume disk arrays consisting of Seagate (Table 5.1) and older HP disks in alternating order. The features of the HP disks are shown in Table 7.1. We note a

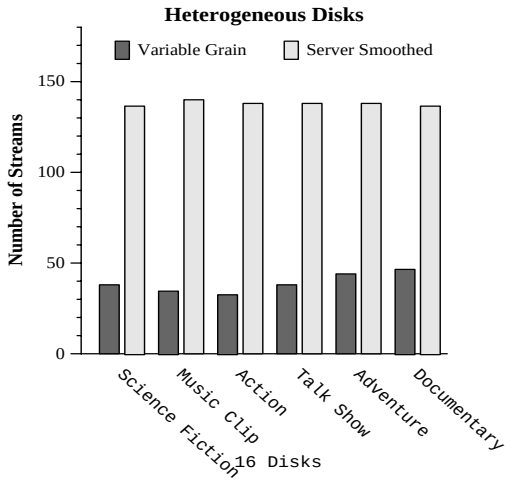


Figure 7.10: Applying Server Smoothing on different stream types, can lead to an increase in the number of streams by more than a factor of 3 at 90% load and server buffer 256MB per disk.

striking difference in the minimum internal transfer rate, which is 2.8MB/sec for the HP disks, one fourth as much as the 11.3MB/sec of the Seagate disks. Such a difference only makes more challenging the balancing of the system load. Although the experiments in this section assume equal number of disks belonging to different disk types, we also tried other ratios in the number of disk types with similar results.

We combined these particular models in the heterogeneous configuration because the corresponding simulation models are available in the DiskSim package for our validation (Section 7.4). The high bandwidth utilization achieved across the different disks means that when N slow disks of bandwidth R_{disk}^{slow} are combined with N fast disks of bandwidth R_{disk}^{fast} they can replace a number of fast disks equal to:

$$N + N \times \frac{R_{disk}^{slow}}{R_{disk}^{fast}}. \tag{7.1}$$

In Figure 7.8(a) we depict an example of a stream striped across an heterogeneous disk array. The lower transfer rate of the HP disks creates peaks of disk time proportion that can exceed 40%. In order to alleviate this problem, we extend the Server Smoothing algorithm to handle heterogeneous disks. In particular, we redefine the disk time $T_{disk}(X)$ and the disk time proportion function $P_d(X)$ to accept a second disk type argument k that specifies the particular disk parameters to be used $P_d(X,k) = \frac{T_{disk}(X,k)}{T_{round}}$. During the operation of the Server

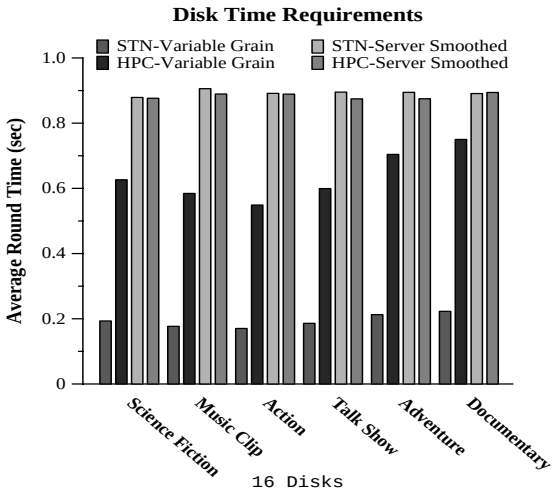


Figure 7.11: The two leftmost bars of each stream type, show the average reserved disk time for the Seagate (STN) and HP (HPC) disks, assuming plain Variable Grain Striping and 90% load. The lower transfer bandwidth of the HP disk creates a bottleneck keeping the reserved disk time of the Seagate disk to less than 25% of the round time. As is shown by the two rightmost bars though, with Server Smoothing both disk types attain average disk time close to 90% of the round time.

Smoothing algorithm, the disk type k assumed in each round i can be derived using a simple rule, such as $k = i(mod\ D)$, where D is the total number of the disks. Finally, if R^k_{disk} is the minimum internal transfer rate of disk k, the service rate definition of Equation 5.1 becomes: $\mu = (T_{round} \cdot \sum_{k=0}^{D-1} R^k_{disk})/S_{tot}$.

We applied the extended Server Smoothing algorithm to the stream example of Figure 7.8(a). The generated transfer sequence shown in Figure 7.8(b) had the buffer space proportion bounded by the disk time proportion, as before. In addition the maximum disk time proportion dropped from over 40% to less than 13%, after the transfer sizes across different rounds were appropriately adjusted according to the available disk bandwidth. From Figure 7.3, the corresponding maximum disk time proportion was about 8.7% in the homogeneous disk case. Hence, in the heterogeneous case, the fact that half of the disks have much lower disk bandwidth creates only moderately higher disk time demand after smoothing is applied.

In Figure 7.9, we can compare the performance of plain Variable-Grain Striping to that of Variable-Grain Striping with Server Smoothing in a range of heterogeneous disks from 4 to 64. Although the number of streams always increases almost linearly with the number of disks,

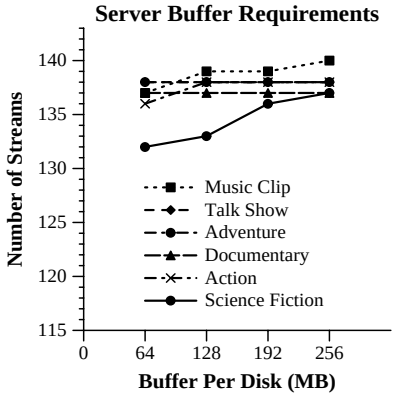


Figure 7.12: With the server buffer (per disk) set to 64MB, most of the benefit of Server Smoothing can be attained (see also Figure 7.10). Increasing the server buffer from 64MB to 256MB increases only marginally (less than 5%) the sustained number of active streams.

