The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

ON THE DESIGN OF SCALABLE AND HIGH PERFORMANCE MULTIMEDIA SERVERS

A Thesis in

Computer Science and Engineering

by

Nabil J. Sarhan

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2003

The thesis of Nabil J. Sarhan has been reviewed and approved* by the following:

Chita R. Das
Professor of Computer Science and Engineering
Thesis Adviser, Chair of Committee

Mary Jane Irwin
Distinguished Professor of Computer Science and Engineering

John Yen
University Professor of Information Sciences and Technology
Professor of Computer Science and Engineering

Vijaykrishnan Narayanan
Assistant Professor of Computer Science and Engineering

Guohong Cao
Assistant Professor of Computer Science and Engineering

John F. Doherty
Associate Professor of Electrical Engineering

Raj Acharya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

Recent advances in communication and storage technologies have spurred a strong interest in *Multimedia-on-Demand* (MOD) services. These services eliminate the shortcomings of their broadcast-based counterparts by providing customers with convenience, choice, and control.

The purpose of this thesis is to address the two principal challenges facing multimedia servers: supporting large numbers of concurrent customers and scaling easily and cost-effectively in order to cope with the increasing demands. The number of customers that can be serviced concurrently is highly constrained by the stringent requirements of the real-time playback and the high transfer rates. These requirements lead to rapid consumptions of server resources, especially the storage subsystem bandwidth and the network bandwidth. The scalability challenge arises primarily because current multimedia servers are based on the conventional fileserver architecture, where all the data flowing from the disks to the clients pass through the server. The required store-and-forward copying imposes unnecessary burdens on these servers, severely limits their scalability, and increases their costs.

First, this thesis proposes an *Adaptive Block Rearrangement* policy that enhances the performance of multimedia storage subsystems by dynamically rearranging the blocks on each disk according to their access frequencies. The effectiveness of the adaptive policy and the impacts of different data placement layouts on performance are demonstrated through extensive simulation.

Second, this thesis proposes a new class of scheduling policies, called *Next Schedule Time First* (NSTF), which provides hard time of service guarantees and accurate schedule times. Providing such guarantees not only enhances customer-perceived quality of service (QoS) but also increases server throughput by motivating customers to wait. The thesis presents alternative implementations of NSTF and demonstrates through extensive simulation that NSTF delivers outstanding performance benefits.

Third, this thesis develops a validated analytical model for caching in multimedia servers. The model is then used to examine the impacts of various system and workload parameters and to experiment with new design alternatives.

Fourth, this thesis proposes using the emerging *Network-Attached Disk* (NAD) architecture for designing large-scale, cost-effective, and scalable multimedia servers. The NAD architecture removes all store-and-forward copying, which severely limits the scalability and increases the costs of conventional multimedia servers.

Finally, this thesis develops an *Integrated Resource Sharing* policy, which improves the performance of NAD-based multimedia servers through caching and intelligent scheduling. This policy combines two schemes that are also proposed here: *Distributed Interval Caching* (DIC) and *Multi-Objective Scheduling* (MOS). The DIC scheme exploits the internal caches of NADs in caching intervals between successive streams. The MOS scheme schedules the waiting requests intelligently based on four performance criteria. The integrated policy increases the number of serviced customers while reducing their waiting times for service, uses server and network resources efficiently, does not expect much resource from customers, achieves greater performance benefits for larger servers,

and provides high configurability. The effectiveness of the integrated policy and the interactions among various system and workload parameters are studied through extensive simulation. The performance limit of DIC is also evaluated using analytical modeling.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I am grateful and indebted to my thesis advisor, Dr. Chita R. Das, for the guidance and encouragement he has shown me during my time here at Penn State.

I would also like to thank all committee members for their efforts and suggestions, which improved the quality of this thesis.

Finally, I am grateful and indebted to my parents, brothers, and sisters, who always surrounded me with love and care, supported me in every way possible, and encouraged me to pursue my graduate studies.

Chapter 1

Introduction

Multimedia networking applications have enjoyed a rapidly widening popularity especially since the emergence of multimedia data as an integral and an essential part of the World Wide Web (WWW). These applications include *Multimedia-on-Demand* (MOD), *Live Streaming*, and *Interactive Real-Time* (such as Internet telephony and video conferencing).

The interest in MOD applications in particular has grown dramatically. By contrast with broadcast-based services like cable TV, MOD services enable customers to access the multimedia contents they want at the times of their choosing and allow them to apply interactive operations. MOD services have led to fierce competitions among motion picture studios and cable and news companies, which earnestly seek to better entertain the needs of their customers, increase their market shares, and create markets for their huge on-the-shelf multimedia assets. The companies that survive in the future are those that best manage their multimedia assets and deliver rich multimedia contents quickly, easily, and efficiently [32]. In addition to the wide use of MOD services for entertainment, they have been of great importance in education and distance learning in particular.

Many driving forces for this growing interest in MOD services exist besides the ever-increasing importance of customized services. These forces include the exponential

expansion of the Internet, the availability of high-speed Internet access at home and office, and the increasing suitability of the Internet for transporting multimedia traffic through the incremental deployment of multicasting [46] and the development of frameworks for providing quality of service (QoS) guarantees by the Internet [12, 10, 7, 24]. The multicast facility increases the number of serviced customers by efficiently using server bandwidth, while the anticipated provision of QoS guarantees will make MOD services over the Internet more interesting and more enjoyable.

The design of MOD servers faces significant challenges to support large numbers of concurrent customers and to scale easily and cost-effectively in order to cope with the increasing demands. The number of customers that can be serviced concurrently depends on the performance of the underlying storage subsystem and the available network bandwidth to the "world". Unfortunately, this number is highly constrained by the stringent requirements of multimedia data. Specifically, multimedia data must be presented continuously in time so as to convey meaningful information, and they require high transfer rates. Therefore, a wide spectrum of techniques has been developed to enhance the performance of MOD servers. These techniques can be classified into three main classes. The first class enhances request *admission control* [33, 6, 70, 104, 105, 60, 106, 108, 107, 69]. The second class improves the performance of the storage subsystem through *disk striping* [98, 19], *disk head scheduling* [79, 80, 39, 108, 74, 66], *data replication* [34, 19, 21], *data block allocation* [78, 34], and *read-ahead buffering* [27]. The third class enhances *resource sharing* by servicing multiple requests from a common set of resources.

Resource sharing strategies for multimedia servers include *Batching* [26, 28, 103, 97, 2], *Patching* [57, 96, 14, 15], *Piggybacking* [49, 48, 1], *Broadcasting* [59, 56, 75, 76,

55, 16], and *Interval Caching* [25, 27, 29, 30, 101, 65, 5]. Batching off-loads the storage subsystem and uses efficiently server bandwidth and network resources by accumulating the requests to the same multimedia contents and servicing them together by utilizing the multicast facility. Patching is similar to batching, but it expands the multicast tree dynamically to include new requests, thereby reducing the request waiting time and improving resource sharing, but it requires additional bandwidth and buffer space at the client. Piggybacking offers similar advantages to patching, but it adjusts the playback rate so that the request catches up with a preceding stream, resulting in a lower-quality presentation of the initial part of the requested multimedia content. Broadcasting techniques divide each popular multimedia content into multiple segments and broadcast each segment periodically on dedicated server channels. With these techniques, the improved resource sharing and the fixed waiting times for the playbacks of the popular multimedia contents come at the expense of requiring very high additional bandwidth and buffer space at the client. Interval caching caches intervals between successive streams in the server. It does not sacrifice the playback QoS, does not lengthen the waiting time, and does not expect much resource from the client. It can also be applied with other techniques for better resource sharing, and it has become more cost-effective with the falling prices of semiconductor memories.

The exploited degrees of resource sharing depend greatly on how MOD servers schedule the waiting requests. By scheduling these requests intelligently, a server can support more concurrent requests, can decrease their waiting times for service, and/or can meet some other performance objectives. Servers that employ batching rely almost entirely on scheduling to boost up their performance. Scheduling is also important

in MOD servers that use other resource sharing strategies. A MOD server maintains a waiting queue for every multimedia content to facilitate scheduling and services all requests in a selected queue together using only one stream. Scheduling policies for MOD servers include *First Come First Serve* (FCFS), *Maximum Queue Length* (MQL), and *Maximum Factored Queue Length* (MFQL). FCFS selects the queue with the oldest request, whereas MQL selects the longest queue, and MFQL selects the queue with the *largest factored length*. The factored length of a queue is its length divided by the square root of the relative access frequency of its corresponding multimedia content.

The scalability challenge arises primarily because current MOD servers are based on the conventional fileserver architecture (also known as the *Server-Attached Disk Architecture*). In this architecture, all the data flowing from the disks to the clients pass through the server. The required *store-and-forward* copying imposes unnecessary burdens on these servers, severely limits their scalability, and increases their costs.

The purpose of this thesis is to address both the throughput and the scalability challenges. The primary application of interest is *Video-on-Demand* (VOD). The following are the main contributions of this work.

- This thesis proposes an *Adaptive Block Rearrangement* policy that improves the performance of multimedia storage subsystems by exploiting the high locality of reference in the access patterns of multimedia data. This policy monitors the accesses to the data blocks and keeps the blocks with comparable access frequencies close to each other on the disks. Two rearrangement algorithms are analyzed: *Centered-Layout* and *Sequential-Layout*. The centered-layout algorithm rearranges the data blocks on each disk so that the blocks of the more popular videos are

located nearer to the center of the disk. The sequential-layout algorithm, however, keeps the blocks of the more popular videos nearer to the edge of the disk. The effectiveness of the adaptive policy and the impacts of the centered, sequential, and random layouts on the performance of the storage subsystem are demonstrated by extensive simulation.

- This thesis proposes a new class of scheduling policies, called *Next Schedule Time First* (NSTF), which provides customers with hard time of service guarantees. By providing these guarantees, a server not only can enhance customer-perceived QoS, but also can motivate customers to wait, thereby increasing throughput. NSTF removes the shortcomings of FCFS. FCFS was believed to provide hard time of service guarantees [103]. By contrast, the thesis demonstrates that FCFS may violate its time of service guarantees. It also shows that FCFS is incapable of producing accurate times of service. Specifically, the average deviation of the time of service guarantees from the actual times of service can be up to 4.5 minutes. With NSTF, however, the average deviation can be within just 0.2 second. The thesis presents alternative implementations of NSTF and shows through extensive simulation that NSTF delivers outstanding performance benefits.

- This thesis develops a validated analytical model for interval caching. It then uses that model to examine the impacts of various system and workload parameters and to experiment with new design alternatives. In particular, it evaluates the effectiveness of a hybrid caching scheme that caches a small set of the most popular videos in their entireties and employs interval caching for the rest.

- The thesis proposes using the emerging *Network-Attached Disk* (NAD) architecture [42, 43, 44, 45] for designing large-scale, cost-effective, and scalable multimedia servers.

- The thesis develops an *Integrated Resource Sharing* policy that enhances the performance of NAD-based multimedia servers through caching and intelligent scheduling. Namely, two schemes are proposed: *Distributed Interval Caching* (DIC) and *Multi-Objective Scheduling* (MOS). The DIC scheme extends and adapts interval caching for the NAD architecture. The MOS scheme schedules the waiting requests for service intelligently based on four performance criteria. These two schemes are then integrated for better resource sharing. The integrated policy offers five main advantages. First, it increases the number of serviced customers while reducing their waiting times for service. Second, it uses server and network resources efficiently. Third, it does not expect much bandwidth and buffer space at the client. Fourth, it provides high configurability. By adjusting appropriate parameters, the administrator can make any necessary tradeoff, including the number of concurrent customers, waiting times, fairness, and implementation complexity. Certain parameters can also be tuned adaptively in order to fit the current server workload, which may change with time (especially from daytime to nighttime and from business days to weekends). Fifth, it achieves greater performance benefits for larger servers. The effectiveness of the integrated policy and the interactions among various system and workload parameters are studied through extensive simulation. The performance limit of DIC is also found through analytical modeling.

The rest of this thesis is organized as follows. Chapter 2 discusses background information and related work. Chapter 3 presents the adaptive block rearrangement policy. Chapter 4 addresses the problem of providing time of service guarantees in VOD servers. Chapter 5 presents an analytical model for interval caching and investigates new caching alternatives. Chapter 6 discusses the design of NAD-based multimedia servers. Finally, the last chapter draws conclusions and outlines future work.

Chapter 2

Related Work

This chapter discusses the main characteristics and access patterns of multimedia data, the different classes of multimedia networking applications, and the various design aspects of multimedia servers.

2.1 Multimedia Data Characteristics

Multimedia data differ from traditional textual and numeric data in two main ways. First, multimedia data require high storage capacities and high transfer rates. Second, they consist of sequences of media quanta, which convey meaningful information only when presented continuously in time. These stringent requirements impose heavy burdens on server resources, especially the disk I/O bandwidth and the network bandwidth.

2.2 Multimedia Data Access Patterns

The accesses to videos are highly localized, with only a small number of videos receiving most of the hits [18]. Specifically, the accesses follow *Zipf-Like* distribution [18, 26, 20]. With this distribution, the probability of choosing the n^{th} most popular video is $C/n^{1-\theta}$, with a parameter θ and a normalized constant C. The parameter θ controls the skewness of video access. The skewness reaches its peak when θ is zero, and

the access becomes uniformly distributed when θ is one. The distribution is called the *Zipf's* distribution [118] when θ is zero.

2.3 Multimedia Networking Applications

Multimedia networking applications can be classified into three main classes: *Multimedia-on-Demand* (MOD), *Live Streaming*, and *Interactive Real-Time*. MOD applications deliver stored multimedia contents to customers and allow them to apply interactive operations such as pause, resume, fast forward, and fast rewind. These operations are also called *VCR-like* operations. By contrast, live streaming applications offer live multimedia contents to customers. Naturally, customers cannot fast forward through the multimedia data with these applications. Other interactive operations, however, such as pause and rewind may be applied with local storage of appropriate portions of the multimedia data. On the other hand, interactive real-time applications allow customers to communicate with each other in real-time. The most common applications in this class are Internet telephony and video conferencing.

MOD in general and *Video-on-Demand* (VOD) in particular is the application class of interest in this thesis. These applications are characterized by the following features.

- The multimedia contents (video and/or audio) are pre-recorded and stored in the server.
- The multimedia contents should be played out continuously according to the original timings of the recordings.

- Clients can apply VCR-like operations.
- The playout of each multimedia content is pipelined with its transmission.

The pipelining of the playout of multimedia data with the transmission is called *streaming*. With this technique, a client starts the playout shortly after beginning to receive the data instead of waiting until the entire file is downloaded. In streaming applications, the hyperlink of the multimedia content points to a *meta file* rather than the actual multimedia file. The meta file is a description file that contains the URL of the media file and its type of encoding. When the web browser opens the link, it retrieves the meta file from the web server, and then opens the meta file and launches the appropriate *helper application*. Subsequently, the helper application streams the multimedia content from the corresponding *streaming server* to the client. The helper application is usually responsible for decompression, jitter removal, error correction, and graphical user interfacing.

Three application-layer protocols are commonly used for streaming multimedia data over the Internet: *Real Time Protocol* (RTP) [83], *Real Time Streaming Protocol* (RTSP) [84], and *Real Time Control Protocol* (RTCP) [83]. RTP provides end-to-end delivery services, including payload type identification, sequence numbering, timestamping, delivery monitoring, and source identification. It also supports the delivery to multiple users using multicasting. RTSP and RTCP are control protocols that work along with RTP. RTSP supports VCR-like operations, whereas RTCP provides some services related to the congestion and the flow control functions in TCP. The services provided by RTCP

are QoS monitoring, congestion control, source identification, inter-media synchronization, and control information scaling.

The delivery of multimedia data over the Internet faces significant challenges to support the real-time playback and the high bandwidth requirements. The Internet was incepted as a data network, providing only a best-effort service. It therefore makes no guarantees whatsoever on the expected packet delay or delay jitter. The best-effort service works well for traditional numeric and textual data, but unfortunately, it is not well-suited for multimedia workload.

The shortcomings of the Internet's best-effort service have been mitigated by using the *User Datagram Protocol* (UDP) instead of the *Transmission Control Protocol* (TCP), adding more bandwidth to the links, and applying network caching, client buffering, adaptive encoding, and multicasting. UDP is preferable over TCP because TCP retransmits lost packets to live up to its guarantees of reliable communication. Such retransmissions amplify the already high delay in the delivery of multimedia packets. Fortunately, these retransmissions can be avoided for multimedia data, which are naturally loss tolerant: losses cause glitches, which can be concealed partially or fully. Using UDP instead of TCP eliminates the added delays and evades the slow start phase of TCP. Adding more bandwidth copes with the high data transfer rates required by multimedia data. Network caching reduces the server load and brings data closer to the client. Buffering the multimedia content at the client for a little while before its playback helps to smooth out the delay jitter. With adaptive encoding, the sender adapts the encoding rate to the client's consumption rate and the current network condition. Finally, the multicast facility uses server bandwidth and network resources efficiently

when the same multimedia data are transported to multiple destinations. It is already employed or can be easily employed in most enterprise and local area networks (LANs). Besides, it is supported by IPv4 and has incrementally been deployed over the Internet because of the development and standardization of pertinent protocols and the willingness of Internet Service Providers (ISPs) to provide a scalable architecture. In addition to application-level multicast [58, 22, 17], a ubiquitous wide-scale deployment of native (network-level) multicast across the Internet is becoming a reality [46]. For example, Sprint has already deployed native multicast across its Internet backbone [99].

The need for the Internet, however, to provide services beyond the best-effort service has grown spectacularly with the emergence of multimedia data as an integrated and essential part of the World Wide Web and by the rapid transformation of the Internet into a commercial infrastructure.

Many frameworks, therefore, have been developed for providing QoS. These frameworks include the *Integrated Services* (Intserv) model [23, 12, 13, 110], the *Differentiated Services* (Diffserv) model [10, 9], *Multiprotocol Label Switching Architecture* (MPLS) [82, 7], *Traffic Engineering* [7], and *Constraint Based Routing* [24]. The Intserv model requires fundamental changes to the Internet so that applications can reserve bandwidth in each link between the sender and the receiver. This model employs the *Resource ReSerVation Protocol* (RSVP) [13, 51, 68, 8] for setting up paths and reserving resources along them. The Diffserv model, however, requires only minor changes to the Internet. It introduces few classes and gives packets differentiated services based on their assigned classes. With MPLS, packets are assigned labels at the ingress of a MPLS-capable domain and then forwarded along paths determined completely by these labels. Traffic

Engineering determines how traffic flows through the network so that congestion caused by uneven network utilization is avoided. The main objectives of Constraint Based Routing, which automates the Traffic Engineering process, are to select routes that can meet certain QoS requirements (bandwidth and/or delay) and to increase network utilization. Additional information on Internet QoS can be found in [117, 67].

These QoS frameworks, which are expected to be deployed in the near future, will make multimedia applications on the Internet more enjoyable and more interesting.

2.4 Design of Multimedia Servers

The major design objectives of multimedia servers are increasing the number of customers that can be serviced concurrently, reducing their waiting times for service, and providing enhanced playback QoS. The design of these servers involves many different aspects, including request admission control, storage subsystem management, resource sharing, and scheduling.

2.4.1 Admission Control

The admission control unit of a multimedia server determines whether a new client can be admitted without violating the real-time performance requirements of any admitted client. Admission control policies can be classified into three main categories: *Deterministic* [33, 6, 70, 104, 105], *Predictive* [60, 106, 108], and *Statistical* [33, 107, 69]. Deterministic admission control policies admit new requests only if the real-time requirements of all existing requests are guaranteed to be met. Predictive admission control policies monitor server utilization and admit new requests if the recent history

suggests that the server is likely to meet the real-time requirements. Statistical admission control policies base their decisions on the server usage statistics. The deterministic policies under-utilize server resources and highly limit throughput by considering the worst-case scenarios in admitting new requests. Conversely, the predictive and statistical policies provide soft QoS guarantees, thereby increasing server throughput. Compared with the predictive policies, the statistical policies have the advantage of not incurring high overheads in the process of collecting the real-time workload history.

2.4.2 Storage Subsystem Management

Multimedia servers typically employ disk arrays because of the large storage and high bandwidth requirements of multimedia data. These servers interleave (stripe) multimedia streams across successive disks in a disk array and service multiple clients by proceedings into periodic rounds and by retrieving a fixed multimedia unit for each client from one (or sometimes more) of these disks during each round.

Numerous techniques were developed to enhance the performance of multimedia storage subsystems. These techniques include *Data Replication* [34, 19, 21], *Data Block Allocation* [78, 34], *Disk Head Scheduling* [79, 80, 39, 108, 74, 66], *Disk Striping* [98, 19, 116], and *Read-Ahead Buffering* [27].

A striping policy determines two parameters: the *stripe unit size* and the *degree of striping*. The stripe unit size denotes the maximum amount of logically continuous data stored on a single disk, whereas the degree of striping denotes the number of disks across which a particular multimedia stream is striped.

Read-ahead buffering is usually used to absorb the variability in the disk access time. This technique reads and buffers blocks for each stream before they are needed. Without buffering, the average disk utilization should be bounded in order to achieve an acceptable level of *jitter* (number of blocks whose real-time constraints are violated). Thus, read-ahead buffering improves the ability of the server to meet the real-time requirements and enhances its throughput with only a small additional cost [27].

Additional information on the design of multimedia storage subsystems can be found in [40].

2.4.3 Resource Sharing

The performance of multimedia servers in general and VOD servers in particular can be significantly improved by servicing multiple requests from a common set of resources. The main classes of resource sharing techniques for VOD servers include *Batching* [26, 28, 103, 97, 2], *Patching* [57, 96, 14, 15], *Piggybacking* [49, 48, 1], *Broadcasting* [59, 56, 75, 76, 55, 16], and *Interval Caching* [25, 27, 29, 30, 101, 65, 5]. These techniques benefit greatly from the high locality of reference in video access patterns.

With batching, the requests to the same videos are accumulated and serviced together by utilizing the multicast facility. Batching, therefore, off-loads the underlying storage subsystem and uses efficiently server bandwidth and network resources.

Patching is similar to batching, but it expands the multicast tree dynamically to include new requests. When a request arrives, the server initiates a patching stream to service the trail of the requested video. Meanwhile, the server transmits the multicast

data, and the client caches them. Patching reduces the request waiting time and improves resource sharing but requires additional bandwidth and buffer space at the client.

Like patching, piggybacking services a request almost immediately, but unlike patching, it adjusts the playback rate so that the request catches up with a preceding stream. Obviously, reducing the waiting time and increasing resource sharing by piggybacking comes at the expense of a lower-quality presentation of the initial portion of the requested video.

Broadcasting techniques divide each video into multiple segments and broadcast each segment periodically on dedicated server channels. Thus, the client has to wait until the next broadcast of the first segment. Because multiple segments of each popular video are transmitted concurrently, these techniques require very high additional bandwidth and buffer space at the client.

Interval caching exploits the locality of reference by caching intervals between successive streams in the main memory of the server. It attempts to pair each playback request with an immediately preceding request for the same video that is currently being serviced (either from the disks or the cache). The two streams are called the *following* stream and the *preceding* stream, respectively. When the server accepts a request for service from the cache, it starts to cache the data of the preceding stream while retrieving and transferring them to the client. The required cache space for servicing the following stream depends on the time interval between the two streams and the compression method used. Interval caching uses the available cache (memory) space efficiently by ensuring that only the shortest intervals are cached at any time. This is done by victimizing longer intervals for caching shorter ones. This technique shortens the

request waiting time and enhances server throughput without increasing the bandwidth or the buffer space requirement at the client. It can also be applied in conjunction with other resource sharing techniques to improve performance further. Furthermore, it has become cost-effective with the falling prices of semiconductor memories.