Server Smoothing can achieve an advantage that exceeds a factor of 2 and 3 respectively at loads 50% and 90%. The reason is that the limited disk bandwidth of the HP disks, prevents the Seagate disks from attaining high bandwidth utilization without appropriate adjustment of the disk transfers. A similar behavior is also demonstrated across different stream types in Figure 7.10. With plain Variable-Grain Striping, the number of supported streams on sixteen disks hardly exceeds 50; when Server Smoothing is added the number of streams gets close to 140.

In Figure 7.11, we depict the average time that the Seagate and HP disks are expected to be busy respectively during each round. We show the statistics for the first two disks only, since the statistics for the rest of the disks were respectively similar. As we see, the average time that the Seagate disks are expected to be busy is less than 25% of the round time with plain Variable-Grain Striping. The reason is the low bandwidth of the HP disks, whose corresponding average time varies between 50% and 80%; it cannot get higher due to the relatively large data requirements of the individual streams. When Server Smoothing is applied, a high average reserved disk time that gets close to 90% is achieved across all the disks of the disk array. This becomes possible with appropriate data prefetching that distributes data accesses across the disks according to the bandwidth that they can support.

In the previous experiments we set the server buffer to 256MB per disk. However, as we can

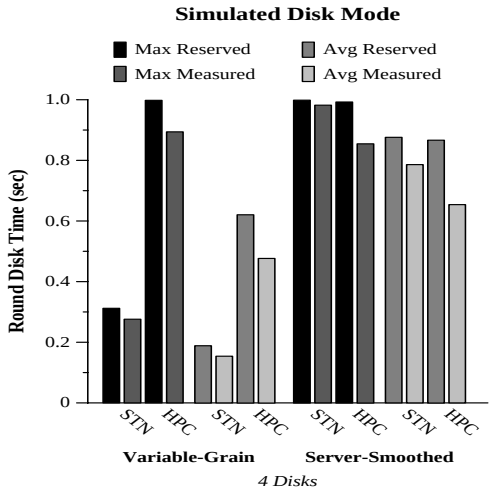


Figure 7.13: In an array of four disks, Seagate (STN) and HP (HPC) models are used in alternating order. The average and maximum reserved and measured time is shown for the first two disks of the array with the mixed workload at 90% load. On the STN disks, the reserved statistics are no more than 8% higher than the measured. On the HPC disk, the corresponding difference can get up to 20%. The measurements have been done using the detailed disk simulation models by Ganger et al.

see from Figure 7.12, 64MB per disk are enough to get most of the benefit of Server Smoothing for the particular streams included in our benchmark.

7.4 Validation in Simulated Disk Mode

In order to keep the computation time reasonable, the previous experiments were conducted with our system in Admission Control mode, where playback requests arrive leading to corresponding resource reservations, but without actual time measurement of the individual disk transfers. In the present section, we compare the statistics of the disk time reservations with the statistics gathered over the access times of all individual data transfers involved, using the DiskSim representation of the Seagate ST-34501 and HP C3323A disks (Ganger et al. 1999). A four-disk array model is used with the disk types in alternating order. Each disk is presumed to be attached to a separate 20MB/sec SCSI bus, and no contention is assumed on the host system bus connecting the SCSI buses. The statistics are gathered during 6,000 rounds after a warmup period of 3,000 rounds, as before. The mixed workload is used with average number of active

streams 9.74 for plain Variable-Grain Striping and 33.87 for Server-Smoothed Variable-Grain Striping, respectively, corresponding to 90% load.

As can be seen from Figure 7.13, in both the average and maximum case, the reserved disk time is no more than 8% higher than the corresponding measurements on the Seagate disk model by Ganger et al. (Ganger et al. 1999). This gap can be attributed to the fact that our disk time reservation assumes a minimum disk transfer rate and ignores on-disk caching. The corresponding gap for the HP disks gets close to 15% with plain striping and 20% with Server Smoothing. Possible reasons for the larger discrepancy of the HP disks are the increased on-disk cache locality due to the smaller disk capacity, and the higher probability that only one head movement is required with the smaller data transfers (smoothed case).

Practically, the achieved accuracy in the disk time predicted by the resource reservation is adequate. In fact, to improve these estimates, it would probably be necessary to have extra disk geometry information that is not readily available in commercial disk drives (Worthington et al. 1995).

7.5 Summary

In this chapter, we introduced the Server Smoothing algorithm for variable bit-rate streams that uses prefetching into the server buffers for smoothing out disk data transfers. Experimentation with both homogeneous and heterogeneous disk arrays demonstrated that the average reserved disk access time can exceed 90% of the round time across the different disks. A significant benefit in the number of streams was sustained across different sizes of disk arrays that we examined.

Chapter 8

Remaining System Operation Issues

In the present chapter, we discuss several issues related to the system operation. These include the procedure of video stream acquisition and alternative approaches for tolerating disk and node failures. In addition, we introduce techniques for resequencing data packets received by the client, possible ways for reorganizing data during system expansion, and ideas for reducing the computational cost of admission control. The treatment of these problems is by no means exhaustive, and we leave for future work any further investigation and experimentation related to them.

8.1 Acquiring Video Streams

Storage of media content into a network server is a basic operation that has to take place before the actual playback service becomes possible. Service providers are expected to receive the digital material either through a network connection, or recorded on some storage media (e.g. tape or optical disc). For access flexibility, the media files can be temporarily transferred to single hard disks attached to computer nodes, called here *Source Nodes*. In general, the source nodes are different from the server components described previously.

Before striping the media files, the decoder corresponding to each media format has to be used in order to identify the size of the individual media units (sound samples or video frames) along each file. This information determines the data transfer requirements in each

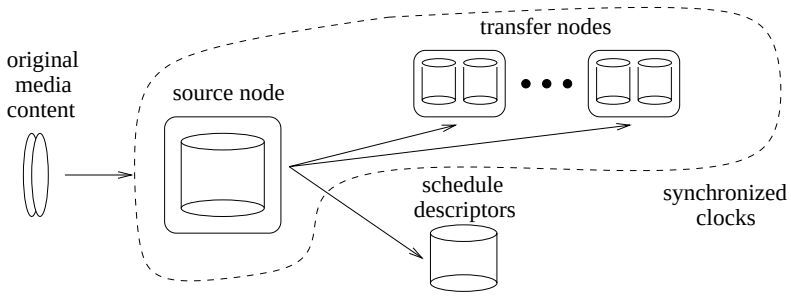


Figure 8.1: Stream data are temporarily staged in a source node disk. After the schedule descriptor generation, transfers occur into the system disks through the transfer nodes.

round during playback. It can be transformed into an actual schedule descriptor according to the target server configuration, the minimal client hardware supported, and the striping or smoothing policy used. During recording, the schedule descriptor specifies the amount of data that should be stored on each disk.