The concept of interval caching was generalized in a policy, called *Generalized Interval Caching* (GIC) [30]. GIC extends interval caching in order to handle workloads of short and long videos. Namely, it does not require paired streams to be concurrent because short videos are unlikely to have concurrent accesses. When two paired streams are not concurrent, it caches the entire corresponding video.

The multicast facility is essential to maximize the benefits of resource sharing. In [102], however, a resource sharing technique that works in the absence of this facility was proposed. That technique, called *Layered Range Multicast* (LRM), serves as an intermediate solution until the multicast facility is fully deployed. It reduces the required server bandwidth, accommodates client heterogeneity, and provides a unicast-based multicast implementation by employing overlay nodes (i.e., application-level routers). Unfortunately, the price for these advantages is requiring the placement of LRM-enabled across the Internet.

2.4.4 Request Scheduling

Resource sharing can be significantly improved through intelligent scheduling of the waiting requests. A VOD server maintains a waiting queue for every video, routes incoming requests to their corresponding queues, selects for service an appropriate queue whenever it has an available *channel*, and services all requests in the selected queue

using only one channel. A channel is a set of resources (disk I/O bandwidth, network bandwidth, etc) needed to deliver a multimedia stream. The number of channels in a server is referred to as *server capacity*.

2.4.4.1 Scheduling Objectives

All scheduling policies are guided by one or more of the following objectives.

1. Minimize the overall customer reneging (defection or turnaway) probability.
2. Minimize the average request waiting time.
3. Prevent starvation.
4. Provide time of service guarantees.
5. Minimize unfairness.
6. Minimize implementation complexity.

The reneging probability is the probability that a new customer leaves the system without being serviced because of a waiting time exceeding the user's tolerance. It is the most important metric because it translates to the number of customers that can be serviced concurrently and to server throughput. The throughput, X, for a given request arrival rate, λ, and reneging probability, P_r, is given by

$$X = (1 - P_r) \times \lambda.$$

The second objective comes next in importance. The second, third, and fourth objectives are indicators of customer-perceived quality of service (QoS). By providing time of service

guarantees, a VOD server can also influence customers to wait, thereby increasing server throughput. It is also usually desirable that VOD servers treat equally the requests for all videos. Unfairness measures the bias of a policy against cold (i.e., unpopular) videos and can be found by the following equation:

$$unfairness = \sqrt{\sum_{i=1}^{N_v} (r_i - \bar{r})^2 / (N_v - 1)},$$

where r_i is the reneging probability for the waiting queue i, $\bar{r}$ is the mean reneging probability across all waiting queues, and N_v is the number of waiting queues (and number of videos as well). Finally, minimizing the implementation complexity is a secondary issue in VOD servers for two reasons. First, the number of objects (videos) and the number of concurrent customers are relatively small compared with those in other servers such as web servers. Second, the CPU and the main memory are not usually performance bottlenecks in VOD servers.

2.4.4.2 Scheduling Policies

Let us now discuss the common scheduling policies for VOD servers.

- *First Come First Serve* (FCFS) [26] – This policy selects the queue with the oldest request.

- *FCFS-sum* [31, 113] – This policy selects the queue with the largest sum of request waiting times. It is referred to as *Longest Wait First* (LWF) in the context of videotex systems [31, 113] and web servers with data broadcasting [3].

- *FCFS-n* [26] – This policy broadcasts periodically the n most common videos on dedicated channels and schedules the requests for the other videos on a FCFS basis. When no request is waiting for the playback of any one of the n most common videos, it uses the corresponding dedicated channel for the playback of one of the other videos.

- *Maximum Queue Length* (MQL) [26] – This policy selects the queue with the largest number of waiting requests. Note that the length of a queue is a discrete number with a typically small value. Thus, there is a significant probability that multiple queues have the same length. The definition of MQL, however, does not specify the selection criterion among the longest queues. In [87], two alternative implementations of MQL were considered: *MQL-u* and *MQL-f*. MQL-u selects the longest queue, and whenever there is more than one eligible queue, it selects the queue of the most popular video among them. MQL-f is similar to MQL-u, but it selects the queue of the least popular video among the eligible queues. MQL-u can be aggressively biased against cold videos, whereas MQL-f can be much fairer. MQL-u and MQL-f vary considerably in terms of the achieved throughput and average request waiting time. In the context of videotex systems, a policy called *Most Requests First Lowest* (MRFL) was proposed in [31, 113]. This policy is essentially the same as MQL-f.

- *Maximum Factored Queue Length* (MFQL) [2] – This policy attempts to minimize the mean request waiting time by selecting the queue with the *largest factored queue length*. The factored length of a queue is defined as its length divided by

the square root of the relative access frequency of its corresponding video. MFQL reduces waiting times optimally only if the server is fully loaded and customers always wait until they receive service (i.e. no defections).

- *Quantized First Come First Server* (QFCFS) [87] – This policy examines both waiting times and queue lengths, thereby serving as a good compromise between FCFS and MQL/MFQL. First, it translates waiting times into discrete levels. The interval between consecutive levels is called the *quantization interval* (Q). Then, it selects for service the queue with the largest *quantized* waiting time. Note that there may be multiple eligible queues. The selection criterion of one queue among these queues leads to two variants of QFCFS: *QFCFS-m* and *QFCFS-f*. *QFCFS-m* chooses the longest queue, whereas *QFCFS-f* chooses the queue with the largest factored length. QFCFS is a generalized policy because if Q approaches zero, then QFCFS becomes FCFS, and if Q approaches a sufficiently large number, it becomes MQL or MFQL, depending on the specific implementation.

- *Group-Guaranteed Server Capacity* (GGSC) [103] – This policy preassigns server channel capacity to groups of requests in order to optimize the mean request waiting time. It groups objects that have nearly equal expected batch sizes and schedules requests in each group on a FCFS basis on the collective channels assigned to each group.

- *Maximum Batching* Schemes [97] – These schemes aggressively pursue batching by deliberately delaying requests. With these schemes, a queue is eligible for selection

only if it has at least one request with a waiting time greater than or equal to a pre-specified batching threshold. The selection criterion of a queue from the eligible queues leads to two alternative policies: *Maximum Batching Maximum Queue Length* (BMQ) and *Maximum Batching Minimum Loss* (BML). BMQ selects the longest queue, whereas BML selects the queue that will incur – if not selected – the largest expected loss of requests till the next stream completion time.

- *Minimum Idling* Schemes [97] – These schemes pursue batching without minimum wait requirements. They partition the waiting queues into two sets: a hot set ($\mathcal{H}$) and a cold set ($\mathcal{C}$). A queue belongs to $\mathcal{H}$ if it meets any of the following three conditions.

  - It corresponds to a popular video. Because videos are numbered in decreasing order of their popularity, a video is classified as popular if its number is less than a fixed number, τ.
  - It has more than one request.
  - It has exactly one request and that request has been waiting longer than a pre-specified threshold, T.

  A queue belongs to $\mathcal{C}$ if it does not meet any of these conditions. These schemes give a higher priority to the queues in $\mathcal{H}$ and schedule the queues in $\mathcal{C}$ on a FCFS basis when $\mathcal{H}$ is empty. They are not very sensitive to the value of τ because they can also recognize popular videos dynamically by examining the numbers of requests in the queues. The selection criterion of a queue in $\mathcal{H}$ leads to two alternative policies: *Minimum Idling Maximum Queue Length* (IMQ) and *Minimum Idling Minimum*

Loss (IML). IMQ selects the longest queue, whereas IML selects the queue that will otherwise incur the largest expected loss of requests till the next stream completion time. In [87], a policy called *Enhanced IML* (IML^+) was proposed. This policy improves IML by capturing the situations in which multiple queues become eligible candidates for selection.

2.4.4.3 Comparisons among Various Policies

Choosing an appropriate scheduling policy is very complicated by the presence of several design tradeoffs and the dependence of performance on customer waiting tolerance and server load. FCFS is the fairest and the easiest to implement, but it leads to relatively long waiting times. MQL and MFQL reduce the average request waiting time but tend to be biased against cold videos, which have relatively few waiting requests. Unlike MQL, MFQL requires periodic computations of access frequencies. MQL-f is not only fairer than MQL-u, but it also yields higher throughput, especially for high server capacities. Thus, MQL-f is used as the implementation of choice of MQL in this thesis. FCFS can prevent starvation, whereas MQL and MFQL cannot. FCFS-sum achieves generally better overall performance than FCFS at the expense of much higher implementation complexity [87]. FCFS-n performs either as well as or worse than FCFS [103]. GGSC does not perform as well as FCFS in high-end servers [103]. Maximum batching and minimum idling schemes have relatively high implementation complexities and may cause starvation, but they can exploit minimum request waiting times. Minimum idling schemes require fewer channels to guarantee a given reneging

percent than maximum batching schemes [97]. A detailed analysis of scheduling policies can be found in [87].

Chapter 3

Adaptive Block Rearrangement

This chapter presents an *Adaptive Block Rearrangement* policy for VOD storage subsystems and demonstrates its performance benefits through extensive simulation.

3.1 Introduction

Disks are identified as a major performance bottleneck in VOD servers because of their relatively long access times and the relatively less effective caching in these servers. Indeed, striping data across multiple disks improves the I/O bandwidth, but the service time of each disk remains a limiting factor in the overall performance. The disk service time consists of a head-positioning time and a data transfer time. The seek time and the rotational time contribute to the head-positioning time, and both depend primarily on the disk technology. The seek time, however, depends not only on the disk technology, but also on the request pattern, the disk scheduling algorithm, and the block arrangement.

This study proposes an *Adaptive Block Rearrangement* policy that reduces the disk seek times by exploiting the high locality of reference in video access patterns. This policy monitors the accesses to videos and rearranges the blocks on each disk according to their access frequencies, whenever considerable performance benefits are expected,

and the server is not too busy. The policy ensures a faster disk access by keeping the blocks of the videos with comparable access frequencies close to each other.

This study presents two algorithms for the adaptive rearrangement: *Centered-Layout* and *Sequential-Layout*. The centered-layout algorithm rearranges the video blocks based on a variation of the organ pipe heuristic [52]. The organ pipe heuristic places the most frequently accessed data in the center of the disk. The next most frequently accessed data is placed to either side of the center, and the process continues until the least frequently accessed data has been placed at or near the edges of the disk. By contrast, the sequential-layout algorithm rearranges the video blocks so that the blocks of the more popular videos are located closer to the edge of the disk.

This study compares, through extensive simulation, the centered layout, the sequential layout, and the possible layouts in the absence of any rearrangement. The simulated system is a disk array of *Quantum Atlas10K*. Random layouts with two different degrees of randomness are used to characterize placements in systems where no rearrangement is maintained. Three disk performance metrics are analyzed: seek distance, seek time, and access time. In addition, the impacts on these metrics of the disk array size, the stripe unit size, and the video durations (lengths) are examined.

The rest of the chapter is organized as follows. Section 3.2 discusses the related research work. Section 3.3 presents the adaptive block rearrangement policy. Section 3.4 discusses the performance evaluation methodology. Section 3.5 presents and analyzes the main simulation results. Finally, the last section draws conclusions.

3.2 Related Work

The organ pipe heuristic was shown to place data optimally if data references are derived from an independent random process with a known fixed distribution [50, 111]. Variations of this heuristic were also shown to be effective in practical systems. This heuristic was used in the *cylinder shuffling* technique [109, 85], which monitors the accesses of disk cylinders over some period of time and then reorders the cylinders based on the measured frequencies. The cylinder shuffling technique reduces the mean seek time by 45% to 50% [109] and the disk service time by up to 10% [85]. Moreover, the organ pipe heuristic has been employed by *iPcress* (an experimental filesystem) [100]. This filesystem observes the file access patterns and moves the files with high "temperatures" near the center of the disk. The term "temperature" here means the frequency of access divided by the file size. Furthermore, a block rearrangement policy [4] based on this heuristic was proposed. This policy copies frequently referenced blocks from their original locations to a reserved space near the center of the disk. It was implemented by modifying a UNIX device driver. Simulation results showed that seek times can be substantially reduced by copying only a small number of blocks to a reserved space in the middle of the disk.

The effectiveness of block rearrangement has not been investigated for VOD servers, where the block access size is much larger than that of traditional filesystems. The *constrained block allocation* technique [78] is of the most relevance to this work. This technique constrains the separation of blocks of a strand (stored stream) in a multimedia storage server so as to guarantee bounds on access times of successive blocks of a strand

and thus meet the continuous playback requirement. This technique, however, does not utilize the skewness in access patterns. In another study [34], the allocation of blocks to different disks in a disk array was investigated without considering the way they should be placed on each of them.

3.3 Adaptive Block Rearrangement for VOD Servers

The *Adaptive Block Rearrangement* proposed here exploits the high locality of reference in video access patterns. This policy observes the accesses to videos and ensures that the videos with comparable access frequencies are located close to each other on the disks. It triggers a rearrangement process whenever it expects significant performance gains, it has sufficient confidence in the measurements of access frequencies, and the server is not too busy. The potential performance gains can be estimated by determining the level of discrepancy between the current placement and the optimal placement. Operator's hints can be used to specify the expected popularity of each video, which may be expressed as initial scores. These hints can prevent unnecessary rearrangements and can allow the rearrangement process to start as soon as the server is lightly loaded.

This study presents two algorithms for the adaptive block rearrangement: *Centered-Layout* and *Sequential-Layout*. The centered-layout algorithm rearranges the blocks on each disk so that the overall placement matches a variation of the organ pipe heuristic. The target layout is similar to that achieved by the organ pipe heuristic except that the blocks of each video on each disk are divided into two groups, with one group being placed above the center of the disk and the other below the center. The distance from the center decreases with the access frequency. By contrast, the sequential-layout algorithm

relocates the blocks on each disk according to their access frequencies, with the blocks of the most popular video being stored first at the edge of the disk.

This study also considers a third alternative, called *Random Layout*, where videos are randomly placed on each disk. Figure 3.1 illustrates the different (target) layouts for a disk containing data blocks of five videos. In the figure, vi represents all the blocks of video i on a certain disk, vi^1 represents the upper half of the blocks of video i on a disk, and vi^2 represents the lower half. The access frequency decreases from $v1$ to video $v5$.

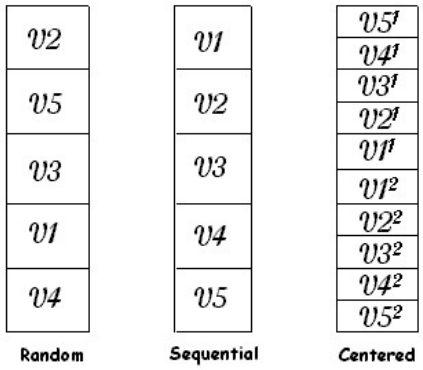


Fig. 3.1: Block Arrangement

Two degrees of randomness are examined in implementing random layouts: *Random+*, and *Random++*. In Random+, the blocks of the same video on each disk are stored contiguously, and the location of the first block is random. Random+ may be a result of aggressive constrained block allocation. In a system where no rearrangement is done, the constraint that the blocks of the same video are stored contiguously may not be met. Thus, Random++ offers a higher degree of randomness to better characterize such a system. With Random++, the blocks of each video on each disk are divided into

n groups. The blocks of each group are stored contiguously, and each group is randomly placed. In this study, n is set to 4 as much larger or smaller values may not be realistic.

These rearrangements are worthwhile and possible for the following reasons.

- They can improve performance significantly as shown in Section 3.5.
- They can be performed only when the servers are idle or lightly loaded. Thus, the incurred overheads can be concealed partially or fully as these periods of time are not typically uncommon in VOD applications.
- They can be performed incompletely or suboptimally while attaining worthy performance benefits.
- They need to be performed only occasionally because video popularities may not change often, and the releases of very hot videos happen infrequently.
- They can take reasonable times because of the advances in storage subsystems.

3.4 Performance Evaluation Methodology

Let us now discuss the workload characteristics, the simulation platform, and essentials of disk drive modeling.

3.4.1 Workload Characteristics

In accordance with prior work, this study assumes that the arrivals of the requests to VOD servers follow a Poisson Process with an average arrival rate λ. Hence, the inter-arrival time is exponentially distributed with a mean $T = 1/\lambda$. The study also

assumes that the accesses to videos are highly localized and follow Zipf's distribution [18]. Besides, it assumes that the video duration is 90 minutes unless otherwise indicated and that the videos are stored using the MPEG-II compression standard with a data rate of 3 Megabits per second. Furthermore, the number of videos for different disk configurations is selected so that all the disks are nearly full. Finally, this study assumes that the videos are striped across all disks on 1/2-second [19] or 1/5-second intervals. Interval length, interleaving unit size, and stripe unit size are used interchangeably in this thesis. Previous studies [61] show that striping on a very large fine grain is not advantageous because the seek overhead will dominate the disk access time. Conversely, striping larger blocks increases the buffer space requirements. Hence, the stripe unit size should be based on a reasonable tradeoff. A study for determining the stripe unit size for multimedia file servers was presented in [98].

3.4.2 Simulation Platform

The simulated system is a disk array of Quantum *Atlas10K*. Atlas 10K is a high-end disk designed to meet the high-performance requirements of data-intensive server, workstation, and storage subsystem applications [77], including online transaction processing (OLTP), 3-D image rendering, and broadcast video subsystems [77]. Table 3.1 summarizes the main disk parameters.

The evaluation study here used *DiskSim* [37] to simulate the disk unit in the disk array. DiskSim was used in several research studies [35, 36, 114, 115]. The results here were collected by feeding DiskSim with validated disk parameters obtained from [38] and with synthetic workloads discussed in the previous subsection. Synthetic workloads were

generated for each set of input parameters (stripe unit size, disk array size, etc) based on the centered, sequential, and random layouts.

Table 3.1: Main Disk Parameters

Parameter	Value
Storage Capacity	9.1 GB
Rotation Speed	10,025 rpm
Blocks Per Disk	179,389,986
Number of Cylinders	1004
Number of Surfaces	6
Full Strobe Seek Time	10.82800 ms
Number of Buffer Segments	10
Number of Write Segments	1
Segment Size	374 blocks

The results in [4] show that a little interaction exists between disk head scheduling and block rearrangement. Thus, the analysis in this study is limited to the *Shortest Seek Time First* algorithm.

3.4.3 Disk Drive Modeling

The disk access time is composed of a head-positioning time and a data transfer time. The head-positioning time consists of a seek time and a rotational time. Both the seek time and the rotational time depend primarily on the disk technology. The seek time, however, depends on the request pattern, the employed disk scheduling algorithm, and the block arrangement as well as the disk technology. The disk seek time consists of

- a speedup, where the arm is accelerated until it reaches half of the seek distance or a fixed maximum velocity,
- a coast for long seeks, where the arm moves at its maximum velocity,

- a slowdown, where the arm is brought to rest close to the desired track, and
- a settle, where the disk controller adjusts the head to access the desired location [86].

Short seeks, therefore, are dominated by the speedup time, whereas long seeks are dominated by the coast time. Very short seeks are dominated by the settle time. Additional details on disk drive modeling can be found in [86].

In order to ensure high accuracy, the evaluation study here used actual measurements to determine the mean seek time for a given seek distance.

3.5 Simulation Results

Let us now discuss the main simulation results, considering three disk performance metrics: seek distance, seek time, and access time. The improvements achieved by the centered and sequential layouts are reported with respect to the average case of 120 randomly generated layouts. (This average case is referred to simply as the random layout.) Conducting an exhaustive test of all possible layouts is time prohibitive.

3.5.1 Improvements by the Centered and Sequential Layouts

Let us first discuss the improvements achieved by the centered and sequential layouts for two stripe unit sizes: 0.5 second and 0.2 second. The evaluation in this subsection is based on a disk array of four disks.

3.5.1.1 Case I: 0.5-Second Stripe Unit

Figures 3.2, 3.3 and 3.4 compare the layouts in terms of the mean seek distance, the mean seek time, and the mean access time, respectively. The three performance metrics are shown versus the mean request inter-arrival time. The range of the mean inter-arrival times is selected so that the mean inter-arrival time that is equal to the mean disk access time falls within that range. As expected, the centered layout achieves the shortest seek distance. Namely, it reduces the mean seek distance by approximately 48% to 53%. This translates to an improvement of about 23% to about 27% in the mean seek time and an improvement of about 12% in the mean access time. On the other hand, the improvements achieved by the sequential layout in the mean seek distance, the mean seek time, and the mean disk access time approximately range from 29% to 37%, from 15% to 19%, and from 10% to 11%, respectively. Note that the seek distance with the centered layout is about 25% to 29% shorter than that with the sequential. The centered layout performs better than the sequential because the disk head tends to be near the centered of the disk most of the time rather than near the edge of the disk. Therefore, the head travels a shorter distance to reach any cylinder in the disk.

Figure 3.5 compares the improvements achieved by the centered-layout rearrangement in the three performance metrics. The figure demonstrates that the improvement in the seek distance substantially surpasses that in the seek time. This is due to the aggressive advances in the disk technology. Similarly, the improvement in the seek time translates to a much lower improvement in the disk access time because the placement primarily impacts the seek component of the disk access time.

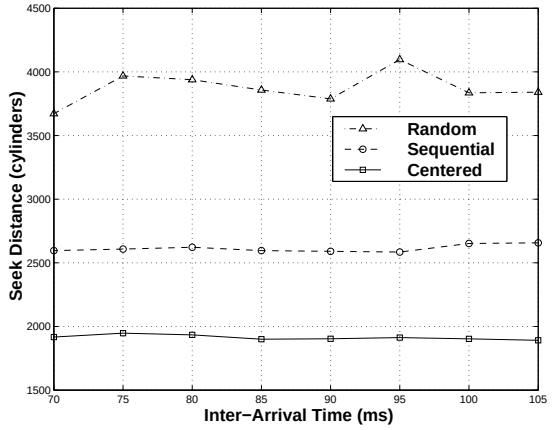


Fig. 3.2: Effects of Layouts on Seek Distance (0.5-Second Stripe Unit, 4 Disks)

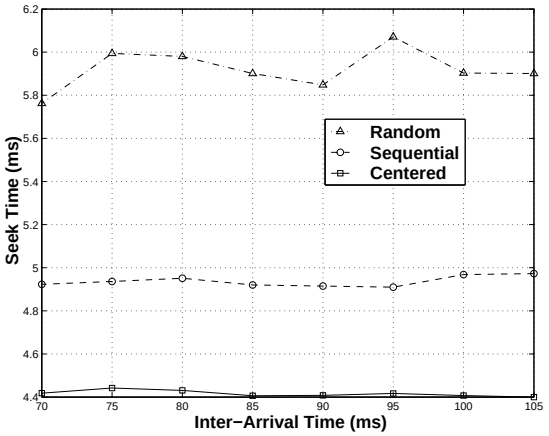


Fig. 3.3: Effects of Layouts on Seek Time (0.5-Second Stripe Unit, 4 Disks)

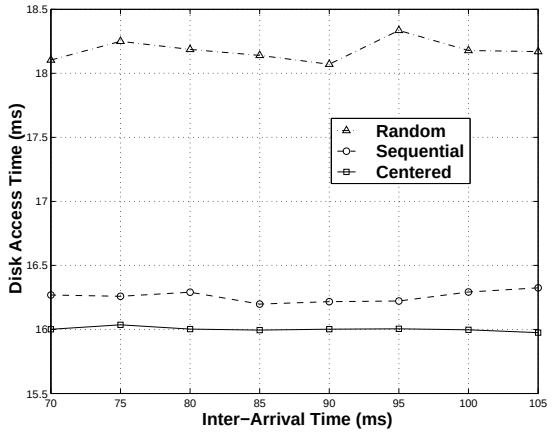


Fig. 3.4: Effects of Layouts on Access Time (0.5-Second Stripe Unit, 4 Disks)

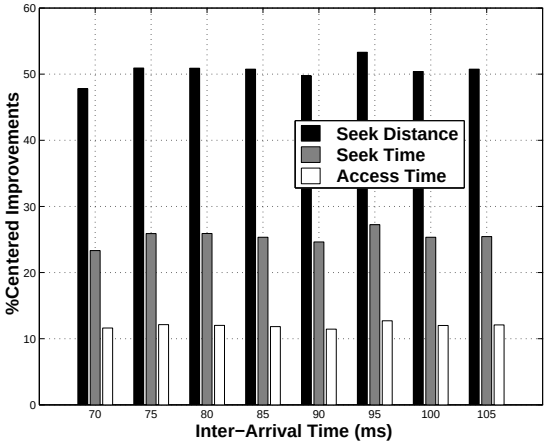


Fig. 3.5: Various Improvements Achieved by the Centered Layout (0.5-Second Stripe Unit, 4 Disks)

3.5.1.2 Case II: 0.2-Second Stripe Unit

Figures 3.6, 3.7, and 3.8 demonstrate the impacts of the layouts on the three performance metrics, while Figure 3.9 compares the improvements achieved by the centered

layout in these metrics. The results regarding the seek distance and the seek time are almost identical to those for a 0.5-second stripe unit because the size of the stripe unit primarily impacts the transfer time component of the disk service time. This is also the reason that explains the difference in the case of the mean disk access time. The effect of the stripe unit size will be discussed further in Subsection 3.5.3.

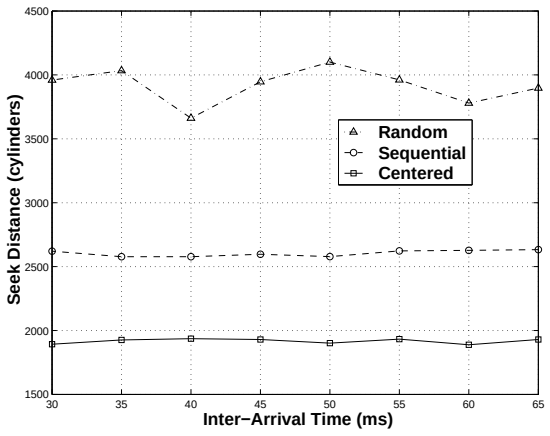


Fig. 3.6: Effects of Layouts on Seek Distance (0.2-Second Stripe Unit, 4 Disks)

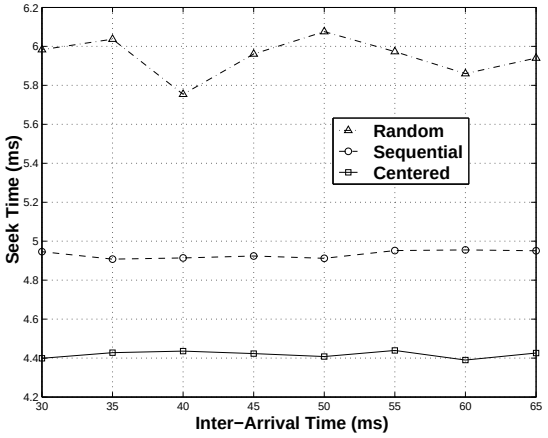


Fig. 3.7: Effects of Layouts on Seek Time (0.2-Second Stripe Unit, 4 Disks)

Table 3.2 summarizes the improvements achieved by the centered and sequential layouts in the three performance metrics for both stripe unit sizes.

	Sequential Layout		**Centered Layout**	
Improvement in \ Stripe Unit	0.2 Second	0.5 Second	0.2 Second	0.5 Second
Seek Distance	30%	33%	47%	51%
Seek Time	15%	17%	23%	26%
Disk Access Time	10%	10%	13%	12%

Table 3.2: Summary of Improvements of Centered and Sequential Layouts (4 Disks)

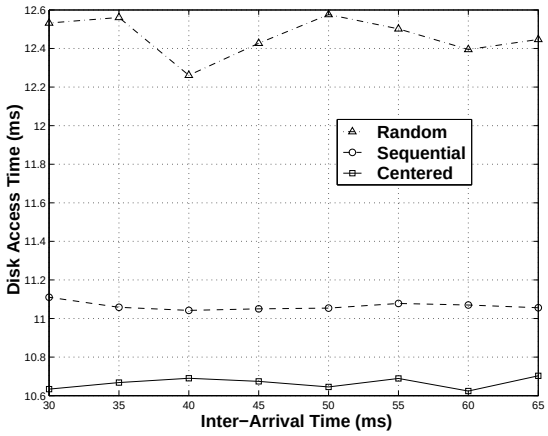


Fig. 3.8: Effects of Layouts on Access Time (0.2-Second Stripe Unit, 4 Disks)

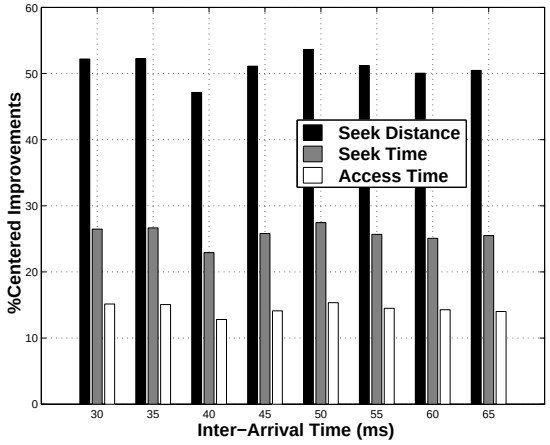


Fig. 3.9: Various Improvements Achieved by the Centered Layout (0.2-Second Stripe Unit, 4 Disks)

3.5.2 Effect of the Number of Disks

Let us now compare how the layouts perform for different disk array sizes. Here, the size of the disk array is varied from 4 to 20 disks, and the number of videos is varied accordingly to keep all the disks nearly full.

Let us start with the results for a stripe unit of 0.5 second. Figures 3.10, 3.11, and 3.12 plot the mean seek distance, the mean seek time, and the mean access time for the three layouts versus the number of disks, respectively. For the random and sequential layouts, the impact of the disk array size is almost negligible except for some occasional dips in the random-layout case. By contrast, with the centered layout, the three performance metrics generally decrease with the number of disks. (This observation will be confirmed for a 0.2-second stripe unit.) This behavior can be explained as follows. As the number of videos increases with the number of disks, the share of each disk of

each video's blocks diminishes. Because of this and the high locality of reference, the centered-layout rearrangement causes most of the disk accesses to fall within a smaller area around the center of the disk.

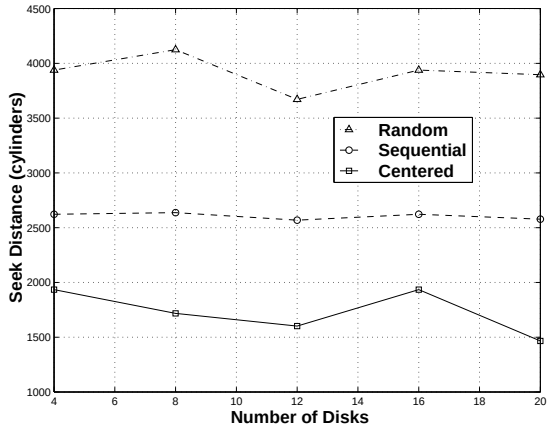


Fig. 3.10: Effect of Number of Disks on Seek Distance (0.5-Second Stripe Unit)

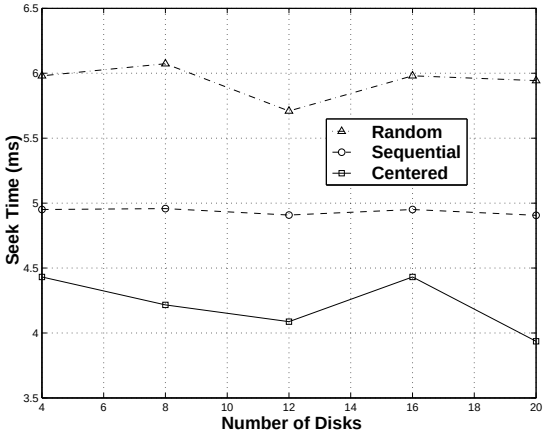


Fig. 3.11: Effect of Number of Disks on Seek Time (0.5-Second Stripe Unit)

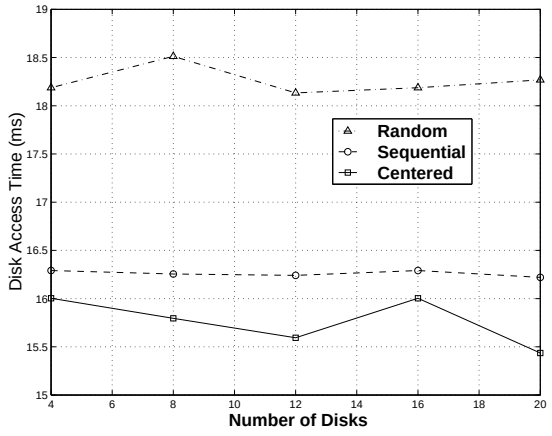


Fig. 3.12: Effect of Number of Disks on Disk Access Time (0.5-Second Stripe Unit)

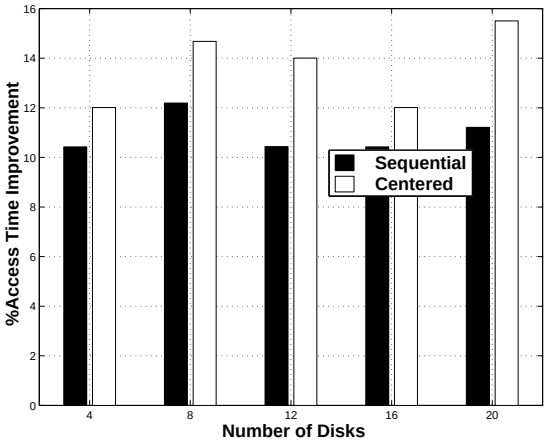


Fig. 3.13: Effect of Number of Disks on Improvements (0.5-Second Stripe Unit)

Figure 3.13 plots the mean disk access time improvements achieved by the centered and sequential layouts versus the size of the disk array. These results indicate that the improvement achieved by the centered layout increases from about 12% when the number of disks is 4 to about 16% when the number of disks is 20. In contrast, not much of a difference exists in the case of the sequential layout. The actual improvements achieved by the centered and sequential layouts are expected to grow more significantly with the size of the disk array than those shown in Figure 3.13. In the simulation, only 120 random layouts are generated (for each set of parameters). This fixed and small number provides an increasingly and substantially reduced coverage of the set of all possible layouts as the number of disks increases. This number, however, is necessary to avoid the factorial growth in the simulation times.

Let us now discuss the results for a stripe unit of 0.2 second. Figures 3.14 and 3.15 show the mean seek distance and the mean seek time for the three layouts versus the number of disks, respectively. Unsurprisingly, these results are very similar to those in Figures 3.10 and 3.11 for a stripe unit of 0.5 second. Figure 3.16 depicts the mean disk access time for the three layouts versus the number of disks. Despite that these results differ quantitatively from those for a stripe unit of 0.5 shown in figure 3.12, they exhibit a similar behavior. The quantitative difference is due to the difference in the data transfer times.

3.5.3 Effect of the Stripe Unit Size and the Number of Disks

Let us now discuss the impacts of both the stripe unit size and the number of disks. Figure 3.17 depicts the improvement achieved by the centered layout in the mean

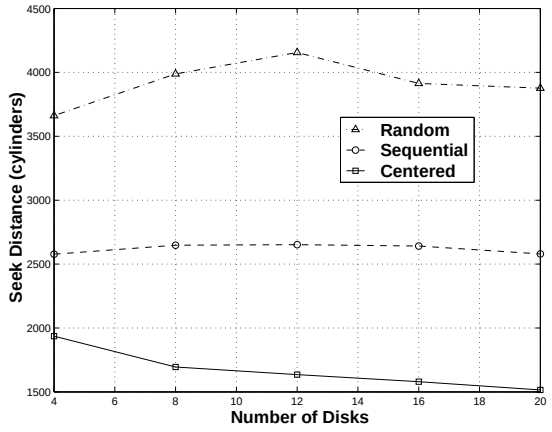


Fig. 3.14: Effect of Number of Disks on Seek Distance (0.2-Second Stripe Unit)

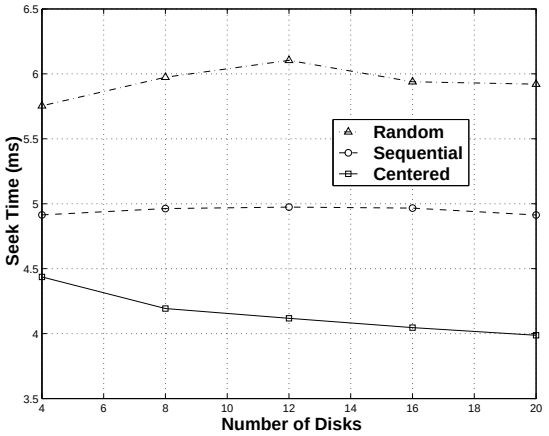


Fig. 3.15: Effect of Number of Disks on Seek Time (0.2-Second Stripe Unit)

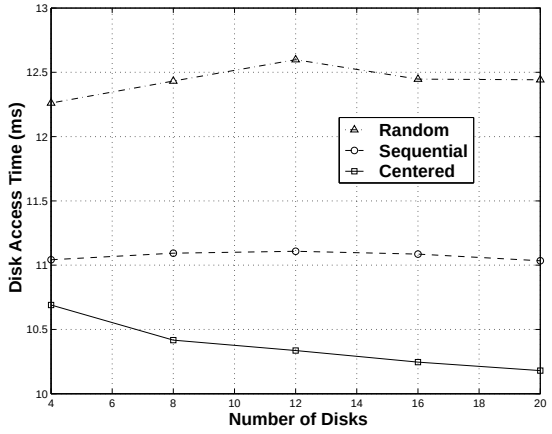


Fig. 3.16: Effect of Number of Disks on Disk Access Time (0.2-Second Stripe Unit)

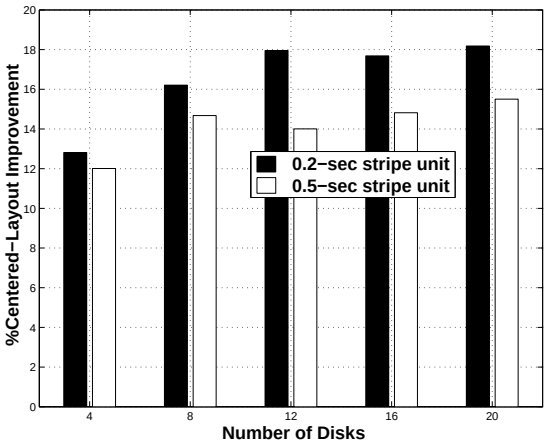


Fig. 3.17: Effect of Stripe Unit Size on Centered-Layout Improvement

disk access time versus the size of the disk array. As expected, the improvement is higher for the smaller stripe unit because a smaller stripe unit demands a shorter transfer time. Hence, the improvement in the seek time produces a higher improvement in the disk access time.

3.5.4 Effect of Video Duration

In order to discuss how the video durations (lengths) impact the performance results, let us consider two different video sets. The first set contains 90-minute videos, whereas the second set contains a balanced mix of 60-minute and 90-minute videos. Consequently, the number of videos in the second set is about 25% larger than the first.

Figure 3.18 plots the improvement achieved by the centered layout in the mean disk access time for both video sets versus the array size. These results generally indicate that the centered layout improves performance slightly better in the mixed-video case than in the uniform case. In comparison, Figure 3.19 shows that the sequential layout improves performance more significantly in the mixed-video case than in the uniform case. Both figures are for a system with 0.2-second stripe units. The improvements (achieved by the centered and sequential layouts) are expected to grow more dramatically with decreasing video durations if the number of randomly generated layouts is adjusted to provide the same coverage of all possible layouts.

3.5.5 Effect of the Degree of Randomness

Finally, let us discuss the improvement achieved by the centered layout with respect to the degree of randomness available to the random layout. Figure 3.20 shows the improvement in the average disk access time versus the size of the disk array for the two cases: Random+ and Random++. The results are for a stripe unit of 0.5 second. Note that the improvement is lower with the higher degree of randomness. This is because, in Random++, the blocks of each video on each disk are divided into

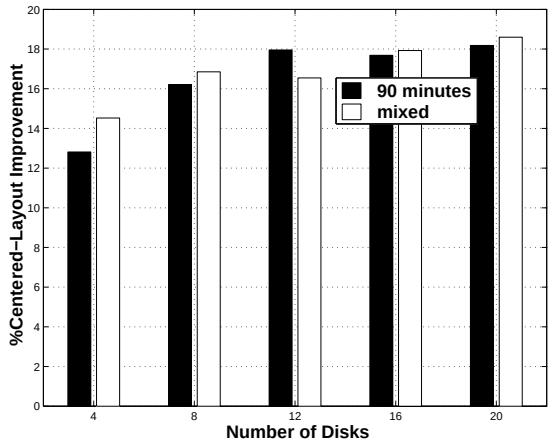


Fig. 3.18: Effect of Video Durations on Centered-layout Improvement (0.2-**Second Stripe Unit**)

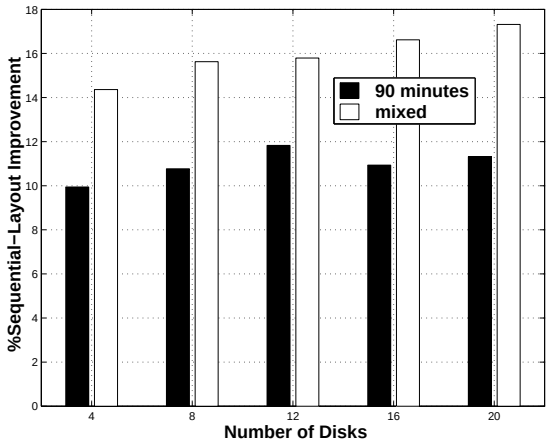


Fig. 3.19: Effect of Video Durations on Sequential-layout Improvement (0.2-**Second Stripe Unit**)

four groups, which are randomly placed. These groups share the same access frequency, thereby decreasing the probability of poor placement.

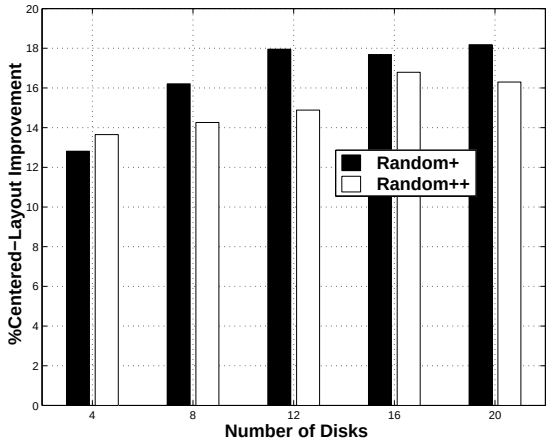


Fig. 3.20: Effect of Degree of Randomness on Centered-layout Improvement (0.5-Second Stripe Unit)

3.6 Conclusions

This study has proposed an *Adaptive Block Rearrangement* policy for VOD servers and has presented two algorithms for this rearrangement: *Centered-Layout* and *Sequential-Layout*. Both algorithms rearrange blocks according to their access frequencies. The centered-layout algorithm places the more popular blocks nearer to the center of the disk, whereas the sequential algorithm places these blocks nearer to the edge of the disk.

The advantages of the adaptive rearrangement policy on disk performance have been demonstrated through extensive simulation. The benefits of the centered layout relative to random layouts can be summarized as follows. The improvements generally increase with the disk array size, provided that the videos are striped across all disks. For example, in a disk array of 4 disks, the centered-layout rearrangement reduces the mean seek distance by about 50%. This translates to an improvement of about 20% to 25% in the mean seek time and an improvement of more than 13% in the mean access time. In contrast, for a disk array of 20 disks, the mean seek distance is minimized by more than 60%, which shortens the mean seek time by more than 30% and the mean disk access time by about 15%. Similarly, the sequential layout performs better than random layouts but worse than the centered. Specifically, the mean seek distance is about 25% to 45% longer than that of the centered. The results also indicate that the improvements achieved by the adaptive rearrangement increase as the stripe unit size and the video duration(s) decrease.

Chapter 4

Provision of Time of Service Guarantees

This chapter presents a new class of scheduling policies, called *Next Schedule Time First* (NSTF), which provides customers with hard time of service guarantees. It also presents alternative implementations of NSTF and shows, through extensive simulation, that NSTF can deliver outstanding performance benefits.

4.1 Introduction

Providing time of service guarantees through scheduling can enhance customer-perceived QoS and can influence customers to wait, thereby increasing server throughput. Unlike most other policies, FCFS is believed to provide time of service guarantees. In [103], it is stated that FCFS can provide time of service guarantees, which are either precise or longer than the actual times of service. By contrast, this study demonstrates that FCFS may violate these guarantees. It also shows that FCFS is incapable of producing accurate time of service guarantees. Specifically, the average deviation of the actual times of service from the time of service guarantees ranges from 20 seconds to more than 4.5 minutes!

This study addresses the shortcomings of FCFS by proposing a new class of scheduling policies, called *Next Schedule Time First* (NSTF). NSTF provides customers with schedule times, and it guarantees that they will be serviced no later than and

accurately at their schedule times. The basic idea of the NSTF is to assign schedule times to incoming requests and to perform scheduling based on these schedule times rather than the arrival times. In the absence of VCR-like operations, a VOD server knows exactly when resources will become available for servicing new requests because each running stream requires a fixed playback time. Hence, when a new request calls for the playback of a video with no waiting requests, NSTF assigns that request a new schedule time that is equal to the closest unassigned completion time of a running stream. If the new request, however, is for a video that has already at least one waiting request, then NSTF assigns it the same schedule time assigned to the other waiting request(s) because all requests for a video can be serviced together using only one stream. Applying VCR-like operations, which are typically supported by using contingency channels [28], leads to early completions and thus servicing some requests earlier than scheduled.

When all customers waiting for the playback of a video defect (i.e., cancel their requests), their schedule time become available and can be used by other customers. This leads to two variants of NSTF: *NSTFn* and *NSTFo*. NSTFn assigns freed schedule times to incoming requests, whereas NSTFo assigns them to existing requests with waiting time guarantees beyond a certain threshold, and thus likely to defect. This study shows that NSTFn and NSTFo are special cases of a policy, called *Generalized Next Schedule Time First* (GNSTF). It also presents three variants of NSTFo: *NSTFo-FCFS*, *NSTFo-MQL*, and *NSTFo-MFQL*, which differ in the selection criterion of existing requests that will be assigned better schedule times. These variants select requests on a FCFS, a MQL, or a MFQL basis, respectively. NSTFo-MQL and NSTFo-MFQL combine the advantages of FCFS and MQL/MFQL.

The effectiveness of the proposed policies is demonstrated through extensive simulation, considering five performance metrics. These metrics are the overall customer reneging (defection or turn-away) percentage, the average request waiting time, the number of violations of time of service guarantees, the average deviation from the time of service guarantees, and unfairness. The reneging percentage is the most important metric because it translates to the number of customers serviced concurrently and to server throughput. Unfairness measures the bias against unpopular videos. All other performance metrics signify QoS. The impacts of customer waiting tolerance and server capacity (or server load) on the results are also examined.