After the schedule descriptor becomes available, a recording request can be submitted to the admission control module that will allow stream data transfer from the source node to the transfer nodes of the server (Figure 8.1). Before recording starts, the required disk, network and buffer resources have to be reserved in order for the system operation to proceed without interruption. Depending on the resources that are available, stream recording can be done faster, slower or at the same speed as the corresponding playback.

One challenge in the media recording process is time synchronization between the source node and the transfer nodes. In order for the transfer schedule to be met, a specified amount of data should be sent from the source node to each transfer node in each round. We can assume that the mechanism used for synchronizing the transfer nodes can be extended to include the source node. The time synchronization problem has been studied extensively during the last decades in distributed systems, and several solutions are available that have been demonstrated to achieve time accuracy within a few milliseconds between different computers (Mills 1991).

8.2 Data Replication for Reliability

Despite recent improvements in the reliability of storage devices and computer systems, commercial server installations are still expected to experience device failures due to the large number of components that they involve. In fact, although the Mean Time To Failure of a modern disk is estimated to be $MTTF_{disk} = 1,000,000$ hours, a practical combination of $D = 256$ such devices reduces the corresponding $MTTF_{array}$ to (Patterson et al. 1988):

$$MTTF_{array} = \frac{MTTF_{disk}}{D} = 162\ days, \tag{8.1}$$

when failure independence is assumed between different disks. The general problem of disk array reliability has been studied extensively during the previous decade in the context of both traditional database workloads and constant bit-rate streams (Holland and Gibson 1991; Chen et al. 1994; Berson et al. 1995; Ozden et al. 1996; Gafsi and Biersack 2000). However, the potential of improved performance that is introduced by variable bit-rate streams, makes it interesting to consider any additional complications involved in that case for providing fault-tolerant operation.

When designing data replication techniques for reliability, one basic objective is to minimize the extra computation, storage and bandwidth requirements during normal and failed operation, in comparison to the nonredundant case. If a device fails, the system load that would be normally served by that device should be equally divided across the remaining components of the system. We begin our study by considering one faulty disk, before we proceed with the more general case of failures on entire transfer nodes or multiple disks.

8.2.1 Mirroring-Based Schemes

In the data replication technique called *Mirroring*, the data of each disk are replicated on one or more different disks. When one disk fails, the corresponding data remains available by retrieving replica data from the rest of the disks. The required storage space is roughly double and the needed bandwidth from each disk at most twice that of the nonredundant case.

In traditional storage systems, the data are striped in fixed-size blocks across multiple disks. The replica of each block can be stored in its entirety on a different disk, which requires only

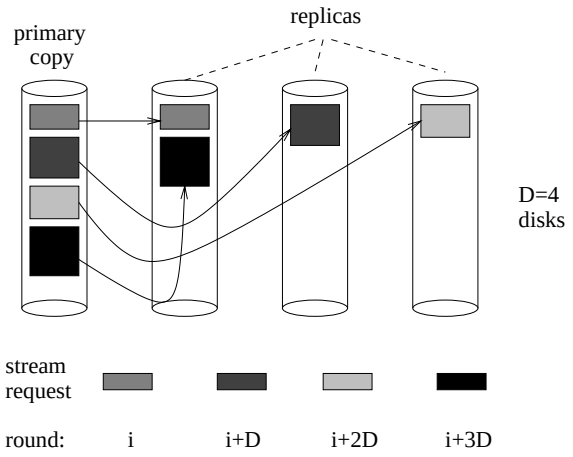


Figure 8.2: Stream data of different rounds stored on one disk are replicated round-robin across the rest of the system. In the case that this disk fails, the corresponding load is expected to be fairly handled by the surviving disks.

one access in the case of failure and minimizes the disk transfer overhead. The alternative of declustering a data block replica across multiple disks can potentially better balance the extra disk access load, but incurs the additional overhead of multiple transfers in the case of a disk failure.

In principle, the above data replication approach can also be applied with variable bit-rate streams striped across multiple disks, when a fixed block is used as with Fixed-Grain Striping. Extra complications might be introduced with Variable-Grain (and Group-Grain) Striping, where the amount of data stored on each disk varies according to the request size in each round. However, we have already demonstrated that the long-term load is equally balanced across the system regardless of which of the three striping policies is actually used.

One way to ensure that the load-balancing property remains valid even with a failed disk would be to replicate the round requests of one disk round-robin across the rest of the disks (Figure 8.2). With the detailed resource reservation that we use, this also implies that on average the total extra bandwidth required for tolerating a disk failure is equal to the bandwidth of the failed disk. This is because, in the case of the disk failure, the respective access load will be equally distributed across the surviving disks (Fig. 8.2).

The above feature of detailed resource reservation is an important advantage with respect

to worst case resource reservation schemes that normally have to keep unused up to half of the total bandwidth of each disk (Gafsi and Biersack 2000). Declustering data replicas of a disk on a subset of the remaining disks addresses only partially this issue, while incurring the extra cost of multiple accesses for a stream in each round when a disk fails (Bolosky et al. 1996).

Although we expect that replicating stream requests from a disk in their entirety on different disks should keep the system load balanced, it would be interesting to experimentally investigate that and compare it against the data declustering approach.

8.2.2 Parity-Based Schemes

Data replication techniques based on parity have also been previously proposed in order to trade extra disk bandwidth or memory buffer for reduced storage space requirements. Since, with current technology, disk storage space is considered the cheapest of the three resources above, it has been recently suggested that mirroring rather than parity should become the preferable data replication technique for tolerating disk failures (Gray and Shenoy 2000). This argument becomes even stronger for the case of I/O-bound workloads, such as those handled by media servers.

In addition, implementation of parity-based data restoration in a distributed architecture requires additional data traffic among the transfer nodes. This can introduce significant extra complexity and resource requirements in terms of network bandwidth and buffer space that has to be reserved, and actually utilized in the case of failure (Bolosky et al. 1996). For all the above reasons, we decided not to consider parity-based techniques any further here.

8.2.3 Node and Multiple Disk Failures

In a practical setting consisting of multiple transfer nodes, it is possible that a transfer node fails. By grouping transfer nodes into independent clusters and replicating appropriately entire stream files across them, it is possible to tolerate failures of individual clusters. However, such an approach introduces all the related problems of tracking user preferences and replicating appropriately different streams.

An alternative solution that avoids the stream replication problem gathers all the transfer

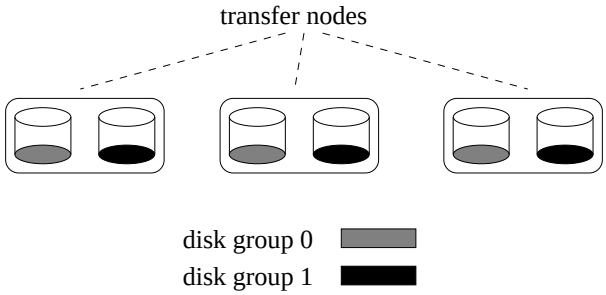


Figure 8.3: Different disks from each node belong to different groups. When a transfer node goes down, data from all its disks can still be restored from replicas available within each group.

nodes in a single cluster. However, the different disks of each transfer node are mapped into different groups (Figure 8.3). Replicas of data from each disk are placed only on disks of the same group. Within each disk group, mirroring-based schemes can be applied, as described before. Even when an entire node fails, data from each missing disk can still be restored by using replicas from their corresponding disk groups. A similar technique is already used in existing video servers for constant rate streams (Bolosky et al. 1996; Gafsi and Biersack 2000).

The previous argument remains valid when multiple disks belonging to different transfer nodes fail, provided that each failed disk is member of a different disk group. We now consider the more general case of multiple disk failures occurring in the same disk group. The mean time to failure of a disk array (or disk group) that can tolerate only one failed disk is estimated to be equal to (Patterson et al. 1988):

$$MTTF_{mirroredArray} = \frac{(MTTF_{disk})^2}{D(D-1)MTTR_{disk}} = 1,748\ years \tag{8.2}$$

with $D = 256$ and Mean Time To Repair $MTTR = 1\ hour$. In other words, the probability that a second disk fails in a disk group before a previously failed disk is replaced is negligible, assuming that failures occur independently of each other. Therefore, we do not consider this case any further here.

8.3 Reordering of Packets Received at the Client

In the system design described in the previous chapters, the network transfer schedule of each stream is allowed to be different from the disk transfer schedule, provided that basic constraints are met for timely data decoding at the client. This decision essentially permits different transmission policies to be applied to the two transfer media, according to the corresponding resource availability and economics of each of them. It is enabled by the server buffer memory within each transfer node, where data can be staged when they are retrieved from the disks earlier than needed.

One complication that the above flexibility introduces is the possibility that, during a round, the data arriving to a client may originate from multiple transfer nodes. Then, the datagrams from the different transfer nodes potentially arrive at a client in an order that is different from the stream data sequence. Note that this problem is distinct from the case that datagram fragments arrive out of order when fragmentation and reassembly take place at the IP level.[1] The datagram fragments themselves are put in the right order automatically by the network protocol code at the receiving side.

There is a case that stream data are striped across the disks according to their rate of decoding, without any disk transfer smoothing. Then, the above issue can be completely avoided by quantizing the network transfers with respect to the logical block size B_l. This process transforms the network schedule to an exact shifted version of the disk transfer schedule.

However, when disk transfer smoothing is applied, the above simplification is no longer effective, and it is expected that the data required in a round by some client will be staged in the buffers of multiple transfer nodes. In that case, each individual datagram can be tagged with some sequence number that is unique within the duration of each stream. The sequence numbers are determined during the schedule generation process, and are stored with the schedule descriptor of each stream.

When network datagrams are received out of order at the client, there is only a minimal overhead for resequencing them given the sequence numbers attached to each of them. A

[1] A simple transport protocol such as UDP/IP is considered adequate for the requirements of stream data network transfers.