The rest of the chapter is organized as follows. Section 4.2 explains why FCFS may violate its time of service guarantees. Section 4.3 presents NSTF and its variants. Section 4.4 discusses the simulation platform, the workload characteristics, and the main simulation results. Finally, the last section draws conclusions.

4.2 Time of Service Guarantees Through FCFS

This section shows why FCFS may violate its time of service guarantees. Let us first discuss how a server may provide time of service guarantees and let us assume, just for now, the absence of VCR-like operations (pause, resume, fast forward, and fast rewind). In the absence of these operations, a VOD server knows exactly when each running stream will complete. A channel becomes available whenever a running stream completes, so the server can assign completion times of running streams as time of service guarantees to incoming requests. Obviously, the server should assign the closest completion times first. Thus, when a request comes and joins an empty waiting

queue, the server grants that request a new time of service guarantee. If the incoming request, however, joins a queue that has at least one request, then the new request can be given the same time of service guarantee as the other request(s) waiting in the queue because of batch scheduling. Let us now discuss the impact of VCR-like operations. Applying a pause, a fast forward, or a fast rewind operation can be considered as an early completion if the corresponding client is the only recipient of the stream because VOD servers typically support interactive operations by using contingency channels [28]. Early completions lead to servicing some requests earlier than their time of service guarantees.

The following example explains why with FCFS, the server may violate time of service guarantees. Let us assume that $t1 < t2 < t3 < t4 < t5 < t6$ and that $i \neq j$. Let us also assume that at the current state of the server, $t5$ is the next stream completion time that has not yet been assigned, and $t6$ is the completion time that immediately follows. At time $t1$, a new request, $R1$, arrives and joins the empty waiting queue i. Thus, the server gives $R1$ the time of service guarantee $t5$. At time $t2$, a new request, $R2$, arrives and joins the empty waiting queue j. Hence, the server gives $R2$ the time of service guarantee $t6$. At time $t3$, a new request, $R3$, arrives and joins the waiting queue i, which already has the request $R1$. So, the server assigns $R3$ the time of service guarantee $t5$. Assume that at time $t4$, $R1$ defects (probably because it was given a far time of service guarantee). Thus, using FCFS (which selects the queue with the oldest request), the server will service $R2$ before $R3$, although $R3$ was given a better time of service guarantee. Assuming that the time of service guarantee $t4$ is precise (i.e., equal to the actual time of service for the request(s) granted this guarantee), the server will violate the time of service guarantee of $R3$! Violations of time of service guarantees were

also observed during simulations that use the same model of waiting tolerance used in [103].

4.3 Next Schedule Time First (NSTF)

The proposed *Next Schedule Time First* (NSTF) class of scheduling policies eliminates the shortcomings of FCFS. Namely, NSTF assigns schedule times to incoming requests, and it guarantees that they will be serviced no later than scheduled. In addition, it ensures that these schedule times are very close to the actual times of service. NSTF, therefore, improves both QoS and server throughput. Improving throughput is attained by influencing the waiting tolerance of customers. In the absence of time of service guarantees, customers are more likely to defect because of the uncertainty of when they will start to receive services. Another desirable feature of NSTF is the ability to prevent starvation (as FCFS).

NSTF selects for service the queue with the closest schedule time. The schedule times are assigned as follows. When a new request for a video with no waiting requests arrives, NSTF assigns that request a new schedule time. This schedule time is equal to the closest unassigned completion time of a running stream. By contrast, when a new request joins a waiting queue that has at least one request, NSTF assigns the new request the same schedule time assigned to the other request(s) because all requests in a queue can be serviced together. Note that a schedule time estimates the time when the server starts to deliver a stream and not the time when the presentation actually starts. Thus, it does not include the network latency or the buffering time at the client for smoothing out the delay jitter.

Next, variants of NSTF and their implementations are discussed. For the sake of comparison, an implementation of FCFS that provides customers with time of service guarantees (which may be violated as discussed in Section 4.2) is also discussed. FCFS that may provide such guarantees is referred to as *FCFSg*, and FCFS that provides no guarantees is referred to simply as *FCFS*.

4.3.1 Variants of NSTF

When all waiting requests for a video get canceled, their schedule time becomes available and can be used by other requests. This leads to two variants of NSTF: *NSTFn* and *NSTFo*. NSTFn assigns the freed schedule times to incoming requests, whereas NSTFo assigns them to existing requests with waiting time guarantees beyond a certain threshold, and thus likely to defect if they are no reassigned better schedule times. Hence, requests that are assigned schedule times that require them to wait beyond a certain threshold should be notified that they may be serviced earlier. This notification may motivate them to wait. All other requests, however, should be given accurate schedule times. NSTFn is simpler than NSTFo but is more biased against old requests.

Let us now discuss how NSTFo works. NSTFo assigns each freed schedule time to an appropriate waiting queue that meets the following three conditions.

- It is nonempty.
- Its assigned schedule time is worse than the freed schedule time.

- The expected waiting time for each request in it is beyond a certain threshold. This threshold is set to 5 minutes in this study because of the models of waiting tolerance used (Section 4.4).

If no candidate is found, NSTFo grants the freed schedule time to a new request. In contrast, if more than one queue meet these conditions, it selects the most appropriate one. This study thus presents three variants of NSTFo: *NSTFo-FCFS*, *NSTFo-MQL*, and *NSTFo-MFQL*. These variants differ in the selection criterion of existing requests that will be assigned better schedule times. NSTFo-FCFS selects the queue with the longest waiting time, whereas *NSTFo-MQL* selects the longest queue, and *NSTFo-MFQL* selects the queue with the largest factored length. NSTFo-MQL and NSTFo-MFQL combine the benefits of FCFS and MQL/MFQL by assigning schedule times on a FCFS basis and re-assigning freed schedule times on a MQL/MFQL basis.

NSTFn and NSTF can be generalized by a policy, called *Generalized Next Schedule Time First* (GNSTF). Let us reconsider the second condition for a queue that is to be assigned a better schedule time, which is that the current schedule time of that queue is worse than the freed schedule time. If the difference between the two schedule times, however, is too small, the freed schedule time may effectively get wasted for not being granted to a well-deserving queue. Thus, GNSTF changes the condition to that the schedule time of a queue is greater than the freed schedule time by a certain threshold, T. Note that GNSTF becomes NSTFo when T is zero and becomes NSTFn when T is sufficiently large.

4.3.2 Implementations of FCFSg and NSTF

This subsection presents implementations of FCFSg and NSTF in C++-like syntax. For simplicity, these implementations assume that the number of channels is greater than or equal to the number of videos, which is typically the case. A VOD server maintains a waiting queue (WQ) for each video and one running queue (RQ) for all videos. All new requests are routed to the corresponding WQs. RQ keeps track of the currently running streams. To provide time of service guarantees, the server needs to maintain an index, $RQIndex$, which points to the next stream in RQ whose completion time has not been assigned yet. The current system time is referred to as $currTime$. $WQi.guarantee$ denotes the time of service guarantee assigned to the waiting queue i. $RQ[0]$ is the first element of RQ, and it corresponds to the *next* running stream. $RQ[RQindex].compTime$ denotes the completion time of the next running stream whose completion time has not been assigned yet. $RQIndex$ must be initialized to -1.

Figure 4.1 shows an implementation of FCFSg. Note that $RQIndex$ is decremented every time a schedule time is assigned and is incremented every time the only waiting request in a queue gets canceled or when a queue is selected for service.

Figure 4.2 shows an implementation of NSTF (with all its variants). The server here needs to maintain a free pool ($freePool$) of freed schedule times. This pool can be implemented using a priority queue, where schedule times are placed in an ascending order. When a request for a video with no waiting requests arrives, the server first tries to assign it a schedule time from the top of $freePool$. $WQi.schedule$ denotes the schedule time assigned to the waiting queue i. If $freePool$ does not contain any live schedule

```
Event: Initialization
    RQIndex = -1;
Event: Arrival of request R to WQi
    if (a channel is available){  // the server is not fully loaded
        Service the request immediately;
        RQIndex++;
    }
    else {  // the server is fully loaded
      if (WQi.empty()){
          WQi.guarantee = RQ[RQIndex].compTime;
          RQIndex−−;
      }
      Inform R of the guarantee WQi.guarantee;
    }
Event: Cancellation of request R in WQi
    if (R is the only request in WQi)
      RQIndex++;
Event: Completion of a running stream
    if (at least one request is waiting){
      Service the queue with the longest waiting request;
      RQIndex++;
    }
```

Fig. 4.1: An Implementation of FCFSg

times (i.e. schedule times that are further than the current time), then the server assigns it a new schedule time that corresponds to the next un-assigned completion time.

```
Event: Initialization
  RQIndex = -1;
Event: Arrival of request R to WQi
    if (a channel is available){  // the server is not fully loaded
       Service the request immediately;
       RQIndex++;
    }
    else {  // the server is fully loaded
      if (WQi.empty()){
        while (!freePool.empty()){
           // remove the expired schedule times from the free pool
          if (freePool.top() < currTime){
            RQIndex++;
            freePool.pop();
          } // end of if
        } // end of while
        if (!freePool.empty()){
           // assign the closest schedule time in the free pool to WQi
          WQi.schedule = freePool.top();
          freePool.pop();
        } // end of if
        else{ // assign the next un-assigned schedule time to WQi
          WQi.schedule = RQ[RQIndex].compTime;
          RQIndex--;
        }//end of else
      } // end of if
      Inform R of the schedule time WQi.schedule;
    } // end of else
Event: Cancellation of request R in WQi
    if (R is the only request in WQi){
      if (policy == NSTFn)
        freePool.add(WQi.schedule); // add the freed schedule time to the free pool
      else if (there is queue WQj that meets the 3 conditions and the selection
              criterion of FCFS, MQL, or MFQL){
        FreePool.add(WQj.schedule); // add the freed schedule time to the free pool
        WQj.schedule = WQi.schedule;
      } // end of else if
      else
        FreePool.add(WQi.schedule); // add the freed schedule time to the free pool
    } // end of if

Event: Completion of a running stream
    if (at least one request is waiting){
      Select for service the queue with the closest schedule time;
      RQIndex++;
    }
```

Fig. 4.2: An Implementation of NSTF

4.4 Performance Evaluation

This section analyzes the effectiveness of the proposed policies through simulation. It first discusses the simulation environment and workload characteristic and then presents the main results.

4.4.1 Simulation Platform

The evaluation study here was conducted by developing a simulator for a VOD server that supports various scheduling policies. The simulated server starts with a state close to the steady state of the common case to accelerate the simulation. In that state, the server delivers its full capacity of running streams, whose remaining times for completion are uniformly distributed between zero and the normal video duration. Many of these results were validated against those generated by simulating an initially unloaded server. The simulation stops after a steady state analysis with 95% confidence interval is guaranteed.

4.4.2 Workload Characteristics

Like most prior work, this study assumes that the arrivals of the requests to VOD servers follow a Poisson Process with an average arrival rate λ. Hence, the inter-arrival time is exponentially distributed with a mean $T = 1/\lambda$. The study also assumes as in previous work that the accesses to videos follow a Zipf-like distribution [18], and that the value of the skewness parameter θ is 0.271.

The studied server has 120 videos, each of which is 120-minute long. The server is examined at different loads by fixing the request arrival rate at 40 requests per minute

and varying the number of channels (server capacity) generally from 500 to 1750. To keep the discussion focused, VCR-like operations are not considered. These interactive operations can be supported by allocating contingency channels [28].

The waiting tolerance of customers is characterized by two models: *Model A* and *Model B*. In Model A, customers who receive time of service guarantees will wait for service if their waiting times are less than or equal to five minutes; the waiting times of all other customers follow an exponential distribution with a mean of 5 minutes. Model B is used in [103] and is the same as Model A except that a truncated normal distribution with a mean of 5 minutes and a standard deviation of 1.67 minutes is used in place of the exponential distribution. Truncation excludes the waiting times that are negative or greater than 12 minutes. In both these models, the evaluation study here assumes that the customers who are expected (according to their time of service guarantees) to wait longer than 12 minutes will defect immediately. Although they differ significantly, both normal and exponential distributions were used in previous studies.

4.4.3 Result Presentation and Analysis

Let us now compare the various NSTF policies with each other, with FCFSg, and with policies that do not provide time of service guarantees: FCFS, MQL, and MFQL. The considered performance metrics are the number of violations of time of service guarantees, the average deviation from these guarantees, the overall customer reneging percent, the average request waiting time, and unfairness.

Tables 4.1 and 4.2 compare FCFSg, NSTFn, and the three variants of NSTFo in terms of the number of violations of time of service guarantees and the average deviation

of these guarantees from the actual times of service when the tolerance follows Model A and Model B, respectively. (The schedule times given by NSTF act as time of service guarantees.) The numbers of violations are collected per 1000 requests, and the arrows show the variations as the server capacity increases from 500 to 1500. For example, with FCFSg under Model A, an average of 11.6 out of 1000 time of service guarantees are violated when the server capacity is 500, compared with 0.03 violations when the capacity is 1500. The number of violations decreases with the server capacity as expected. The results demonstrate that FCFSg may violate its time of service guarantees, but these violations happen very occasionally (especially for high server capacities) because FCFSg tends to overestimate significantly these guarantees. In fact, overestimating these guarantees is the most critical problem of FCFSg. With FCFSg, the actual times of service differ from the time of service guarantees by 20 seconds to more than 4.5 minutes on the average! The inaccuracy of time service guarantees leads to uncertainty of when customers will start to receive services. Customers would greatly appreciate receiving accurate schedule times so that they could plan accordingly. Moreover, customers are more likely to defect if their waiting times are overestimated. In contrast with FCFSg, NSTF produces accurate schedule times and services requests no later than scheduled. The average deviation of the actual times of service from the schedule times given by NSTFn and NSTFo are within 6 seconds and 0.2 second, respectively.

Let us now compare the three variants of NSTFo in terms of throughput, waiting times, and unfairness. Figures 4.3 and 4.4 show the performance of these variants under Model A and Model B of the waiting tolerance, respectively. The results indicate that NSTFo-MQL performs the best in terms of both throughput and waiting times, followed

Table 4.1: Violations of Time of Service Guarantees and Average Deviations from these Guarantees (Model A)

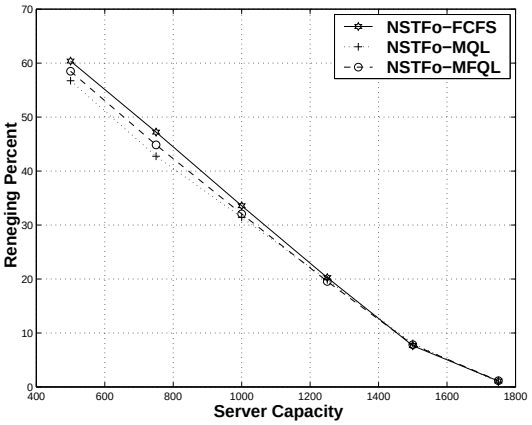
Policy	Violations per 1000 Request	Deviation (second)
FCFSg	11.6 → 0.03	226 → 19.6
NSTFn	0	3.8 → 0.29
NSTFo-FCFS	0	0.12 → 0.05
NSTFo-MQL	0	0 → 0.04
NSTFo-MFQL	0	0 → 0.05

Table 4.2: Violations of Time of Service Guarantees and Average Deviations from these Guarantees (Model B)

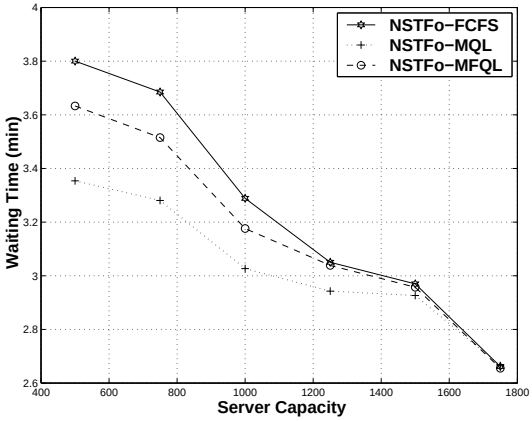
Policy	Violations per 1000 Requests	Deviation (second)
FCFSg	4.6 → 0.09	272.9 → 26.2
NSTFn	0	6.03 → 1.01
NSTFo-FCFS	0	0.14 → 0.16
NSTFo-MQL	0	0 → 0.07
NSTFo-MFQL	0	0 → 0.05

by NSTFo-MFQL. Despite that NSTFo-MQL is not the fairest, it stands out as the clear winner because fairness is far less important than throughput and waiting times. Therefore, only NSTFo-MQL will be considered in the subsequent analysis.

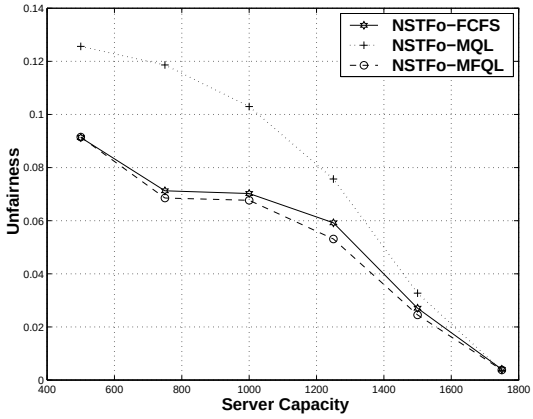
Figures 4.5 and 4.6 compare the performance of FCFSg, NSTFn, and NSTFo-MQL. The results demonstrate that NSTFo-MQL achieves better throughput and waiting times than NSTFn under both tolerance models, but NSTFn is fairer. Among the three policies, NSTFo-MQL delivers the highest throughput under both tolerance models. The NSTF policies are expected to perform even better in real systems because customers are more likely to defect with FCFSg, which tends to overestimate the waiting times. Unfortunately, the increased likelihood of defection with FCFSg is not captured very accurately by the tolerance models. In both models, the waiting tolerance of customers



(a) In Reneging Percent



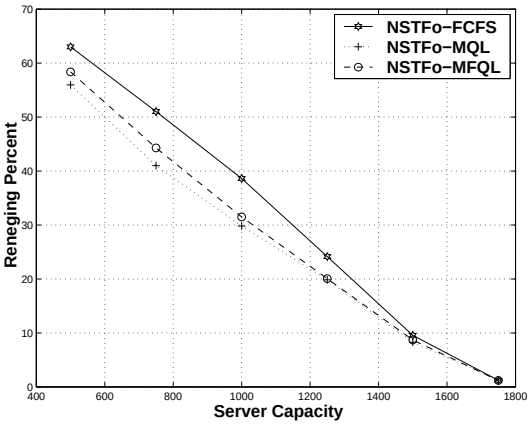
(b) In Waiting Time



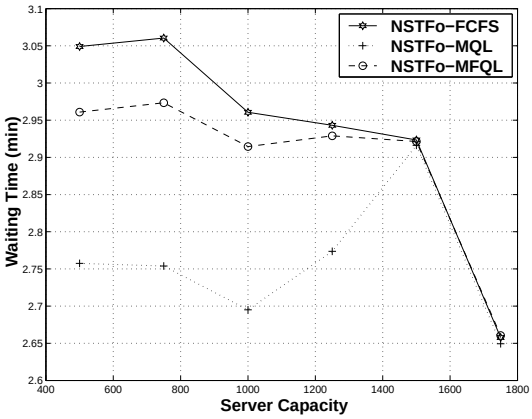
(c) In Unfairness

Fig. 4.3: Comparison among NSTFo Variants (Model A)

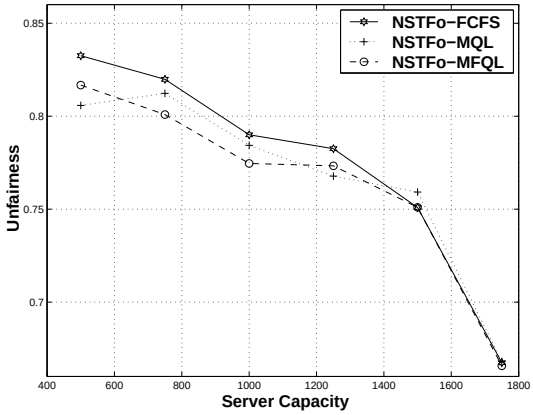
does not depend on the assigned time of service guarantees if the waiting times (according to these guarantees) may be greater than 5 minutes. This suggests that this study is not entirely fair to the proposed policies. NSTFo-MQL also generally achieves the shortest waiting times when the tolerance follows Model B. In the case of Model A, however, FCFSg generally achieves the shortest waiting times, and NSTFo-MQL performs a little worse. Under both tolerance models, FCFSg is the fairest.



(a) In Reneging Percent



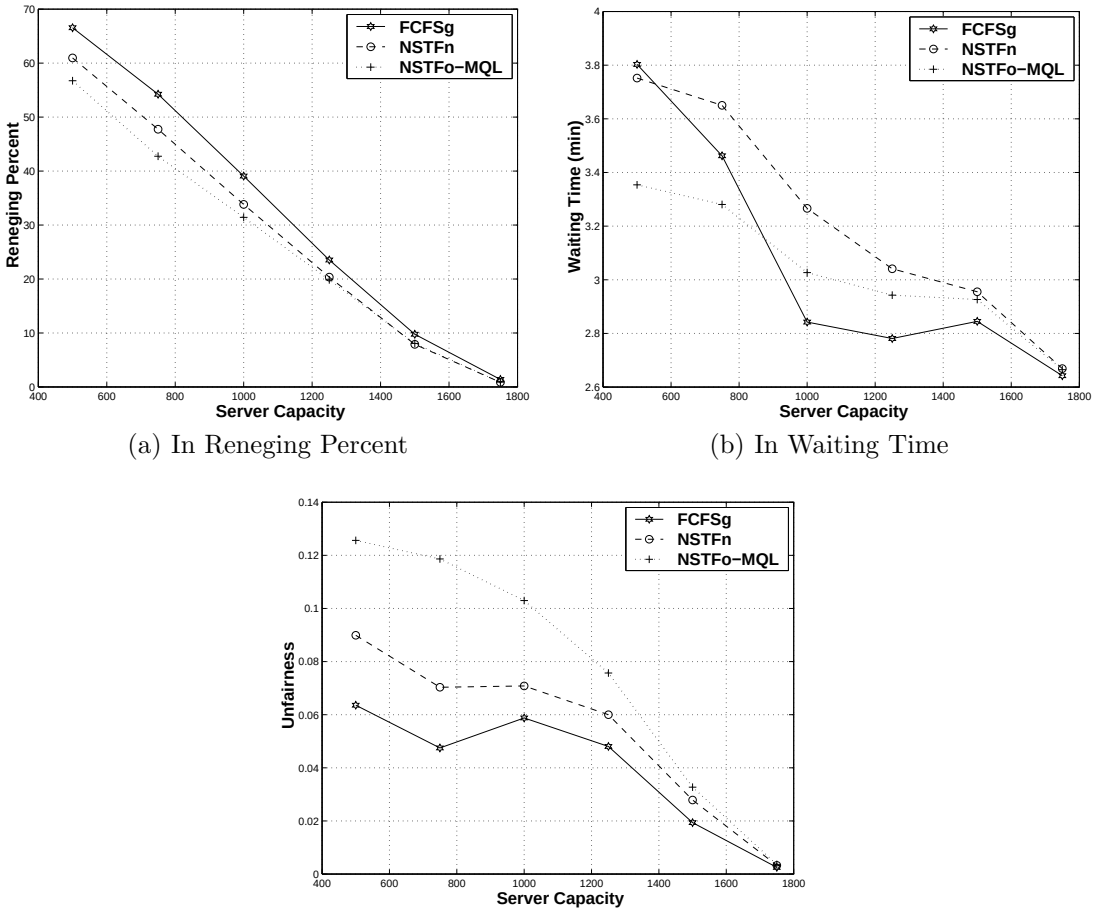
(b) In Waiting Time



(c) In Unfairness

Fig. 4.4: Comparison among NSTFo Variants (Model B)

Figures 4.7 and 4.8 compare the performance of NSTFo-MQL with policies that do not provide time of service guarantees: FCFS, MQL, and MFQL. Note that NSTFo-MQL leads to the highest throughput under Model A. It also achieves the highest throughput under Model B but only when the reneging percent is less than 20, which is the most likely operating region as much larger reneging percents would be unacceptable. As expected, MQL and MFQL perform the best in terms of waiting times, and FCFS is the fairest.