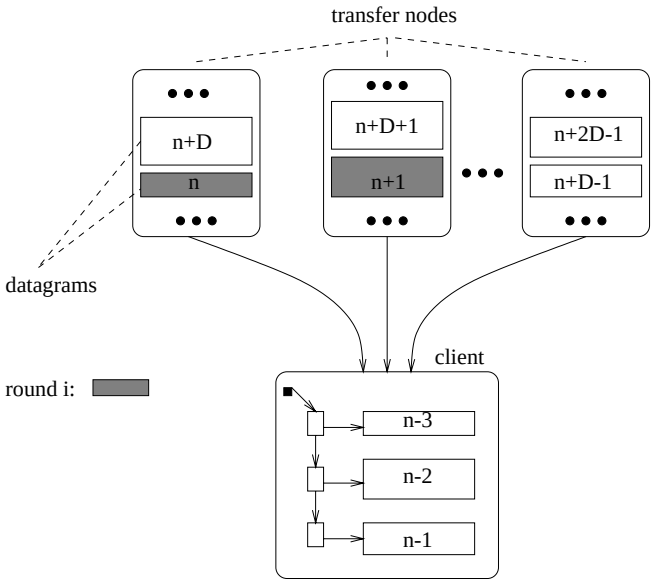


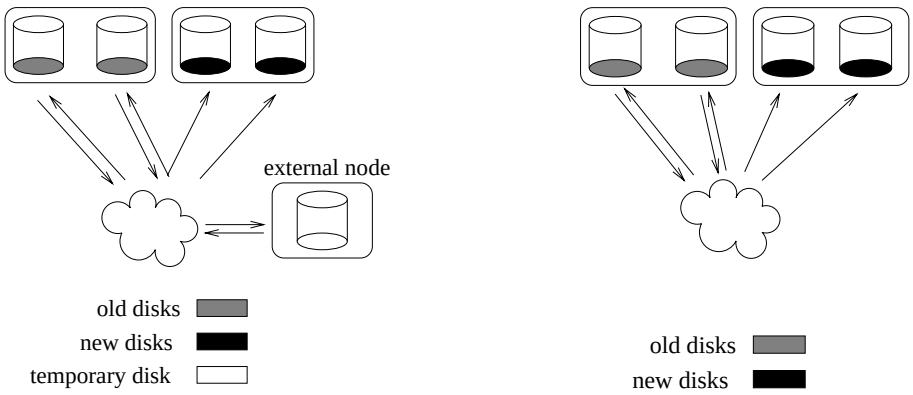
Figure 8.4: Individual datagrams corresponding to each client are assigned sequence numbers that are unique within each stream. Packets arriving to a client out of order can be resequenced according to the attached sequence numbers, which are decided offline during the schedule descriptor generation.

possible implementation keeps the arriving datagrams in the right order using a linked list (Figure 8.4). This partially replicates the IP datagram fragment resequencing code (Comer and Stevens 1995). Only minor modifications are required in the stream data decoder to read bits from multiple buffers instead of a single one.

Although in the previous discussion we assumed that datagram resequencing takes place at the client, a similar functionality could be located in a proxy between the client and the server. Such an approach would keep the client software simpler.

8.4 Data Reorganization for System Upgrades

Basic objective in online data reorganization is the ability to dynamically change the placement of data across multiple storage devices, according to the expected access frequency of the data, and the performance characteristics of the available storage devices (Borosky et al. 1997). Traditionally, the problem has been examined extensively in the context of transaction processing workloads in database systems. It has also been studied in media servers storing constant rate



(a) Using External Nodes (b) Directly

Figure 8.5: Stream data can be restriped either a) through an external node, or b) directly across the different devices. The actual data transfers are controlled by the transfer nodes to which the disks are virtually or physically attached.

streams (Dan and Sitaram 1995; Wolf et al. 1995; Dan et al. 1995; Bolosky et al. 1997).

In this section, we deal with the more specific case that variable bit rate streams need to be restriped across multiple disks. Such a need can arise due to either extra disks and transfer nodes added to a media server, or existing disks being upgraded. These modifications are alternative ways for increasing the storage space or improving the throughput of the system. We assume that the amount of data stored on each disk is appropriately determined offline using techniques similar to the smoothing algorithm introduced in the previous chapter. Our focus here is on the mechanisms of the actual data movement that is necessary during stream restriping.

The flexibility of adding extra devices without interrupting the operation of the system depends significantly on the interconnect technology that attaches the disks to the transfer nodes. Virtual assignment of disks to hosts through storage-area or general-purpose networks makes it easier to change the storage topology, when compared to fixed, physical attachment of disks to specific hosts through a local I/O bus. Virtual attachment permits the placement of new disks in arbitrary order with respect to currently installed disks, which can facilitate balancing the access load among them in a heterogeneous environment. After the new disks have been

physically added to a server, new schedule descriptors can also be generated for already stored streams taking into account the new hardware configuration parameters. Finally, physical data transfers have to take place in order for the new schedule descriptors to become effective.

During the normal recording/playback operation, all the transfer nodes are connected through the high speed network to a particular external node, for the actual data transfers to occur. However, in the case of stream restriping, each transfer node should be able to exchange data directly with any other transfer node.

One possibility for resolving the above extra complication is to temporarily transfer a stream file to a single disk attached to an external node, using the protocol already used for sending data to clients (Figure 8.5(a)). Subsequently, a new stream replica can be recorded using the new schedule descriptor, in a way similar to that used for the original storage of the streams on the system. The advantage of this approach is its simplicity. Connections are established from the external temporary node to all the transfer nodes, as with regular clients. A possible drawback is the extra resources that are required by the external nodes.

Alternatively, it is possible to restripe the stream data across the new hardware configuration directly from the already stored stream files (Figure 8.5(b)). This means that each transfer node can exchange datagrams directly with any other transfer node, in order for the playback and recording to take place. A mechanism for achieving a similar function on distributed servers for constant rate streams has previously been described (Bolosky et al. 1997). One difference from that work is that in our system the exact new disk block locations of the stream data are selected online by the metadata managers, instead of being determined centrally before the recording. Our approach can reduce significantly the preprocessing overhead actually required for the data reorganization.

Other issues that have to be considered when implementing a direct restriping approach, is the expansion of the schedule descriptor of each stream to allow specifying multiple recipients (transfer nodes) of the transmitted data, as opposed to a single one (client). The data transfer itself involves playback and recording sessions working concurrently for each restriped stream. Its duration depends on the available bandwidth that can be devoted to the system management and could be different from that involved in stream playback.

8.5 Reducing the Cost of Admission Control

The storage descriptor corresponding to a stream file and system configuration pair is stored in one or more nodes (*schedule nodes*) separately from the stream data. It specifies the amount of data that should be retrieved from a single disk (with Variable-Grain Striping), the amount of data that should be sent from one or more nodes through their network interfaces to the client, and the buffer space that is required on each transfer node. Although this detailed specification is necessary for initiating the actual data transfers, the admission control test could be based on summarized versions of the above information. For example, the admission control process could be accelerated significantly, if instead of using data amounts transferred in a single round, it is based on the maximum amount of data required over a number of rounds. However, summarized information can reduce the accuracy of the resource reservations, as well.

In addition, some portions of the admission tests could be streamlined. We could assume that the striping policy guarantees the memory buffer never to become a bottleneck. This follows from the trivial memory requirements in the unsmoothed case, and the memory buffer constraint of the Server Smoothing algorithm. Also, the data sent to a particular client during a round should originate from a limited number of transfer nodes. This number cannot exceed two nodes in the unsmoothed case, while in the smoothed case it can be derived from the ratio between the size of the maximum network transfer and the average disk transfer. In addition, only one disk is accessed per round per stream. Therefore, the computation cost of admission control becomes bounded and independent of the total number of disks or transfer nodes. This observation further improves the scalability properties of the system architecture that we propose.