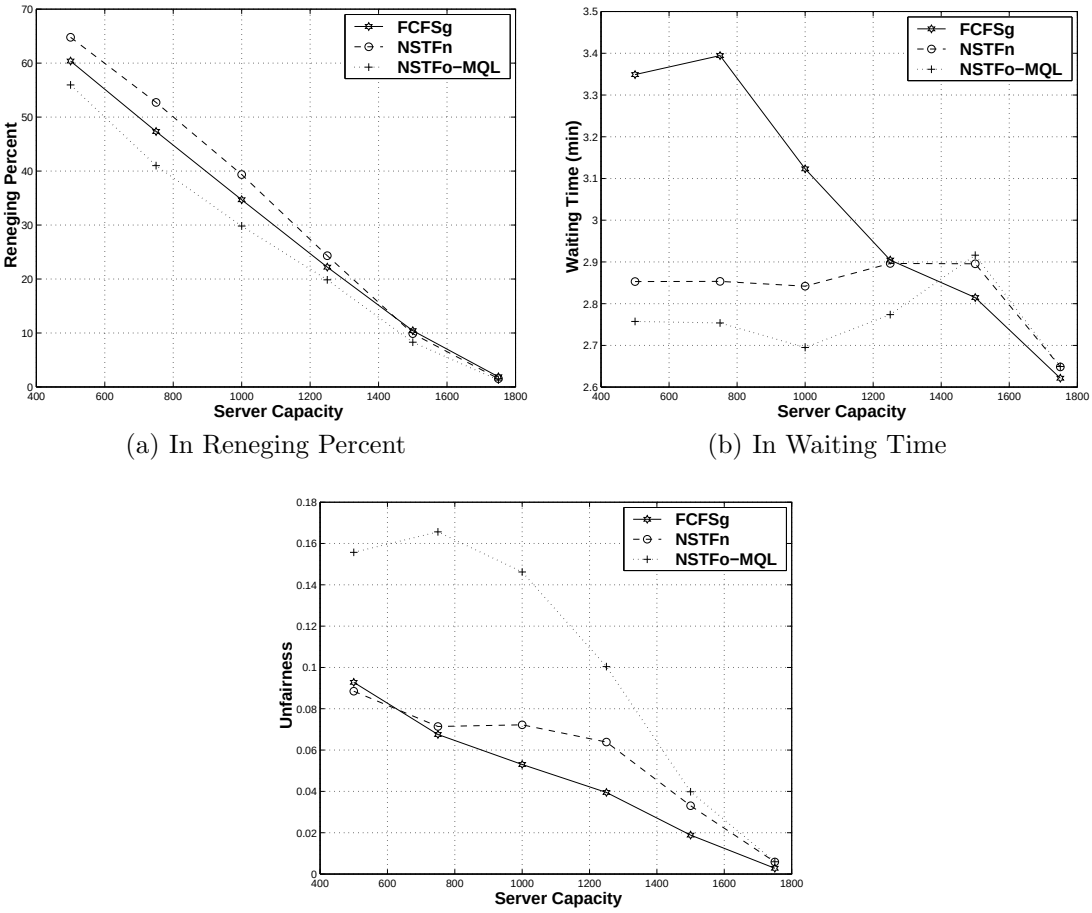
(a) In Reneging Percent

(b) In Waiting Time

(c) In Unfairness

Fig. 4.5: Comparison among FCFSg, NSTFn, and NSTFo-MQL (Model A)

Let us now examine the impact of the threshold T in GNSTF on performance. The implementation of GNSTF that uses MQL for re-assigning schedule times is considered here because of its performance benefits. Figure 4.9 shows the effect of T on throughput and waiting times when the tolerance follows Model A. The results for Model B are not shown because they exhibit a very similar behavior. The results for unfairness are not shown either because T has a little impact on it. Note that GNSTF becomes NSTFo when T approaches zero and becomes NSTFn when T is greater than a sufficiently large



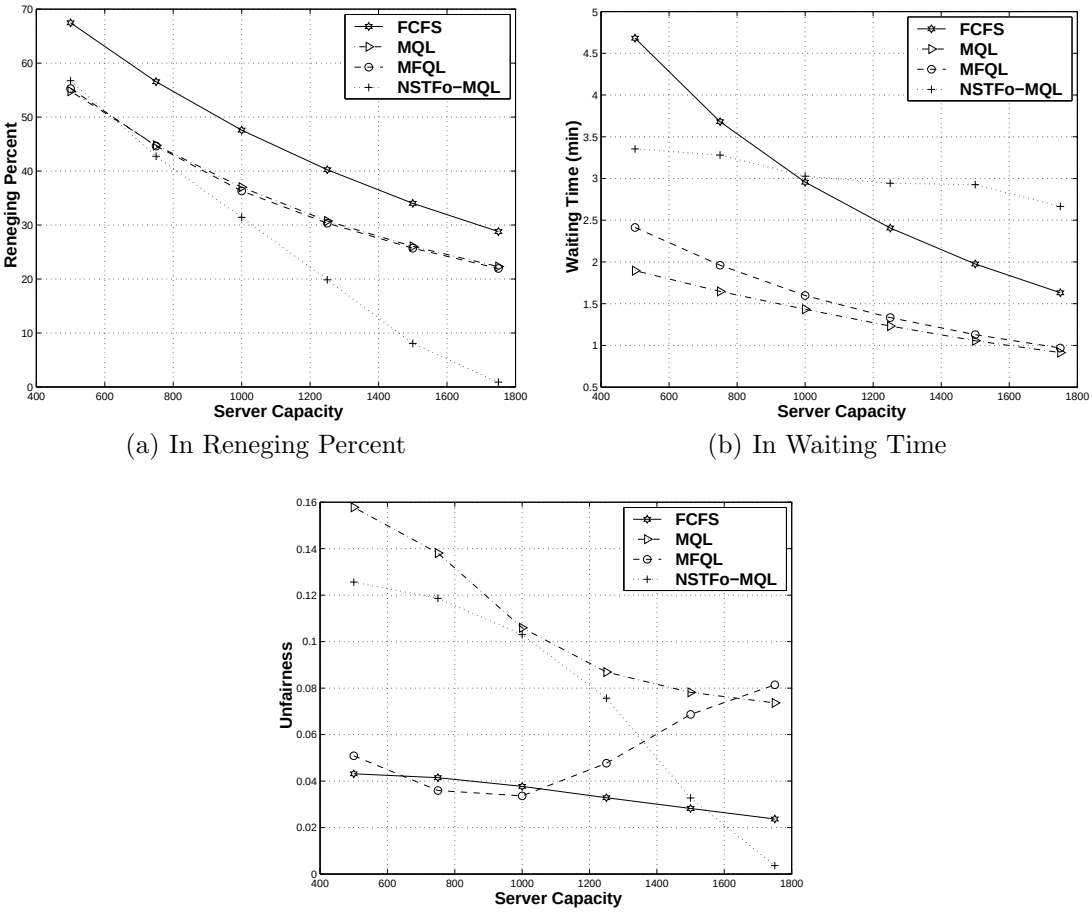
(a) In Reneging Percent

(b) In Waiting Time

(c) In Unfairness

Fig. 4.6: Comparison among FCFSg, NSTFn, and NSTFo-MQL (Model B)

value. The results demonstrate that the performance improves slightly as T increases up to a certain value, then it starts to degrade more significantly. The impact of T is more dramatic for lower server capacities (not shown in the figure). Generally, a value of about 2 minutes achieves the best overall performance.



(a) In Reneging Percent

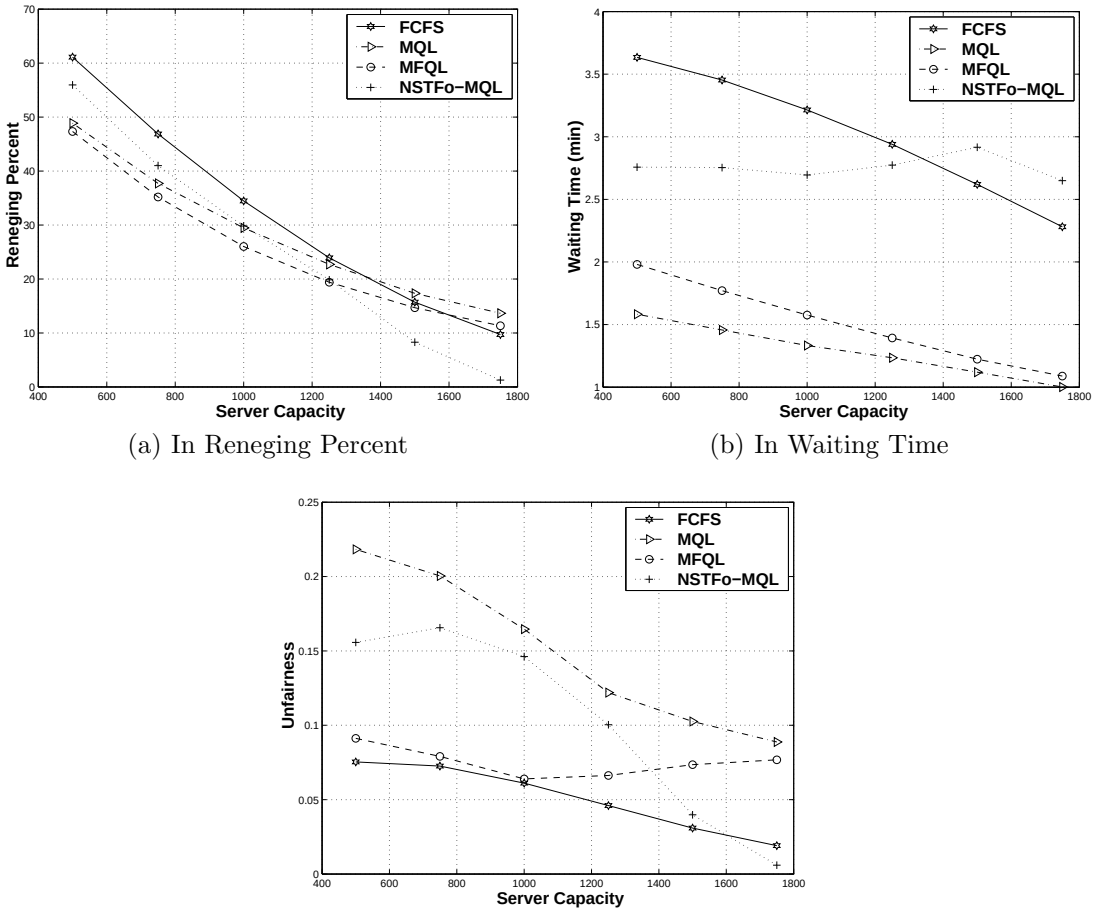
(b) In Waiting Time

(c) In Unfairness

Fig. 4.7: Comparison of NSTF0-MQL with FCFS, MQL, and MFQL (Model A)

4.5 Conclusions

This study has proposed a new class of scheduling policies for VOD servers, called *Next Schedule Time First* (NSTF), which provides customers with hard time of service guarantees and with very accurate schedule times. The study has also presented two variants of NSTF: *NSTFn* and *NSTFo*. NSTFn assigns freed schedule times to incoming requests, whereas NSTFo assigns them to existing requests. Besides, the study has shown



(a) In Reneging Percent

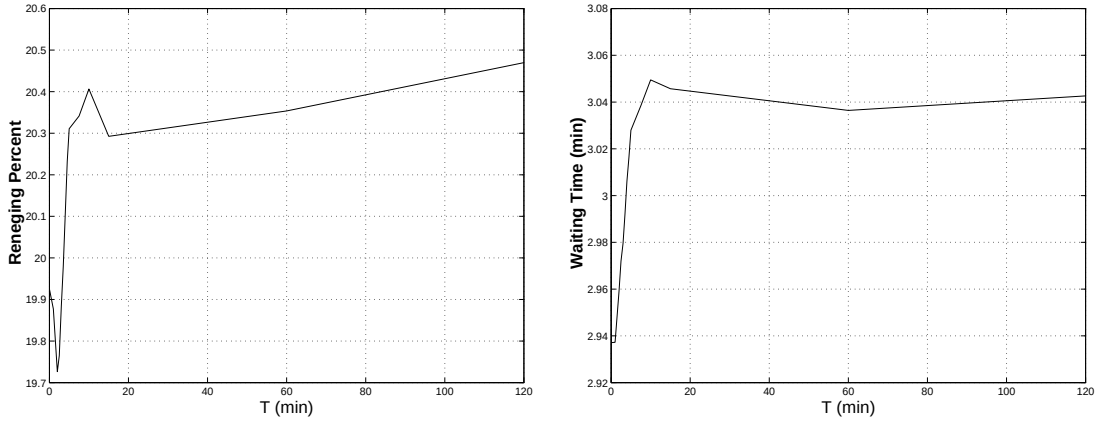
(b) In Waiting Time

(c) In Unfairness

Fig. 4.8: Comparison of NSTF0-MQL with FCFS, MQL, and MFQL (Model B)

that NSTFn and NSTFo are special cases of a policy, called *Generalized Next Schedule Time First* (GNSTF). By appropriately adjusting a threshold value, T, GNSTF can become either NSTFn or NSTFo. Furthermore, the study has presented three variants of NSTFo: *NSTFo-FCFS*, *NSTFo-MQL*, and *NSTFo-MFQL*, which differ in the selection criterion of existing requests that will be assigned better schedule times.

The effectiveness of NSTF has been demonstrated through extensive simulation. The considered performance metrics are the number of violations of time of service



(a) On Reneging Percent (b) On Waiting Time

Fig. 4.9: Effect of Threshold T (Model A, 1250 Channels)

guarantees, the average deviation of the actual times of service from the time of service guarantees, the overall customer reneging percentage, the average request waiting time, and unfairness. The impacts of customer waiting tolerance, server capacity (or server load), and the threshold T on the results are also examined.

The main simulation results can be summarized as follows.

- NSTF always meets the time of service guarantees and produces very accurate schedule times. The average deviation of the actual times of service from the schedule times are within 0.2 second (when any implementation of NSTFo is used) and 6 seconds (when NSTFn is used). In contrast, FCFS may violate its time of service guarantees, and these guarantees differ from the actual times of service by 20 seconds to more than 4.5 minutes on the average!
- NSTFo-MQL is the clear winner among the variants of NSTF when all performance metrics are considered.

- NSTFo-MQL achieves higher throughput and, in certain situations, shorter waiting times than FCFS that may provide limited time of service guarantees, even under statistically identical models of waiting tolerance.
- By motivating customers to wait, NSTFo-MQL outperforms MQL and MFQL (both of which cannot provide time of service guarantees) in terms of throughput for one of the models of waiting tolerance. For the other model, NSTFo-MQL also achieves the highest throughput but only within the most likely operating region of the server. NSTFo-MQL is also fairer than MQL and MFQL for high server capacities because schedule times are assigned on a FCFS basis. As expected, NSTFo-MQL leads to longer waiting times than MQL and MFQL.

NSTF, therefore, not only can provide hard time of service guarantees and very accurate schedule times, but also can deliver outstanding performance benefits.

Chapter 5

Analysis of Caching Strategies

This chapter presents an analytical model for interval caching and then uses the model to investigate the impacts of various parameters and to explore and evaluate new design alternatives.

5.1 Introduction

Caching of video data is a cost-effective way to reduce the disk bandwidth requirements in VOD servers. Traditional caching techniques use temporal and spatial locality of reference to bring selected objects to the cache. These techniques are well-suited for applications that consist of relatively small objects. An alternative and an efficient scheme, called *Interval Caching*, was proposed in [25] for VOD applications, which consist of relatively large objects. It has also been used later to cache continuous media contents in proxy servers [101, 5]. This scheme exploits the temporal locality of reference and the sequential access pattern by caching intervals between successive streams accessing the same contents in the main memory of the server.

Modeling of interval caching can be used as a design tool to evaluate different design choices. Many factors impact caching performance in VOD servers. These factors include the cache size, the number of videos, the video duration(s), the video compression

method, the video access distribution, the request arrival distribution, and the average request arrival rate. Simulation-based analysis of interval caching is a very time-consuming process. The time inefficiency often hinders thorough investigations of different design choices. An analytical model can come to the rescue. Deriving an accurate analytical model, however, is challenging. The authors who proposed interval caching attempted to derive an analytical model but their efforts stopped midway by concluding that the task is extremely cumbersome and difficult [25]. Later on, an analytical model was developed in [63, 65]. Unfortunately, this model requires prior knowledge of the total number of customers that can be serviced concurrently, which itself depends on the number of customers serviced from the cache.

This study develops a validated analytical model for interval caching. The model requires no prior knowledge of the total number of concurrent streams. The study then uses the model to examine the impacts of various system and workload parameters. Furthermore, it uses the model as a tool to experiment with new design alternatives. In particular, it evaluates the effectiveness of a hybrid caching scheme that caches a small set of the most popular videos in their entireties and employs interval caching for the rest. Caching whole videos was not appealing years ago because of the high prices of semiconductor memories and their highly constrained sizes. With the recent technological advances, however, these prices have fallen dramatically. Moreover, the main memory now comes in much higher capacities; the Symmetric 8000 series of EMC storage subsystems, for instance, can be shipped with up to 120 GB of cache.

The rest of this chapter is organized as follows. Section 5.2 discusses interval caching, and then Section 5.3 presents an analytical model for this technique. The

model is then used to examine the impacts of various system and workload parameters in Section 5.4 and to evaluate the effectiveness of the hybrid caching scheme in Section 5.5. Finally, the last section draws conclusions.

5.2 Interval Caching

Interval Caching attempts to pair each playback request with an immediately preceding request for the same video that is currently being serviced (either from the disks or the cache). Specifically, interval caching reuses the data brought by a stream in servicing a closely following stream if sufficient cache space exists to retain the data blocks in the cache. The two streams are called the *following* stream and the *preceding* stream, respectively. The required cache space for servicing the following stream depends on the time interval between the two streams and the compression method used. When the server accepts a request for service from the cache, it starts to cache the data of the preceding stream while retrieving and transferring them to the client. The following stream will have to be serviced from disks until the playback reaches the first cached block. The time during which the following stream is serviced from the disks is called the *catchup time*. Interval caching uses the cache (memory) space efficiently by victimizing longer intervals for caching shorter ones. This cache replacement policy is called *victimization*.

Interval caching can achieve significant performance benefits when the request arrival rate is fairly high, even with a modest cache size. The high request rate coupled with the large skewness in video access reduce the average cached interval, thereby improving the effectiveness of caching. This technique shortens the request waiting time

and enhances server throughput without increasing the bandwidth or the buffer space requirement at the client. It can also be applied in conjunction with other resource sharing techniques to improve performance further. Besides, it has become cost-effective with the falling prices of semiconductor memories.

5.3 Analytical Model for Interval Caching

The model developed here is based on the following two observations. First, the victimization process ensures that only the shortest intervals between successive streams are cached. Therefore, the length of a cached interval is bounded by an upper value, denoted as L seconds. Second, the underlying storage subsystem impacts caching performance only slightly. The reason is that interval caching gives higher precedence to servicing requests from the cache than from the disks. The effectiveness of victimization in reducing the average cached interval, however, depends on the ability of the storage subsystem to support the victimized requests directly from the disks. Fortunately, the impact of the underlying storage subsystem can be isolated with a little effect on the accuracy of the model if the storage subsystem can support by itself a relatively fair number of streams, which is typically the case. The number of streams that can be serviced from the disks is referred to here as the *number of storage channels*. Figure 5.1 demonstrates that the number of streams serviced from the cache does not change considerably with the number of storage channels after this number becomes greater or equal to 500. These results were obtained by a simulator developed by this study for interval caching and are for a server supporting 120 2-hour long videos with a skewness

parameter (θ) of 0.271 and a request arrival rate of 12 requests per minute. Similar behaviors were observed when these parameters were varied.

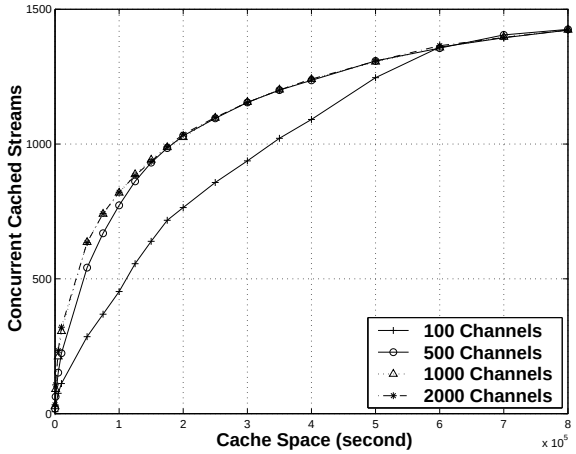


Fig. 5.1: Effect of Number of Storage Channels on Caching Performance

5.3.1 Main Idea

The number of concurrent cached streams, N, can be found by counting the number of arriving requests that fall within L-length distances from their preceding streams during one video duration, D. For this purpose, the number of cached requests for each video should be found individually. This number can be estimated through the following process.

- Divide a D-length time window into T L-length time windows, where $T = D/L$.
- Assume that the leading request in each time window is serviced from the disks or the cache. This assumption will be eliminated later.

- Find the total number of subsequent requests in each time window and use it as an initial estimate of the number of concurrent streams.

- Account for the requests that fall in adjacent time windows but are within L-length distances from each other.

Subsequently, N can be found by adding up the number of concurrent cached streams for each video. Figure 5.2 further explains the counting process. The figure shows the arrival of requests for a certain video within nine consecutive L-length time windows. A dark circle represents an arriving stream that can be cached, while a white circle represents an arriving stream that cannot be cached. Observe the effect of the overlap between time windows 5 and 6 and time windows 8 and 9.