8.6 Summary

In this chapter, we explained the sequence of steps that is required during stream recording. Then, we discussed the extra complications that have to be handled for tolerating hardware component failures in network servers for variable bit rate streams. Subsequently, we considered the issue of network packets sent from multiple nodes and arriving out of order at a client

during a round. Then, we described potential schemes for restriping stream data as a result of upgrades in the hardware configuration of a system. Finally, we argued that practically the cost of admission control for a stream can be independent of the server size and the total amount of resources involved.

Chapter 9

Conclusions and Future Work

9.1 Contributions

It is our thesis that building scalable and efficient media servers is feasible. In the present study we focus on the support of variable bit-rate video streams, because they have been shown to reduce resource requirements when compared to constant bit-rate streams of equivalent quality. We examine and experiment with storage management issues using a prototype system that we built.

In the *Exedra* media server architecture that we propose, continuous stream playback is guaranteed through deterministic reservation of resources. The probability of overloads and data losses within the server is minimized by keeping detailed account for the buffer space and the disk or network transfer delays corresponding to each accepted stream over time. Quality of service guarantees are offered, while keeping the utilization of the server resources high.

Using the stride-based disk space allocation scheme, the disk transfer sizes corresponding to each stream can change over time at a configurable granularity. The estimated head movement overhead is kept bounded, while internal and external fragmentation issues are handled in an efficient way.

Separating the admission control from playback dispatching allows increased flexibility in delaying playback initiations after stream requests are admitted into the system. The dispatching functionality can be distributed across the different transfer nodes, thus reducing resource

requirements in the admission control nodes.

The metadata management is handled by a different module for each disk. This simplifies significantly the system structure, and enables the system to operate with heterogeneous disks. The Circular Scan disk scheduling policy is adapted to the round-based operation of the system. The use of two different priority queues for each disk increased system resilience to round length overloads during disk transfers.

Memory buffers are allocated contiguously in virtual memory for each request, which keeps the disk transfer bandwidth high, and simplifies the performance tuning complexity of the system. Aggressive deallocation of individual buffers is permitted for improved buffer space utilization.

We describe a method for performance evaluation of media servers, that can be applied to servers supporting streams with different requirements, and servers with different transfer capacities. We define several parameters of the system, and study their effect on system throughput and rejection ratio, as system load changes. 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. We also introduce the Group-Grain Striping policy, as a generalized version of Variable-Grain Striping. In experiments that we do with MPEG-2 streams, we demonstrate an almost linear increase in the supported number of concurrent users, as the number of disks increases. This property of almost linear striping scalability remains valid across the different striping policies.

Variable-Grain Striping is shown to outperform Fixed-Grain Striping by 38-50% for stream types with reasonable variability, on sixteen disks. With projection based on previous trends in disk technology, we find a significant advantage of Group-Grain Striping over Fixed-Grain Striping two and five years into the future.

Proliferation of client devices with varying hardware configurations motivates the development of resource management policies that make minimal assumptions about the available client resources. We introduce the Server Smoothing algorithm for variable bit rate streams, that uses prefetching into the server buffers for smoothing out disk data transfers. Experimentation with

homogeneous disk arrays and moderate server buffer space shows that Server Smoothing can achieve 12-15% increase in the number of streams supported by the server, for streams types with reasonable variability. This benefit is sustained across different sizes of disk arrays that we examined.

It is also important to consider the efficient operation of a server with heterogeneous disks because this allows server installations to be incrementally expanded using the most advanced and cost-effective storage devices as the system load increases. We used our Server Smoothing algorithm for striping variable bit rate streams across arrays of heterogeneous disks.

We demonstrate that, when plain disk striping is used, disks with lower transfer rates prevent the system from reaching high utilization. Instead, when Server Smoothing is applied, the average reserved disk access time can get as high as 90% of the round time across the different disks. The corresponding benefit in the number of streams accepted by the server exceeded a factor of three for the particular disk array configuration that we used.

In a separate chapter, we outlined a way for replicating data in order to tolerate disk and node failures. We described a method for resequencing data packets arriving out of order at the client. We briefly examined the problem of dynamic data reorganization. We also argued that the computational requirements for the admission control test of each stream could be independent of the server size.

9.2 Issues for further investigation

There are several problems that will probably require further investigation in the future. One of them is examination of the engineering issues involved in building media servers for variable bit rate streams using multiple computer nodes. When a playback request is accepted, network connections are established between each transfer node and the client. Also, each transfer node is appropriately prepared for starting data transfers to the client. The actual disk accesses occur only after a latency period passes that is determined by the admission control algorithm. During playback, it should be possible to pause or completely terminate one stream playback. This implies that appropriate commands are sent to all the transfer nodes, and are handled in

a timely fashion.

Additionally, it would be worthwhile to make an experimental comparative study of alternative fault-tolerance techniques, that are based on data replication. Questions that could be investigated are related to the buffer space and disk bandwidth that each approach requires in normal and failed operation. Balancing the system load across the entire system, even when some devices fail, remains critical for utilizing efficiently the resources. Declustering the replica of a stream disk request across multiple disks distributes the load of a failed disk on multiple other disks, but also reduces the I/O efficiency.

The detailed representation of stream resource requirements that was assumed introduces the need for a study related to the computational cost of admission control. In the previous chapter, we claimed that the cost of the admission control test for a stream can be independent of the actual size of the system, and the number of resources that it comprises. This is a question that could be verified experimentally. More importantly, the number of admission control tests that should be handled per time unit increases linearly with the system capacity. It is possible that the corresponding computational cost for each stream drops, as the system size increases. The reason has to do with the maximum number of initiating rounds that should be considered for each stream before it is rejected (Chapter 7). However, there is still an open question of developing techniques for admission control in large servers and studying their scalability.