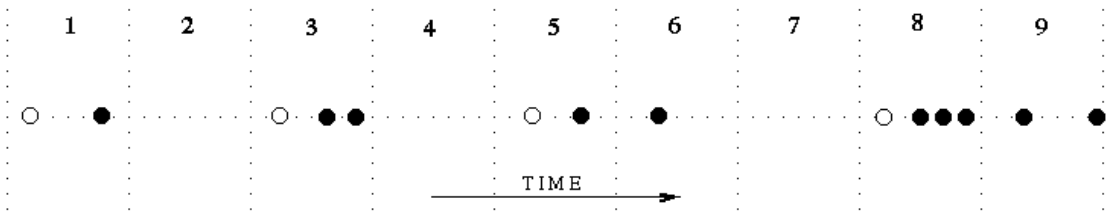


Fig. 5.2: Clarification of the Analytical Model

5.3.2 Modeling Process

Let us now discuss the entire process of the model development. Table 5.1 describes the primary parameters used in the model. Because the arrival of requests follows a Poisson Process distribution with parameter λ, the inter-arrival time to video j is exponentially distributed with a mean $1/\lambda_j$, where $\lambda_j = \lambda \times A_j$, and A_j is the probability of

Table 5.1: Main Model Parameters

Parameter Name	**Symbol**
Request Arrival Rate (Request/s)	λ
Request Arrival Rate for the j^{th} Video (Request/s)	λ_j
Request Defection Rate	dr
Request Service Probability	f
Cache Size (MB)	S
Number of Videos	N_v
Video Duration (s)	D
Video Data Rate (MB/s)	r
Probability of Selecting the j^{th} Video	A_j
Maximum Cached Interval Length (s)	L
Number of Cached Streams	N
Number of Cached Streams for the j^{th} Video	N^j

selecting video j. Therefore, the probability of k arrivals for video j within an L-length time window is

$$P_k^j = \frac{(\lambda_j L)^k}{k!} e^{-\lambda_j L}. \tag{5.1}$$

Consequently, the expected number of requests for video j during D seconds that have preceding streams in their L-length time windows, assuming exactly k arrivals in each of these time windows, can be found using the Bernoulli distribution as follows:

$$E[\hat{N}_k^j] = (k-1) \sum_{i=1}^{\lceil T \rceil} i \binom{\lceil T \rceil}{i} P_k^{j^i} (1 - P_k^j)^{\lceil T \rceil - i}. \tag{5.2}$$

In Equation 5.2, $k-1$ is the number of cached requests in each of those L-length time windows. The equation is precise when D is a multiple of L. The expected number of arriving requests for video j that fall within the same L-length time windows as their preceding streams is

$$E[\hat{N}^j] = \sum_{k=2}^{\infty} E[\hat{N}^j_k]. \tag{5.3}$$

We need now to account for the requests arriving in adjacent time windows but still falling within an L distance from each other. The number of such requests can be found using the Bernoulli distribution. Hence, the expected number of concurrent cached requests for video j is

$$E[N^j] = E[\hat{N}^j] + \sum_{i=1}^{\lceil T \rceil} i \binom{\lceil T \rceil}{i} P_2^{j\,i} (1 - P_2^j)^{\lceil T \rceil - i}, \tag{5.4}$$

and the expected number of concurrent cached streams for all videos is

$$E[N] = \sum_{j=1}^{N_v} E[N^j]. \tag{5.5}$$

In Equation 5.4, P_2 is the probability of exactly two arrivals within an L-length time window.

To find $E[N]$, we need to know L, which depends on the average cached interval. The average cached interval in seconds for video j can be calculated using the probability

density function of the corresponding Poisson process, $f_x(x)$, as follows:

$$E[I^j] = \int_0^L x f_x(x)\, dx = \lambda_j \int_0^L x e^{-\lambda_j x}\, dx = \frac{1 - (\lambda_j L + 1)e^{-\lambda_j L}}{\lambda_j}. \tag{5.6}$$

Finally, the number of concurrent cached streams can be found subject to constraint on the available cache space:

$$\sum_{j=1}^{N_v} E[I^j] \times E[N^j] \times r <= S. \tag{5.7}$$

Let us now reconsider the assumption that the leading request in each L-length time window is serviced. Instead, let us assume that the server services that request with probability f. This probability can be determined by using the average request defection rate, dr, of the server ($f = 1 - dr$). Thus, assuming that the service of a request is independent from the service of the other requests in the same L-length time window, the number of cached requests in each time window is

$$\begin{aligned} N_l &= f(k-1) + (1-f)f(k-2) + \cdots + (1-f)^{k-2} f(1) \\ &= f \sum_{j=1}^{k-1} (1-f)^{j-1}(k-j). \end{aligned} \tag{5.8}$$

Therefore, $k - 1$ in Equation 5.2 should be replaced with N_l. Because the request defection rate is an unknown output parameter, the value of f should be adjusted so that the rejection rate equals to $1 - f$. Another alternative is to substitute a reasonable expected value. Fortunately, as Figure 5.1 suggests, there is no pressing need to adjust

the value of $k-1$ in Equation 5.2, especially for not too small values of the cache size and no too large values of the request defection rates.

5.3.3 Validation

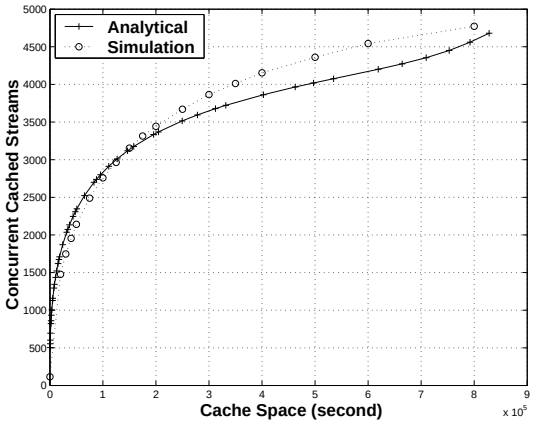
This study validates the analytical model by developing a simulator for interval caching. The validation is done for three sets of parameters shown in Table 5.2. Figure 5.3 compares the results of the analytical model and those of simulation for these three sets of parameters. The total cache space is reported in seconds in order to make the results independent of the video compression method used. The analytical and simulation models produce very close results, especially when the cache space holds fewer than 2.5×10^5 seconds, which is equivalent to about 90 GB with the MPEG-II compression standard and a data rate of 3 Megabits per second. The difference is always within 8%.

Table 5.2: Descriptions of Three Sets of Parameters Used for Validation

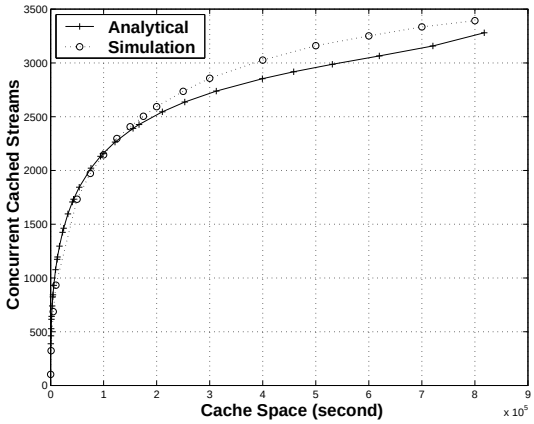
Parameter	**Set 1**	**Set 2**	**Set 3**
Request Arrival Rate (Request/min)	40	34.3	30
Number of Videos	120	150	200
Video Duration (min)	120	100	90
Video Skewness (θ)	0.271	0.15	0

5.4 Impacts of Various Parameters

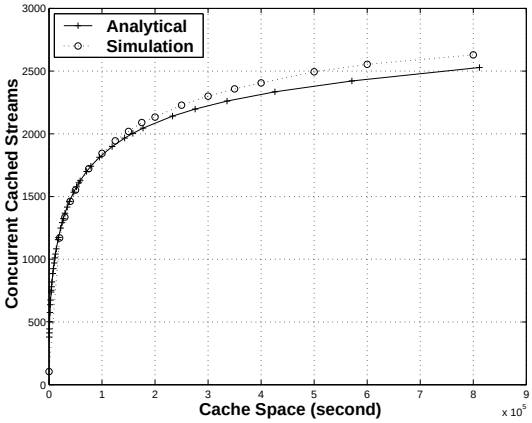
Let us now discuss the impacts on interval caching performance of the longest cached interval (L), the cache size, the skewness in video access, the request arrival rate, the video duration, and the number of videos. Table 5.3 shows the default values of the input parameters used.



(a) Input Parameter Set 1



(b) Input Parameter Set 2



(c) Input Parameter Set 3

Fig. 5.3: Validation of the Analytical Model

Table 5.3: Default Parameter Values

Parameter	**Default Value**
Request Arrival Rate (Request/min)	40
Number of Videos	120
Video Duration (min)	120
Video Skewness (θ)	0.271

Figure 5.4 shows the impacts of L on the required cache size and the number of concurrent cached streams. As expected, both these metrics increase with L.

The number of concurrent cached streams for a given cache space can be found by first finding L for that cache space and then finding the number of concurrent cached streams for that L. Figure 5.5 plots the number of concurrent cached streams versus the cache size. These results indicate that the cost-performance effectiveness of interval caching diminishes with the cache size because of the sublinear growth in the number of cached streams.

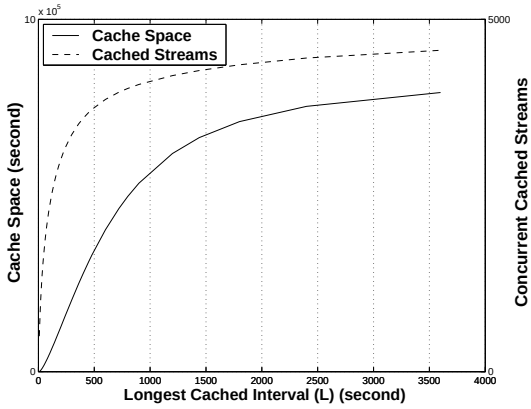


Fig. 5.4: Effect of Longest Cached Interval (L)

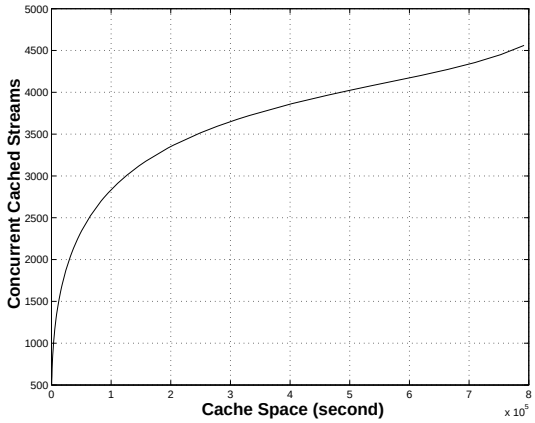


Fig. 5.5: Effect of Cache Size

Figure 5.6 demonstrates how the average request arrival rate affects the number of concurrent cached streams. Note that the number of cached streams increases with the arrival rate because requests tend to arrive closer to each other with a higher arrival rate, thereby shortening the average cached interval.

Figure 5.7 depicts the impact on the video skewness parameter (θ) on the performance of interval caching. As expected, the number of cached streams increases with the skewness because of the increased locality of reference, which improves the overall effectiveness of caching.

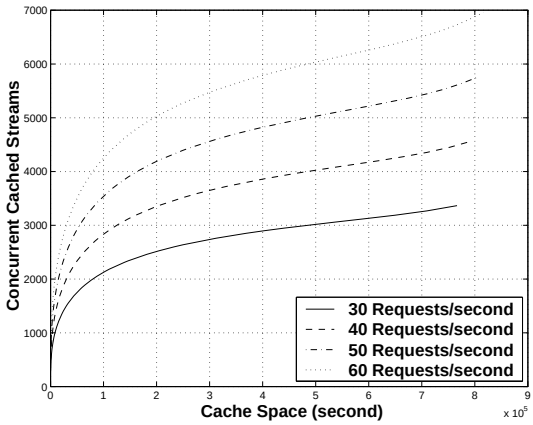


Fig. 5.6: Effect of Request Arrival Rate

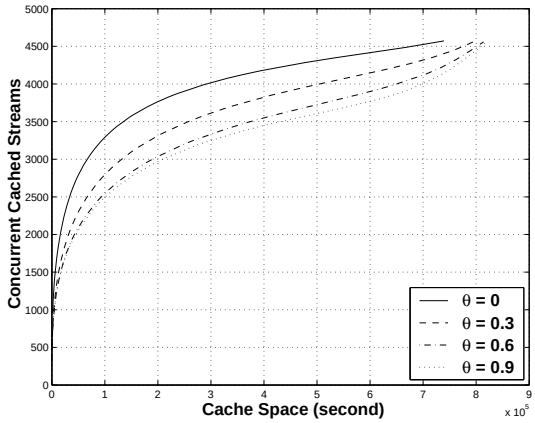


Fig. 5.7: Effect of Skewness Parameter (θ)

Figure 5.8 shows the effect of the video duration. The number of concurrent cached streams is scaled so that it corresponds to the number of cached streams during a 120-minute period, which is equivalent to the longest video duration. Note that this number increases as the video duration decreases because of the reduced data size. The video duration is expected to have a greater impact if the ability of the server to handle a higher request arrival rate when the videos are shorter is considered. The server can support more requests because shorter videos require shorter service (playback) times.

Finally, Figure 5.9 demonstrates how the number of concurrent cached streams varies with the number of videos supported by the server. These results show that the number of concurrent cached streams decreases with the number of videos. This behavior is due to the diminishing of the effective skewness in video access patterns as the number of videos increases.

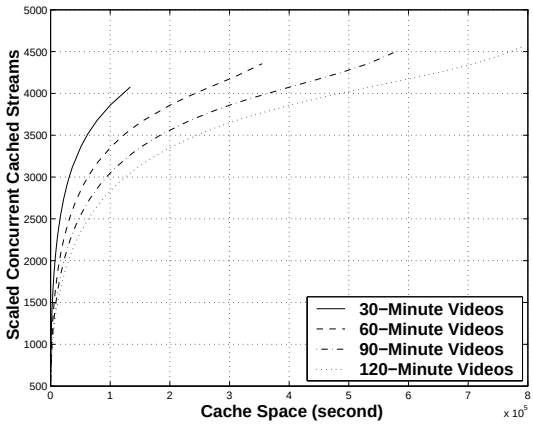


Fig. 5.8: Effect of Video Duration

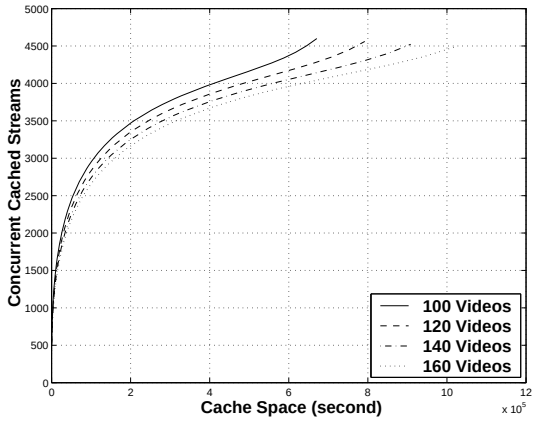


Fig. 5.9: Effect of Number of Videos

5.5 Hybrid Caching

In this section, the developed analytical model is used as a tool to experiment with new design alternatives. Namely, the effectiveness of a hybrid caching scheme is evaluated. This scheme caches the first N_h most popular videos in their entireties and employs interval caching for the rest. This scheme differs from *Generalized Interval Caching* (GIC), discussed in Subsection 2.4.3, primarily in that whole regular videos are cached in the memory rather than only short video clips (for advertisement, etc). The hybrid scheme targets servers with large cache sizes.

The analytical model presented in Section 5.3 can be used to model the hybrid scheme with only minor changes. The total number of concurrent cached streams is the sum of the number of concurrent cached streams for the first N_h most popular videos and the number of concurrent cached streams for the rest of the videos:

$$E[N] = \sum_{j=1}^{N_h} \lambda_j \times D + \sum_{j=N_h+1}^{N_v} E[N^j]. \tag{5.9}$$

This number can be found subject to the constraint on the available cache space:

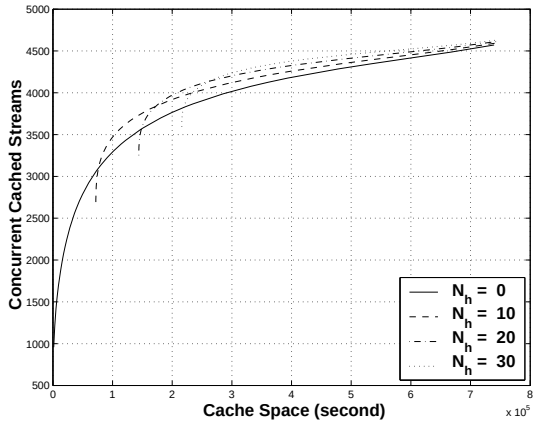
$$N_h \times D \times r + \sum_{j=N_h+1}^{N_v} E[I^j] \times E[N^j] \times r <= S. \tag{5.10}$$

Next, the main results are presented and analyzed for the cases where videos have equal and different durations.

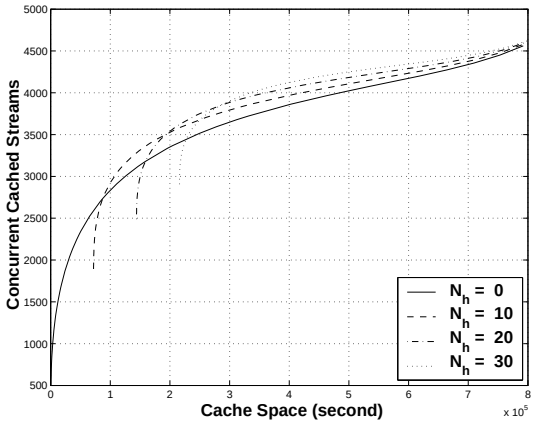
5.5.1 Analysis with Equal Video Durations

Let us now discuss the impact of N_h on the performance of the hybrid caching scheme. The default values of the input parameters used in this subsection are the same to those in Section 5.3 and shown in Table 5.3.

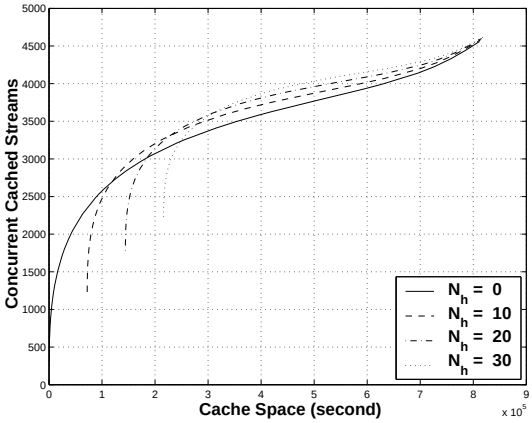
Figure 5.10 demonstrates the impact of N_h on the number concurrent cached streams for three values of the skewness parameter (θ): 0, 0.271, and 0.542. Note that when N_h is zero, the hybrid scheme is reduced to interval caching. These results show that a larger value of N_h requires an initially larger cache size and produces an initially smaller number of cached streams, but this number soon becomes larger than that with a smaller value of N_h when the cache size increases further. The hybrid scheme with a larger value of N_h starts to outperform the ones with smaller values sooner when the skewness in video access is higher. In conclusion, the hybrid scheme achieves better than interval caching for sufficiently large caches, and N_h should be set to the largest value that the cache can accommodate while having moderate additional space.



(a) Skewness Parameter (θ): 0



(b) Skewness Parameter (θ): 0.271



(c) Skewness Parameter (θ): 0.542

Fig. 5.10: Effect of Number of Videos Entirely Cached (N_h) on Caching Performance

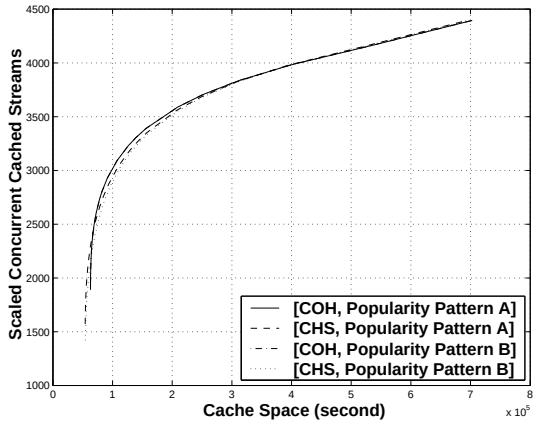
5.5.2 Analysis with Varying Video Durations

For simplicity, let us assume that the server supports videos with only two different durations: $D1$ and $D2$. The set of videos with the shorter duration is called the *short set*, while the other is called the *long set*. (Extending the analytical model to handle varying video durations is straightforward and thus is not discussed here.) Let us also assume that the video popularity follows one or the other of the following two patterns: *Pattern A* and *Pattern B*. In Pattern A, the most popular video is from the short set,

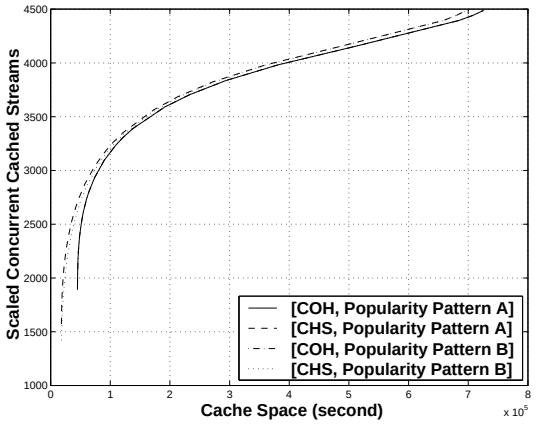
the second most popular video is from the long set, the third most popular is from the short set, and so on. Pattern B is similar except that the pattern starts with a video from the long set. This study considers two alternatives of the hybrid scheme: *Cache Only Hottest* (COH) and *Cache Hottest and Shortest* (CHS). COH caches the first N_h most popular videos, whereas CHS caches the first N_h most popular videos of the short video set. Both these schemes employ interval caching for the rest of the videos.

Figure 5.11 compares the two alternatives of the hybrid caching scheme for the two popularity patterns and for three different combinations of values of video durations and N_h. The number of concurrent cached streams is scaled so that it corresponds to the number of cached streams during the longer video duration. In general, the two alternatives perform very closely to each other, and the video popularity pattern has a nearly negligible impact on the results. The little difference in performance between the two alternatives is because the advantage of CHS in requiring smaller cache space for the videos cached entirely seems to be counterbalanced by the advantage of COH in caching entirely only the globally most popular videos, which will receive most of the accesses. The difference, however, increases slightly with N_h and with the difference in video durations. Next, the analysis is limited to CHS and video popularity Pattern B.

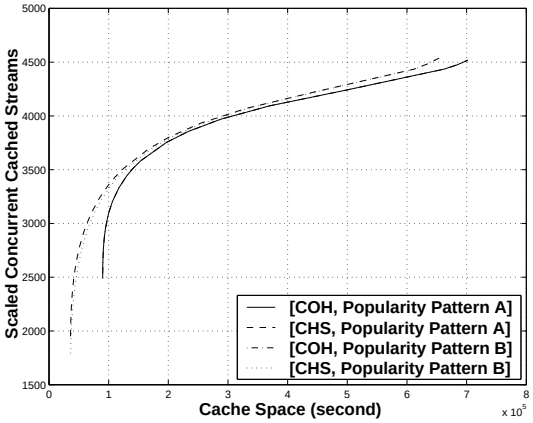
Figure 5.12 demonstrates the impact of N_h on the performance of the hybrid caching scheme for three combinations of video durations. These results exhibit a similar behavior to those in the case of equal video durations. Note that the benefits of larger values of N_h are greater when the difference in video durations is bigger.