In the past, video encoding methods have been developed that allow representation of a stream in multiple layers. Depending on the network capacity and the hardware capabilities of the client, a different number of layers can be retrieved and transmitted. A larger number of layers corresponds to better video quality. Therefore, it would be interesting to adapt the storage methods described in this thesis to handle multi-layer encoded streams as well. We expect that alternative ways of grouping and storing the data corresponding to each layer in each round can have a significant effect in the system structure and the disk access efficiency.

In addition to normal playback, existing media servers usually allow playback initiation at arbitrary positions of a stream or access in fast-forward and rewind modes. Arbitrary position initiation is not straighforward for variable bit rate streams with detailed resource reservation. Complications arise that are related to time-shifting the resource reservations, while keeping the

latency and bandwidth requirements minimal. Fast-forward and rewind modes can be achieved by access to alternate video files that, when decoded, provide the illusion of the above functions. Therefore, they can also be reduced to the previous problem of arbitrary position inititiaton.

Expected improvements in the processing power available on disks, has recently motivated research on how to make secure and efficient data transfers directly from the disks to the client. Although this approach has been mostly focused on general-purpose file systems, similar ideas could also be applied on media server environments. In some sense, this is equivalent to having multiple transfer nodes with only one disk attached to each of them. Depending on the actual assumptions that can be made about the capabilities of each disk, research is required for determining the best way of spliting the media server functionality among the disks, the server nodes and the client.

Appendix A

Summary of Symbols

Symbol	Description
B_p	Disk sector size
B_l	Logical block size
B_s	Stride block size
B_f	Block size in Fixed-Grain Striping
G	Group size in Group-Grain Striping
L_n	Network sequence length
$\mathbf{S_n}(i)$	Network sequence
$\mathbf{S_n}(i,u)$	Network striping sequence
L_b	Buffer sequence length
$\mathbf{S_m}(i)$	Buffer sequence
$\mathbf{S_{mb}}(i,q)$	Buffer striping sequence
L_d	Disk sequence length
$\mathbf{S_d}(i)$	Disk sequence
$\mathbf{S_{md}}(i,k)$	Disk striping sequence
M_i	Active streams in system round i
T_{round}	Round length

continued on next page

continued from previous page	
Symbol	**Description**
$T_{fullSeek}$	Maximum seek time
$T_{trackSeek}$	Track-to-track seek time
T_{avgRot}	Average rotation latency
R_{disk}	Minimum internal disk transfer rate
R_{net}^{u}	Transfer rate at network interface u
$T_{net}^{j}(i,u)$	Reserved time on network interface u in round i for client j
$B^{j}(i,q)$	Reserved buffer space on node q in round i for client j
$T_{disk}^{j}(i,k)$	Reserved time on disk k in round i for client j
$T_{disk}(i,k)$	Total reserved time on disk k in round i
μ	Stream service rate
λ	Stream arrival rate
ρ	System load
H_l	Lookahead distance
H_l^{basic}	Basic lookahead distance
F_l	Lookahead factor
$P_d(X)$	Disk time proportion
$P_b(X)$	Buffer space proportion

Bibliography

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

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

Anastasiadis, S. V., K. C. Sevcik, and M. Stumm (2001c, March). Server-Based Smoothing of Variable Bit Rate Streams. Technical Report CSRG-424, Computer Systems Research Group, University of Toronto.

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

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

Biersack, E., F. Thiesse, and C. Bernhardt (1996, June). Constant Data Length Retrieval for Video Servers with Variable Bit Rate Streams. In *IEEE Multimedia Computing and Systems*, Hiroshima, Japan, pp. 151–155.

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

Birrel, A. D. and R. M. Needham (1980, September). A Universal File Server. *IEEE Transaction on Software Engineering 6*(5), 450–453.

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

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

Bolosky, W. J., D. R. Need, and S. Debgupta (1997, June). *Continuous media file server for cold restriping following capacity change by repositioning data blocks in the multiple data servers*. Washington, DC: U.S. Patent and Trademark Office. Patent No 5,991,804.

Borosky, E., R. Golding, A. Merchant, L. Schreier, E. Shriver, M. Spasojevic, and J. Wilkes (1997, May). Using attribute-managed storage to achieve QOS. In *Intl. Work. Quality of Service*, New York, NY, pp. 203–206. Published in Building QoS into Distributed Systems, edited by Andrew Campbell and Klara Nahrstedt, Chapman & Hall, London UK, 1997.

Buddhikot, M. M. and G. M. Parulkar (1995, April). Efficient Data Layout, Scheduling and Playout Control in MARS. In *Intl. Work. on Network and Operating System Support for Digital Audio and Video*, Durham, NH, pp. 318–329.

Cabrera, L.-F. and D. D. E. Long (1991). Swift: Using Distributed Disk Striping to Provide High I/O Data Rates. *Computing Systems 4*(4), 405–436.

Chang, E. and A. Zakhor (1994, February). Scalable Video Data Placement on Parallel Disk Arrays. In *IS&T/SPIE International Symposium on Electronic Imaging:Image and Video Databases*, San Jose, CA, pp. 208–221.

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

Chang, E. and A. Zakhor (1997, October). Disk-based Storage for Scalable Video. *IEEE Transactions on Circuits and Systems for Video Technology* (5), 758–770.

Chen, P. M., E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson (1994, June).

RAID: High-Performance, Reliable Secondary Storage. *ACM Computing Surveys 26*(2), 145–185.

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

Clark, T. (1999). *Designing Storage Area Networks*. Reading, Mass.: Addison-Wesley.

Comer, D. E. and D. L. Stevens (1995). *Internetworking with TCP/IP: Design, Implementation, and Internals, Volume II*. Upper Saddle River, NJ: Prentice Hall.

Dan, A., M. Kienzle, and D. Sitaram (1995). A dynamic policy of segment replication for load-balancing in video-on-demand servers. *Multimedia Systems 3*, 93–103.

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

Douceur, J. R. and W. J. Bolosky (1999, January). Improving responsiveness of a stripe-scheduled media server. In *ACM/SPIE Multimedia Computing and Networking Conf.*, San Jose, CA, pp. 192–203.