(a) $D1 = 120$ min, $D2 = 90$ min, $N_h = 10$



(b) $D1 = 120$ min, $D2 = 30$ min, $N_h = 10$

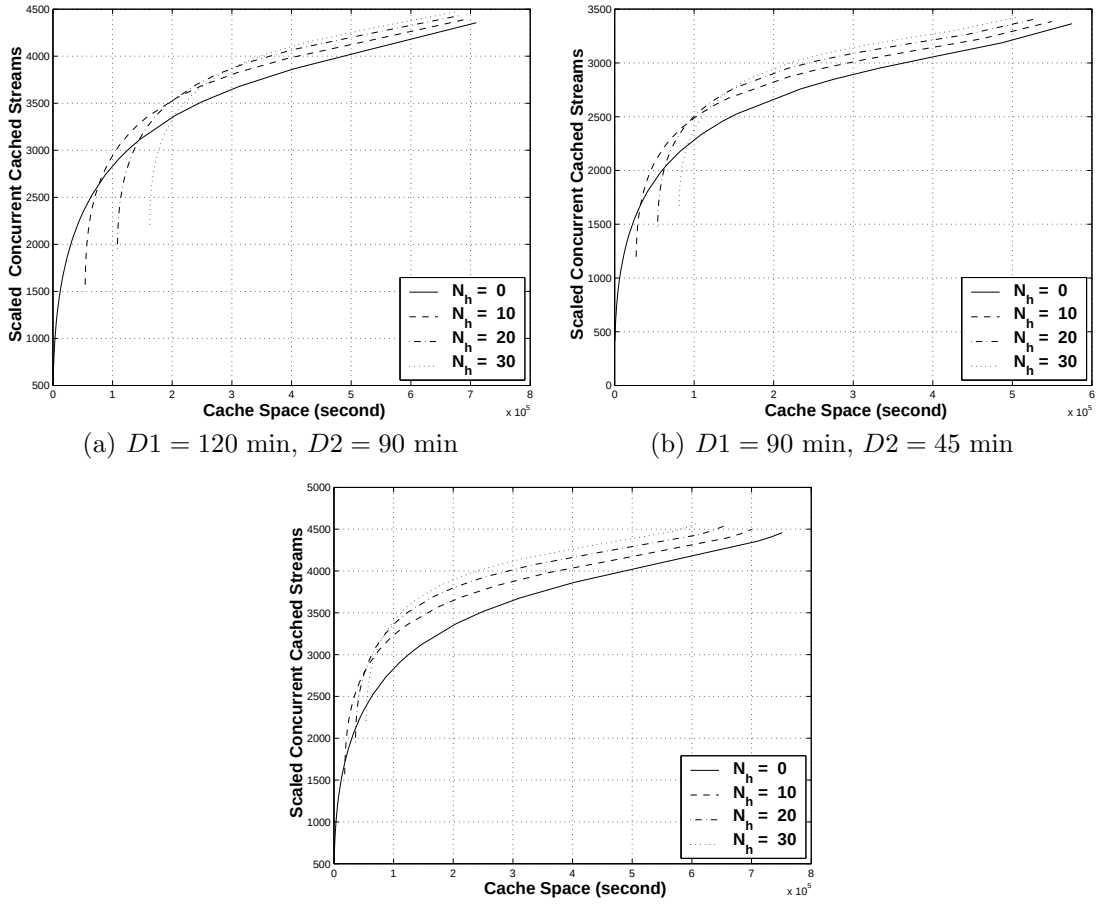


(c) $D1 = 120$ min, $D2 = 30$ min, $N_h = 20$

Fig. 5.11: Comparison between the Two Alternatives of Hybrid Caching

5.6 Conclusions

Interval Caching is an efficient caching scheme for VOD servers. This scheme exploits the temporal locality of reference and the sequential access pattern by caching intervals between successive streams accessing the same contents. Simulation-based analysis of interval caching is a very time-consuming process. The time inefficiency often hinders thorough investigations of different design choices. An analytical model removes this time inefficiency, but deriving such a model is a challenging problem.



(a) $D1 = 120$ min, $D2 = 90$ min

(b) $D1 = 90$ min, $D2 = 45$ min

(c) $D1 = 120$ min, $D2 = 30$ min

Fig. 5.12: Effect of Number of Videos Cached Entirely (N_h) on Hybrid Caching (CHS Alternative, Popularity Pattern B)

This study has developed an analytical model for interval caching and has validated it by simulation. Moreover, the study has used the model to examine the impacts of various system and workload parameters. Furthermore, it has used the model as a tool to experiment with new design alternatives. In particular, it has evaluated the effectiveness of a hybrid caching scheme that caches a small set of the most popular videos in their entireties and employs interval caching for the rest. The results show that the hybrid scheme outperforms interval caching when the cache is sufficiently large.

The number of videos cached entirely should be set to the largest value that the cache can accommodate while having moderate additional space for caching intervals between successive streams for the rest of the videos.

Chapter 6

Design of Multimedia Servers Based on Network-Attached Disks

This chapter uses the *Network-Attached Disk* (NAD) architecture for designing highly scalable and cost-effective multimedia servers. It also presents an integrated resource sharing policy that improves the performance of these servers through caching and intelligent scheduling. It also demonstrates the effectiveness of this policy by extensive simulation and evaluates the performance limit of caching in these servers by analytical modeling.

6.1 Introduction

Current multimedia servers, such as Tiger Shark [54] and Tiger Video [11], are based on the conventional fileserver architecture. This architecture is also known as the *Server-Attached Disk Architecture* and is shown in Figure 6.1. In this architecture, all the data flowing between the disks and the clients pass through the server. The required *store-and-forward* copying imposes unnecessary burdens on these servers and severely limits their scalability.

Recently, the *Network-Attached Disk* (NAD) architecture [42, 43, 44, 45] has emerged as a cost-effective solution to design scalable storage servers. In this architecture, the disks are connected to the network as shown in Figure 6.2. Therefore, the

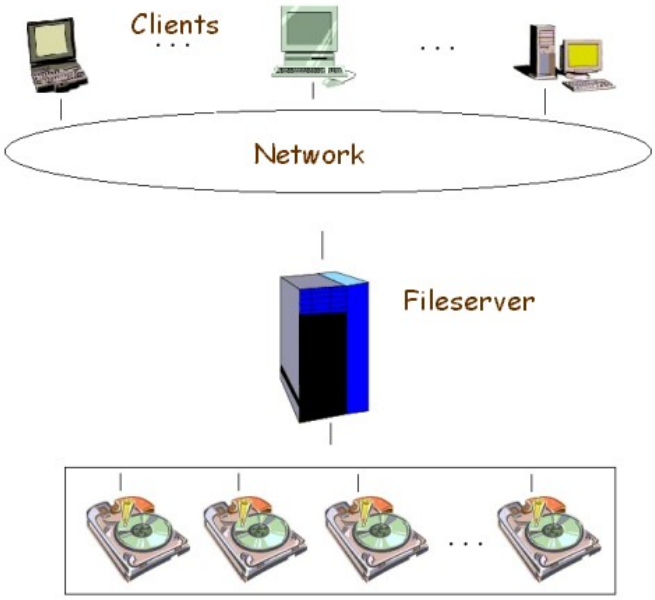


Fig. 6.1: The Server-Attached Disk Architecture

data can be transferred directly between the disks and the clients. The NAD architecture offers four major advantages over the conventional architecture.

- It scales inherently with storage capacity by removing the server as a bottleneck. Hence, adding a new disk not only increases storage capacity, but also throughput and computing power.
- It reduces cost by eliminating the resources used for store-and-forward copying.
- It enhances performance with network striping and shorter data latency.
- It helps to utilize the available on-disk computing powers and caches (buffer) effectively.

The scalability and the performance issues of the NAD architecture are discussed in [45, 71]. This architecture requires efficient security protocols to ensure data confidentiality

and integrity as the disks are connected directly to the network. The security problem is addressed in [47, 81, 72].

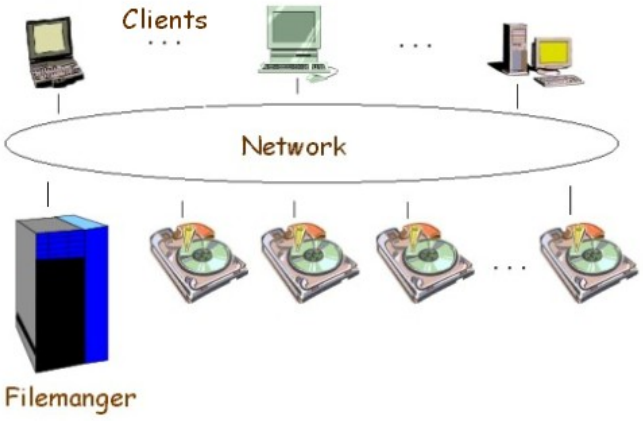


Fig. 6.2: The Network-Attached Disk Architecture

This study uses the NAD architecture to design highly scalable and cost-effective multimedia servers and ensures high performance by proposing an *Integrated Resource Sharing* policy. The integrated policy meets the following objectives.

- Increase the number of serviced customers while reducing their waiting times for service.
- Use server and network resources efficiently.
- Do not expect much bandwidth and buffer space at the client.
- Provide high configurability: the policy should be tunable so as to make any necessary tradeoff (including the number of concurrent customers, waiting times, fairness, and implementation complexity) and to fit the current server workload, which

may change with time (especially from daytime to nighttime and from business days to weekends).

- Work well for large servers.

The integrated resource sharing policy employs both caching and intelligent scheduling to enhance the performance of NAD-based multimedia servers. For caching, this study proposes a scheme, called *Distributed-Interval Caching* (DIC), which extends interval caching and adapts it to the NAD architecture. Modern hard disks have large internal caches and embedded processors, and these on-disk assets are likely to grow at a faster rate because of the scaling of technology and the widening market for NADs. DIC utilizes these caches for caching intervals between successive streams. The required low-level control of caches can be entertained by the ever-growing on-disk intelligence. For scheduling, this study proposes a scheme, called *Multi-Objective Scheduling* (MOS), which increases the degrees of resource sharing by selecting requests for service based on four predefined criteria.

A further contribution of this work is studying VOD servers at the disk level and considering many design aspects, including admission control, batching, scheduling, and caching. Analyzing the integrated policy, rather than only the individual schemes, helps in understanding various tradeoffs, making appropriate design decisions, and ensuring that global optimizations can be attained.

The effectiveness of the integrated policy and the interactions among various system and workload parameters are demonstrated through extensive simulation. The

primarily considered performance metrics are the average number of requests that can be serviced concurrently and the average request waiting time.

Finally, the performance limit of DIC is evaluated by analytical modeling. The limit is then used to determine whether a further investigation is required to push the performance envelope.

The rest of the chapter is organized as follows. Section 6.2 presents the DIC scheme. Section 6.3 presents the MOS scheme. Section 6.4 presents the integrated policy. Section 6.5 discusses the performance evaluation. Finally, the last section draws conclusions.

6.2 Distributed Interval Caching (DIC)

The DIC scheme proposed here extends interval caching (discussed in Section 5.2) and adapts it to the NAD architecture. With interval caching, data are cached in the main memory of the fileserver. In the NAD architecture, the disks are connected to the network as well as the fileserver (filemanager). Thus, using interval caching in a NAD-based server involves extra network activity in the process of bringing the *to-be-cached* data from the disks to the filemanager. More importantly, it negates the whole purpose of the NAD architecture (i.e., eliminating the fileserver as a bottleneck). In the proposed DIC scheme, however, an interval between two streams is cached concurrently in multiple disks using each disk's internal cache. Because of the distributed environment, the scheme has to deal with the arising complications in admission and victimization.

Before discussing the DIC scheme, let us discuss briefly how a typical multimedia server operates. In a multimedia server, videos are striped across all or a collection of

disks. The server handles multiple clients by proceeding in periodic rounds and retrieving a fixed duration of media from one of these disks during each round. This fixed duration of media is called *interleaving unit* (IU). For simplicity, let us assume that videos are striped across all N_d disks in a round-robin (RR) fashion. A mapping function, given by $Disk\_map(m) = m \ mod \ N_d$, is used to map the first IU of video m to a disk. This mapping helps to enhance the cache utilization and to balance the I/O load among all the disks.

The DIC scheme works as follows. First, it searches for a preceding stream for a new request. The scheme accepts the request for service from the on-disk caches if the request has a preceding stream, the corresponding interval can be cached in appropriate disks, and the new request can be serviced from disks during catchup. If no sufficient cache space exists, then it tries to victimize a longer interval. Finally, if no victimization can be applied or the request has no preceding stream, then it resorts (if possible) to servicing the request directly from the disks (and not from their caches).

Let us now examine what happens after a request is accepted for service from the on-disk caches. Let us assume that during round i, a new request calls for the playback of a video whose first IU is stored on disk f_disk, and that disk p_disk will service the corresponding preceding stream in the next round. If the server accepts this request for service from the on-disk caches, then in round $i + 1$, disk p_disk caches the current IU of the preceding stream, while it retrieves and transfers it to the client. Meanwhile, disk f_disk services the new request from its media (not cache). In the subsequent rounds, the next disks in the striping sequence will be involved in the process. After catchup,

the following stream will be serviced from the on-disk caches until the completion of the video.

Figure 6.3 further explains the scheme for an 8-disk system. It shows what happens after *S2* (the following stream of video 2) is paired up with *S1* (the preceding stream of the same video), which came three rounds earlier than *S2* and is being serviced from disks. Here, f_disk is 2, p_disk is 5, the time interval length is $3u$, and the cached interval length (after catchup) is $(3+1) \times u = 4u$, where u is the size of the IU in seconds. The figure shows the process during the first five rounds in the service life of *S2*. The shading shows where the currently cached data are stored. The c or the d below an arrow shows whether the stream is being serviced from a cache or a disk, respectively. Note that *S2* is serviced for three rounds from disks (during catchup) and that a wrap-around occurs as *S1* proceeds to round 5.

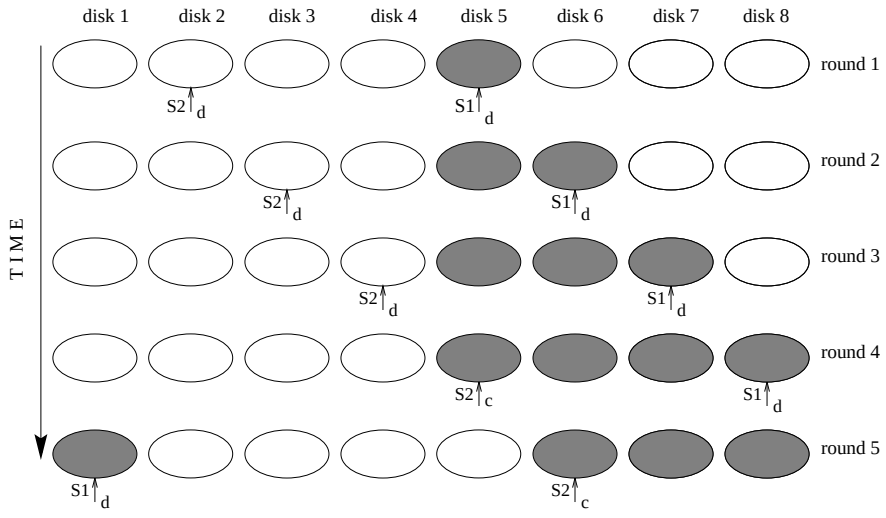


Fig. 6.3: Clarification of Distributed Interval Caching ($S1$: the preceding stream, $S2$: the following stream, f_disk: 2, p_disk: 5)

Next, the admission control and victimization issues are discussed, and then the DIC algorithm is presented.

6.2.1 Admission Control

Striping videos in a round-robin fashion simplifies admission control. With the RR scheme, all the requests serviced by a disk in a round will be serviced by the next disk in the next round. Hence, a request can be serviced from the disks if there is sufficient I/O bandwidth in disk f_disk during the first round. This condition guarantees sufficient I/O bandwidth for the request in all subsequent rounds.

Deciding whether a request can be serviced from the on-disk caches is somewhat tricky because an interval may be cached concurrently in multiple disks. In this case, the server has to ensure that sufficient cache space exists in each disk between disk f_disk and disk p_disk, inclusively during the first round. (Specifically, these disks are f_disk, $(f\_disk + 1)$ mod N, $(f\_disk + 2)$ mod N, ..., and p_disk.) Note that in Figure 6.3, the availability of sufficient cache space in disks 2 through 5 in round 1 guarantees that the interval between the two streams can be cached in all subsequent rounds. (With the round robin scheme, if there is sufficient cache space in disks 2 through 5 in round 1, then there will be sufficient cache space in disks 3 through disk 6 in round 2, sufficient cache space in disk 4 through disk 7 in round 2, and so on.) In determining the required cache space of each of these disks, we must account for long intervals that may require storing more than one IU on each of these disks. Thus, caching can only be applied if sufficient cache space exists to store $\lceil \frac{I+u}{N_d \times u} \rceil \times u$ seconds in each disk between and

including disk f_disk and disk p_disk, where u is the size of the IU, and I is the time interval between the two streams, both in seconds.

6.2.2 Victimization

When no sufficient cache space exists for servicing a new request from the on-disk caches, the server can victimize an existing request from the caches only if all the following four conditions are met.

1. The potential victim can be serviced from the disks.
2. The new request can be serviced from appropriate disks during catchup time.
3. The victimization can create sufficient cache space for the new request.
4. The interval length of the new request is shorter than that of the potential victim.

Victimizations ensure efficient usage of the available cache space at the expense of increasing the implementation overhead. This overhead, however, is an important issue in the NAD environment because of the relatively modest on-disk computing powers. Therefore, a parameter, called *Victimization Ratio* (VR), is introduced, and the last condition is changed to that the interval length required by the new request is shorter than VR times the interval length required by the potential victim. VR, which can take any value between zero and one, inclusive, should be tuned carefully in order to avoid victimizations that may not lead to considerable performance benefits.

6.2.3 DIC Algorithm

Figure 6.4 presents the algorithm in a C++-like syntax. The algorithm assumes that the disks have sufficient network bandwidth to handle their I/O rates. Note that a request can be admitted only if it can be serviced from the disks during its entire service time or only during catchup if this request can be serviced from the on-disk caches afterwards. Also note that all cached intervals are inserted into a priority queue (in an appropriate format) and placed in descending order of interval length. This queue is called the *Cached-Interval Queue* (CIQ) and is used to determine whether a victimization can be applied. The intervals in CIQ are examined from top to bottom for a possible victimization. The search terminates successfully when the four conditions are met or unsuccessfully when the time interval of the new request is greater than $VR \times l$, where l is the time interval of the currently examined cached interval.

```
01 for (each waiting request){
02   set m to the request's video number;
03   f_disk = Disk_map(m); // map a video to a starting disk
04   // Check for available disk bandwidth
05   if (no request can be serviced from disk f_disk)
06    return;
07  // Try to service the request from the caches
08  Search for a preceding stream;
09  if (there is a preceding stream) {
10    Set I to the length of the interval in seconds between the request and its preceding
      stream;
11     p_disk = (f_disk + I/u) mod N_d;
12    if ( ⌈(I+u)/(N_d×u)⌉ × u seconds can be cached in each disk between disks f_disk and
         p_disk, inclusive){
13      Accept the waiting request for service from the caches;
14      Insert the cached interval into CIQ;
15      return;
16    } // end of if
17    // Try to victimize
18    while(there are more elements to examine in CIQ){
19      Get the next interval, Im, from CIQ;
20      Set l to the length of the time interval of Im in seconds;
21      Set a_victim to the disk that will service the potential victim in the next round;
22      if(I > VR × l )
23        break; // stop trying to victimize
24      if (the potential victim can be serviced from a_victim in the next round and
           victimization can create sufficient cache space for the new request){
25        Victimize the request corresponding to Im;
26        Remove Im from CIQ;
27        Accept the waiting request for service from the caches;
28        Insert the new cached interval into CIQ;
29        return;
30      } // end of if
31    } // end of while
32  } // end of if
33  Accept the waiting request for service from the disks;
34 } //end of for
```

Fig. 6.4: The Distributed Interval Caching Algorithm

6.3 Multi-Objective Scheduling (MOS)

A VOD server maintains a waiting queue for each video and services all requests in a selected queue together using only one stream, whenever the necessary resources become available. The main objectives of scheduling are increasing the number of customers that can be serviced concurrently and decreasing their waiting times for service.

The proposed *Multi-Objective Scheduling* (MOS) scheme combines the advantages of the following four scheduling criteria.

- **Maximum Queue Length** (MQL) [26] - This criterion selects the longest queue and can be captured by the optimization function $F_b(i) = B_i$, where i is the video number, and B_i is the number of waiting requests for video i.

- **First Come First Serve** (FCFS) [26] - This criterion selects the queue with the oldest request and can be captured by $F_t(i) = T_i$, where T_i is the longest request waiting time for video i.

- **Shortest Interval First** (SIF) - This criterion selects the queue that requires the shortest cached interval and can be captured by $F_n(i) = 1/I_i$, where I_i is the required interval to cache the requests for video i. This criterion is important in the NAD environment because of the highly constrained victimization. So, it is important to select the shortest intervals during scheduling rather than to rely entirely on subsequent victimizations as done in traditional interval caching.

- **Most Popular First** (MPF) - This criterion selects the queue of the most popular video among all non-empty queues and can be captured by $F_m(i) = 1/i$. (Videos

are numbered in decreasing order of popularity.) It favors popular videos, whose requests have a higher tendency to accumulate in the waiting queues and are likely to have closer (in time) preceding and following streams that can be paired up with using a smaller cache space.

SIF and MPF are new criteria proposed in this thesis.

The MOS scheme applies the optimization function $F(i)$, which is a weighted sum of the normalized $F_b(i)$, $F_t(i)$, $F_n(i)$, and $F_m(i)$:

$$F(i) = w_b \frac{B_i}{\max B_j} + w_t \frac{T_i}{\max T_j} + w_n \frac{\min I_j}{I_i} + w_m \frac{\min j}{i},$$

where w_b, w_t, w_n, and w_m are the weights assigned for the above criteria, respectively, and $\min j$ is the lowest video number among those with waiting requests. The weights should be tuned carefully to achieve the desired tradeoff. These weights are referred to as the *scheduling parameters* in the subsequent analysis.

6.4 The Integrated Resource Sharing Policy

The integrated policy combines the DIC and the MOS schemes and is implemented by maintaining a priority queue, called the *Scheduling Queue* (SQ). Figure 6.5 shows the overall architecture and how the integrated policy works during one round. $Q1$, $Q2$,..., and Qm are the video waiting queues, and *CIQ* is the *Cached-Interval Queue*. CIQ stores all currently cached intervals in descending order of interval length. The intervals in CIQ are examined from top to bottom for a possible victimization. The *merger* combines the requests of each nonempty waiting queue i into one aggregate request and inserts it

into SQ according to $F(i)$. The *controller* schedules requests in decreasing order of F, updates the CIQ, and implements the rest of the integrated policy. The dashed lines show how a client requests the playback of video 2, and how the controller translates that into commands to a disk. The dash-dot-dot line shows the direct data path between the client and the disk.