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

FORE Systems, Inc. (1999, November). *ForeRunner HE/LE/200E ATM Adapters for the PC User's Manual*. Warrendale, PA: FORE Systems, Inc.

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

Ganger, G. R., B. L. Worthington, and Y. N. Patt (1999, December). The DiskSim Simulation Environment: Version 2.0 Reference Manual. Technical Report CSE-TR-358-98, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan.

Garofalakis, M. N., Y. E. Ioannidis, and B. Ozden (1998, August). Resource Scheduling for Composite Multimedia Objects. In *Very Large Data Bases Conf.*, New York, NY, pp. 74–85.

Gibson, G. A., D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka (1998, October). A Cost-Effective, High-Bandwidth Storage Architecture. In *Conf. Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, pp. 92–103.

Gilder, G. (1997, April). Fiber Keeps Its Promise: Get ready. Bandwidth will triple each year for the next 25. *Forbes*. www.forbes.com/asap/97/0407/090.htm.

Gray, J. (2000, March 23). Informal Talk, Database Group Seminar, University of Toronto.

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

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

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

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

Jones, M. B. (1997). The Microsoft Interactive TV System: An Experience Report. Technical Report MSR-TR-97-18, Microsoft Research. ftp://ftp.research.microsoft.com/pub/tr/tr-97-18/tr-97-18.html.

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

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

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

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

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

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

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

McKusick, M. K., W. N. Joy, S. Leffler, and R. S. Fabry (1984, August). A Fast File System for UNIX. *ACM Transactions on Computer Systems 2*(3), 181–197.

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

McVoy, L. and S. R. Kleiman (1991). Extent-like Performance from a Unix File System. In *USENIX Winter Technical Conference*, Dallas, TX, pp. 33–43.

Mills, D. L. (1991, October). Internet Time Synchronization: The Network Time Protocol. *IEEE Transactions on Communications 39*(10), 1482–1493.

MPEG Software Simulation Group (1996). *MPEG-2 Encoder/Decoder, Version 1.2*. MPEG Software Simulation Group.

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

Muntz, R., J. R. Santos, and S. Berson (1998, December). A Parallel Disk Storage System for Real-Time Multimedia Applications. *International Journal of Intelligent Systems 13*(12), 1137–1174.

Neufeld, G., D. Makaroff, and N. Hutchinson (1996, January). Design of a Variable Bit Rate Continuous Media File Server for an ATM Network. In *IS&T/SPIE Multimedia Computing and Networking Conf.*, San Jose, CA, pp. 370–380.

Ng, S. W. (1998, May). Advances in Disk Technology: Performance Issues. *Computer 31*(15), 75–81.

Ozden, B., R. Rastogi, P. Shenoy, and A. Silberschatz (1996, June). Fault-tolerant Architectures for Continuous Media Servers. Montreal, Canada, pp. 79–90.

Ozden, B., R. Rastogi, and A. Silberschatz (1996, June). Disk Striping in Video Server Environments. In *IEEE Multimedia Computing and Systems*, Hiroshima, Japan, pp. 580–589.

Paek, S., P. Bockeck, and S.-F. Chang (1995, April). Scalable MPEG2 Video servers with Heterogeneous QoS on Parallel Disk Arrays. In *Intl. Work. Network and Operating Systems Support for Audio and Video*, Durham, NH, pp. 342–353.

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

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

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

PCI Special Interest Group (1995, June). *PCI Local Bus Specification* Revision 2.1. Portland, OR: PCI Special Interest Group.

Reddy, A. L. N. and R. Wijayaratne (1999, January). Techniques for improving the through-

put of VBR streams. In *ACM/SPIE Multimedia Computing and Networking Conf.*, San Jose, CA, pp. 216–227.

Reibman, A. R. and A. W. Berger (1995, June). Traffic Descriptors for VBR Video Teleconferencing Over ATM Networks. *IEEE/ACM Transactions on Networking 3*(3).

Rexford, J., S. Sen, J. Dey, W. Feng, J. Kurose, J. Stankovic, and D. Towsley (2000, March). Online Smoothing of Live, Variable-Bit-Rate Video. *IEEE Trans. on Multimedia 2*(1), 37–48.

Ruemmler, C. and J. Wilkes (1994, March). An Introduction to Disk Drive Modeling. *Computer 27*(3), 17–28.

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

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

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

Santos, J. R., R. R. Muntz, and B. Ribeiro-Neto (2000, June). Comparing Random Data Allocation and Data Striping in Multimedia Servers. In *ACM SIGMETRICS*, Santa Clara, CA, pp. 44–55.

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

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

Shenoy, P. J., P. Goyal, S. S. Rao, and H. M. Vin (1998, January). Symphony: An Integrated

Multimedia File System. In *ACM/SPIE Multimedia Computing and Networking Conf.*, San Jose, CA, pp. 124–138.

Shenoy, P. J., P. Goyal, and H. M. Vin (1995, December). Issues in Multimedia Server Design. *ACM Computing Surveys 27*(4), 636–639.

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

Tan, W. S., N. Duong, and J. Princen (1991, July). A comparison study of variable bit rate versus fixed bit rate video transmission. In *Australian Broadband Switching and Services Symposium*, pp. 134–141.

Triantafillou, P. and S. Harizopoulos (1999, June). Prefetching into Smart-Disk Caches for High Performance Media Server. In *IEEE Multimedia Computing and Systems*, Florence, Italy, pp. 800–805.

Vin, H. M., S. S. Rao, and P. Goyal (1995, May). Optimizing the Placement of Multimedia Objects on Disk Arrays. In *IEEE Intl Conf Multimedia Computing and Systems*, Washington, DC, pp. 158–165.

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

Wolf, J. L., P. S. Yu, and H. Shachnai (1995, May). DASD Dancing: A Disk Load Balancing Optimization Scheme for Video-on-Demand Computer Systems. In *ACM SIGMETRICS*, Ottawa, Canada, pp. 157–166.

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

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

Zimmermann, R. and S. Ghandeharizadeh (1997, November). Continuous Display Using

Heterogeneous Disk-Subsystems. In *ACM Multimedia*, Seattle, WA, pp. 227–328.