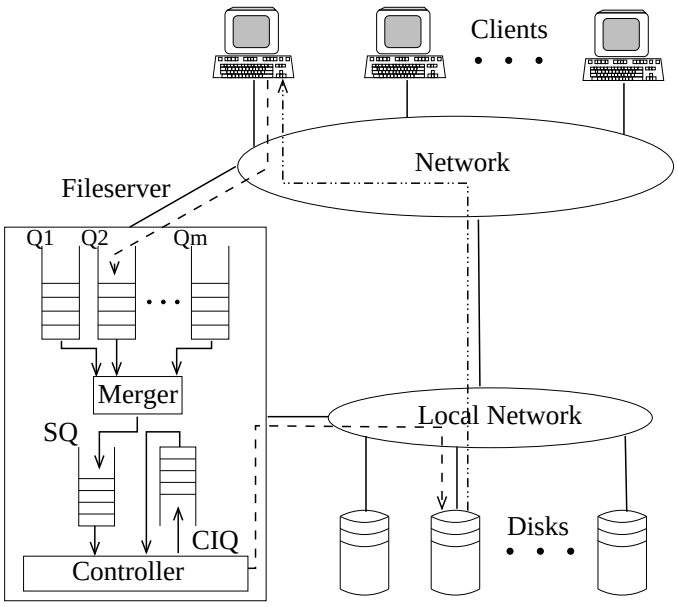


Fig. 6.5: Clarifications of the Overall Architecture and the Integrated Policy

The integrated policy enhances the DIC algorithm by incorporating two parameters: *Request Check Ratio* (RCR) and *Delay Service Ratio* (DSR). These parameters should be carefully tuned. RCR specifies the fraction of the requests in SQ that will be considered for scheduling. A small value of RCR reduces the implementation overhead and may lead to the selection of only the "best" requests, which are likely to boost performance. It, however, lengthens the delay in servicing some requests, which in turn may lead to lengthening the average cached interval and increasing the customer defection

rate. The idea behind DSR is to delay servicing from disks the requests that cannot be cached currently with the hope that they can be cached in subsequent rounds. DSR is the maximum ratio of the request waiting time to the *maximum waiting time* (MWT). If the ratio of the request waiting time to MWT is less than DSR, then the server delays the request further. A large value of DSR improves both batching and caching, but it increases the average request waiting time.

6.5 Performance Evaluation

This section discusses the effectiveness of the proposed techniques through extensive simulation and studies the performance limit of DIC through analytical modeling. In the target server, the disks are connected directly to the network, and the on-disk caches (buffers) are used for caching intervals between successive streams.

6.5.1 Workload Characteristics

In accordance with prior work, the evaluation study here assumes that the arrivals of the requests to multimedia servers follow a Poisson Process with an average arrival rate λ. Hence, the inter-arrival time is exponentially distributed with a mean $T = 1/\lambda$. It also assumes that videos are stored using the MPEG-II compression standard, and that the accesses to videos follow Zipf-like distribution [18]. Moreover, it assumes that customers can wait until a predefined maximum time (MWT). Furthermore, it assumes that videos are striped across all disks on 1/2-second intervals [19]. This assumption, however, can be relaxed to stripe across any number of disks. The interval size should be chosen based on a reasonable tradeoff. A study for determining interval size for

multimedia file servers was presented in [98]. To keep the discussion focused, VCR-like operations are not considered, and the network bandwidth for each disk is assumed to be sufficient for supporting the disk transfer rate. VCR-like operations can be supported by allocating contingency channels [28].

6.5.2 Simulation Platform

The evaluation study here was conducted by developing a simulator for NAD-based multimedia servers. The simulator takes numerous workload, system, and policy parameters in addition to three disk performance parameters: mean seek time, mean rotational latency, and mean data transfer rate. For simplicity, the simulator does not consider the variability in disk access times. This study experimented primarily with *Quantum Atlas III* disks. Table 6.1 shows its main parameters. The study used *DiskSim* [37] to find the disk performance parameters required by the developed simulator. In this process, DiskSim was fed with synthetic workloads (described in the previous subsection) and with validated disk parameters obtained from [38]. To ensure accurate simulation, this study used actual measurements to find the seek time for each seek distance. Table 6.2 shows the main disk performance parameters, which are found by simulation.

Table 6.1: Main Disk Parameters

Parameter	**Value**
Storage Capacity	9.1 GB
Rotation Speed	7200 rpm
Number of Cylinders	8,057
Number of Surfaces	10
Full Strobe Seek Time	15.36 ms
Head Switch Time	0.999 ms
Internal Transfer Rate	110 to 180 Megabits/s

Table 6.2: Estimated Values of Disk Performance Parameters

Parameter	Value
Mean Seek Time	7.778 ms
Mean Rotational Time	2.431 ms
Mean Transfer Rate	4.217 MB/s

6.5.3 Evaluation Methodology

The primary objective functions analyzed here are the number of requests that can be serviced concurrently and the average request waiting time. The integrated policy, however, can also be used to optimize other performance metrics.

Table 6.3 shows the main input parameters and their default values. The number of disks in the server is varied from 30 to 240, and the number of videos is varied accordingly to keep all disks nearly full. Only a small range of the disk cache size (0 - 40 MB) is investigated because of the limited disk power budget [62]. The request arrival rate, λ, is usually selected such that the customer defection (reneging or turnaway) rate is about 10% when the cache size per disk is 30 MB.

To evaluate the effectiveness of the integrated policy, two cases are considered: *Caching is Off* and *Caching is On*. In the first case, caching is disabled and the two optimizations in the integrated policy regarding RCR and DSR are not utilized (effectively $RCR = 1$ and $DSR = 0$), but both batching and scheduling are employed, and the values of the other parameters are kept the same to those when caching is turned on to ensure a fair comparison.

The contribution of the MOS scheme is evaluated when caching is on by varying only the values of the scheduling parameters. MFQL is not considered in this study

Table 6.3: Main Input Parameter and Default Values

Parameter	Default Value
Number of Disks	120,
Cache Size per Disk	30 MB
Number of Videos	480
Video Data Rate	3 Megabits/s
Video Duration	90 minute
Request Arrival Rate (λ)	0.435
Skewness Parameter (θ)	0
Maximum Request Waiting Time (MWT)	3 minute
Victimization Ratio (VR)	0.7
Request Check Ratio (RCR)	0.5
Delay Service Ration (DSR)	0.1
Scheduling Parameter w_b	0
Scheduling Parameter w_n	0
Scheduling Parameter w_m	0
Scheduling Parameter w_t	1

because MFQL serves as a compromise between FCFS and MQL [2], while this study aims at developing a scheme that can outperform both FCFS and MQL. Similarly, GGSC is not considered because it does not perform well in high-end servers [103].

6.5.4 Simulation Results

The following simulation results were obtained using a steady state analysis with 95% confidence interval.

6.5.4.1 Effectiveness of the Integrated Policy

Let us now examine the effectiveness of the integrated policy and discuss the impacts of the cache size per disk, the number of disks, and the request arrival rate. Figure 6.6 plots the number of concurrent requests that can be serviced versus the cache size per disk and the number of disks. In this figure (and all others), caching is turned off when the cache size is 0. As expected, the number of concurrent requests increases

with the number of disks and the cache size, but the number of disks plays a more significant role. Next, the number of disks is fixed (at 120), and the impact of the cache size per disk is analyzed in terms of the number of concurrent requests and the customer defection rate. Figure 6.7 shows that both these metrics exhibit a faster growth/decay when the cache size is small. This happens primarily because the integrated policy improves performance through both batching and caching, but the cache size does not contribute to batching. Figure 6.8 depicts the impact of the number of disks while the cache size per disk is kept constant (30 MB). Note that the benefit of the integrated policy generally increases with the number of disks. This behavior is due to the increase in λ that the server can handle. Increasing λ shortens the intervals between successive streams, thereby allowing the server to service more requests from the on-disk caches. In addition, as λ increases, batching improves since more requests accumulate in the waiting queues.

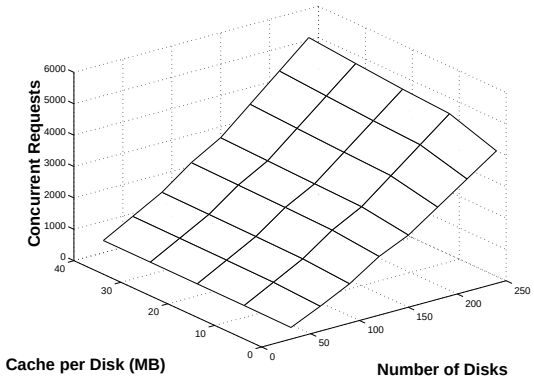


Fig. 6.6: Effect of Number of Disks and Cache Size

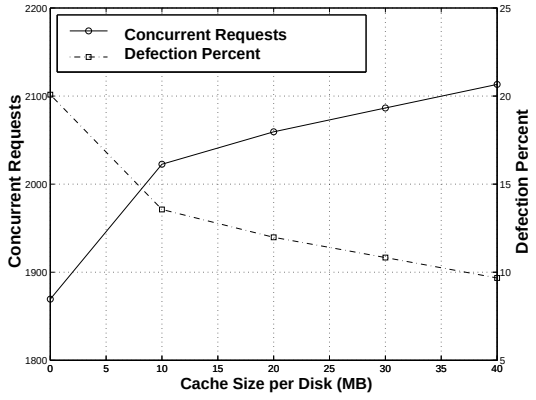


Fig. 6.7: Effect of Cache Size

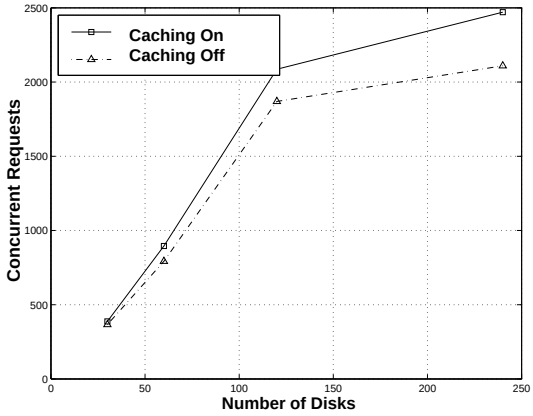


Fig. 6.8: Effect of Number of Disks

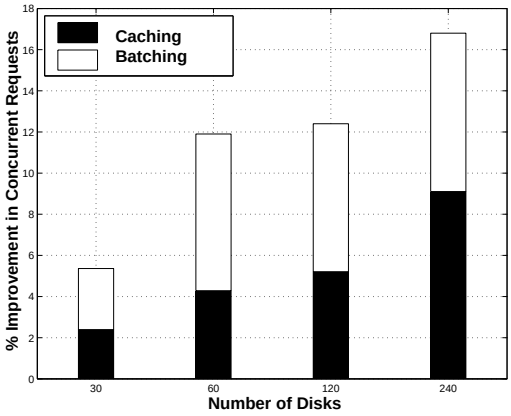


Fig. 6.9: Contributions to Improvements in Concurrent Requests ($VR = 0.99$)

Figure 6.9 plots the percentage improvement achieved by the integrated policy versus the number of disks, and it also shows the contributions of caching and batching. Note that the improvement with the integrated policy in the number of concurrent requests approximately ranges from 6% to 17% as the number of disks increases from 30 to 240. Caching contributes to a little less than half of the improvement on the average. The improvement resulting from caching does not follow a linear relationship with the number of disks because of the complex interaction between caching and batching: caching performance improves by servicing requests immediately, whereas batching benefits primarily from delaying their service.

Figure 6.10 depicts the number of concurrent requests versus the average request inter-arrival time ($1/\lambda$). These results show that the number of concurrent requests increases with λ. As stated earlier, this is due to the increase in the average batch size and the number of cached requests. Note that the improvement attained by the integrated policy also increases with λ. The zero-improvement point happens when the

server is lightly loaded, and so it is able to service all incoming requests (defection rate is zero).

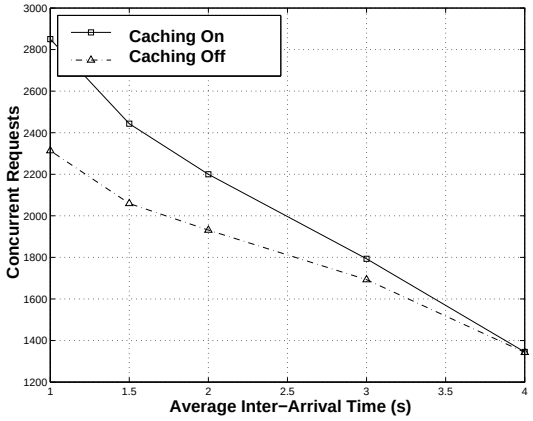


Fig. 6.10: Effect of Average Inter-Arrival Time

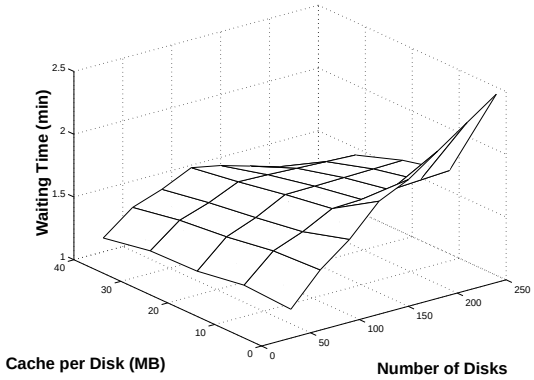


Fig. 6.11: Effect of Number of Disks and Cache Size on Waiting Time

Figure 6.11 illustrates the effects of the number of disks and the cache size per disk on the average request waiting time. The results show that when the caching is turned off (i.e., $cache\,size = 0$), the average request waiting time increases by a factor of two as the number of disks varies from 30 to 240. This behavior is because videos are striped across all disks, so requests wait longer to find empty slots for scheduling. With the integrated policy, however, the average request waiting time increases less dramatically with the number of disks up to a certain point and then starts to decrease since caching services requests as soon as possible. With 30 MB of cache per disk, the reduction in waiting times increases from about 4% to about 40% as the number of disks varies from 30 to 240. The average request waiting time, however, remains more than 1.2 minutes,

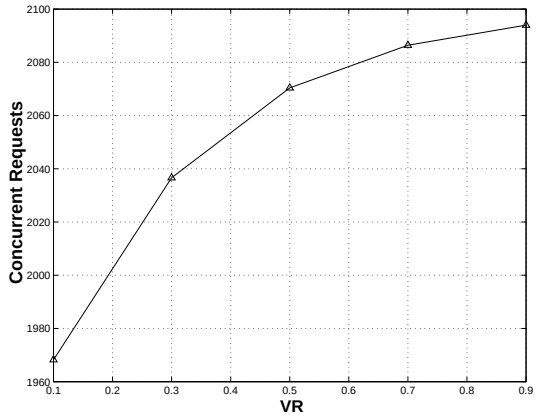
which is 41% of MWT (3 minutes). Such long average waiting time implies that some customers defect just because of the long delay. Thus, dividing disks into partitions can enhance the performance by reducing the waiting times.

6.5.4.2 Effects of the Tunable Parameters: VR, RCR, and DSR

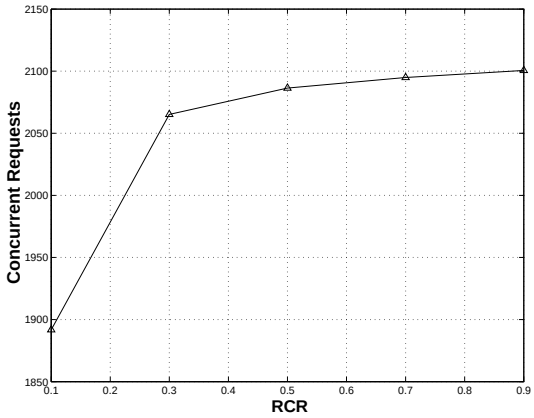
Let us now discuss how VR, RCR, and DSR can be tuned. Figure 6.12(a) shows the effect of VR on performance. This figure indicates that designers should choose the largest possible value of VR whose incurred implementation overhead can be tolerated. The value 0.7 may lead to a good tradeoff because the number of concurrent requests increases slightly with VR after that point.

Figure 6.12(b) plots the number of concurrent requests versus RCR. Note that the number of serviced requests increases with RCR since more requests are examined for scheduling. Ideally, the value of RCR should be kept between 0.3 and 0.5 unless the implementation overhead resulting from using larger values can be tolerated.

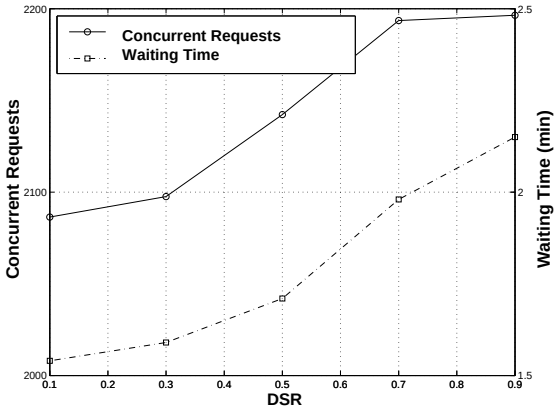
The impact of DSR on the number of concurrent requests and waiting times is shown in Figure 6.12(c). As expected, increasing DSR has a positive effect on the number of concurrent requests and a negative effect on waiting times. Note that increasing DSR from 0.1 to 0.9 increases the number of concurrent requests by about 5.3% and increases the waiting times by about 40%. Hence, if customers are not very sensitive to longer waiting times, the performance of the server can be enhanced by using a larger value of DSR.



(a) Victimization Ratio (VR)



(b) Request Check Ratio (RCR)



(c) Delay Service Ratio (DSR)

Fig. 6.12: Effects of Tunable Parameters

6.5.4.3 Effects of the Scheduling Parameters

Let us now examine the effectiveness of the MOS scheme. Although this scheme can be used to meet various design tradeoffs, the main goal of the evaluation study here is to demonstrate that it can increase the number of concurrent customers while reducing their waiting times for service. Simulation results show that in order to meet this objective, MQL should have a large weight and the other criteria should have small weights. The idea behind these selections of weights is make MQL determine primarily

the scheduling decisions while allowing the other criteria to influence these decisions only when multiple queues have almost the same number of waiting requests. In addition to the four scheduling criteria (discussed in Section 3), two promising special cases of the MOS scheme are examined: *MOS 1* and *MOS 2*. Table 6.4 shows the values of the scheduling parameters for each of these criteria/schemes.

Table 6.4: Scheduling Parameters of Various Criteria

Criterion	$\mathbf{w_b}$	$\mathbf{w_t}$	$\mathbf{w_n}$	$\mathbf{w_m}$
Maximum Queue Length (MQL)	1	0	0	0
First Come First Serve (FCFS)	0	1	0	0
Shortest Interval First (SIF)	0	0	1	0
Most Popular First (MPF)	0	0	0	1
Multi-Objective Scheduling 1 (MOS 1)	1	10^{-4}	10^{-5}	0
Multi-Objective Scheduling 2 (MOS 2)	1	10^{-5}	10^{-6}	10^{-4}

Figures 6.13 and 6.14 plot the percentage improvements achieved by the different criteria with respect to MQL in terms of the number of concurrent requests and the waiting times, respectively, for three settings of λ and RCR. The results of FCFS are not shown in the figure because of its poor performance: it performs 25% - 40% worse than MQL in terms of the number of concurrent requests and 120% - 150% worse than it in the waiting times. These results differ from prior work primarily because this study considers caching, whose performance degrades substantially by scheduling older requests. The number of concurrent requests with FCFS is relatively low because this criterion utilizes neither batching nor caching, and the average waiting time is relatively long because it favors older requests. SIF performs relatively well in terms of the waiting times because it favors newer requests. Similarly, the waiting times are longer with *MOS*

1 than that with *MOS 2* because *MOS 1* has a larger weight w_t. In all the three settings (of λ and RCR), SIF and *MOS 2* reduce the waiting times better than MQL, and *MOS 1* outperforms MQL in terms of the number of concurrent requests.

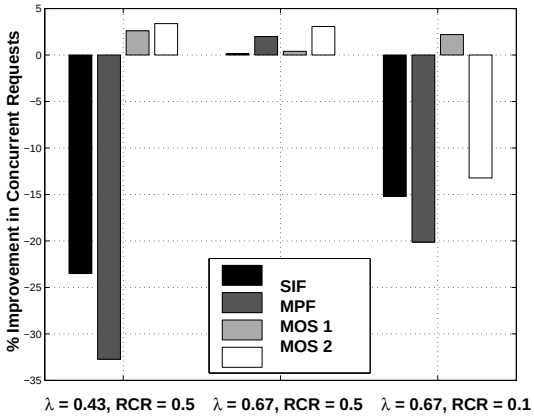


Fig. 6.13: Improvements in Concurrent Requests cf MQL

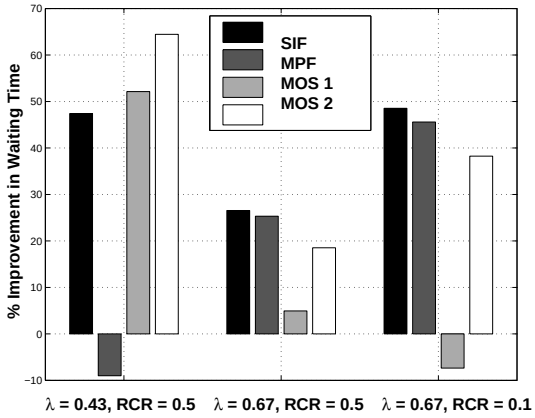


Fig. 6.14: Improvements in Waiting Times cf MQL

The main results can be summarized as follows. The values of the scheduling parameters should be based on the values of λ and RCR. With a careful tuning of these parameters, the MOS scheme can perform better than MQL in terms of both the number of concurrent requests and the waiting times. Both these performance metrics can be improved with a very large value of w_b and a very small value of w_t. The optimal selection of the scheduling parameters is left for a future study.

6.5.4.4 Effects of Other Parameters

Let us now discuss the impacts of disk performance, the maximum request waiting time (MWT), the number of videos, the video data rate, the video duration, and the skewness parameter (θ). Figure 6.15 shows the effect of disk performance (measured

as the number of concurrent requests it can service) on the overall performance. The value 10 in the x-axis corresponds to Atlas III, while higher values correspond to more advanced disks. Note that the performance improvement achieved by the integrated policy ranges between 12% and 14%.

The effect of MWT on the number of concurrent requests is shown in Figure 6.16. As MWT increases, more requests can be serviced because they can wait longer, and the degree of resource sharing improves.

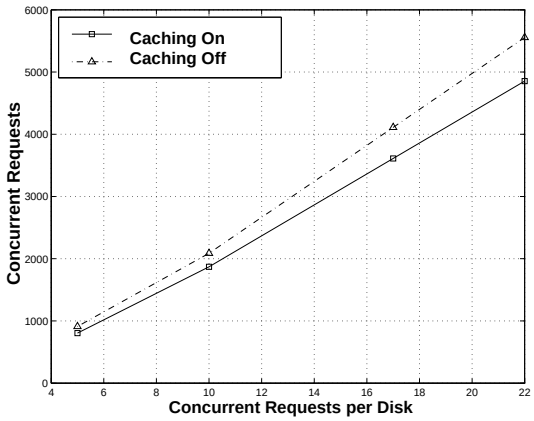


Fig. 6.15: Effect of Disk Performance

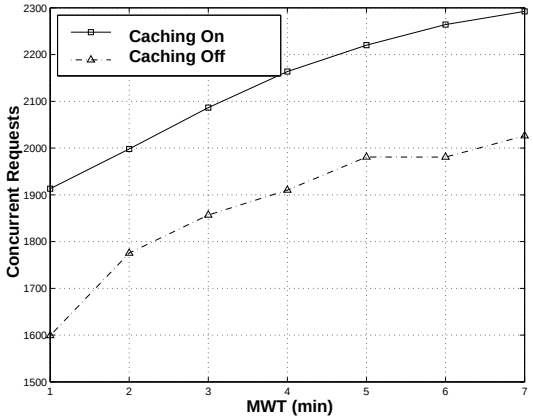


Fig. 6.16: Effect of Maximum Waiting Time (MWT)

Figures 6.17 and 6.18 demonstrate the impacts of the number of videos and video data rate on performance, respectively. Figure 6.17 shows that the number of concurrent streams decreases with the number of videos, which can be explained as follows. The video locality of reference diminishes as the number of videos increases since every additional video consumes part of the total access frequency. The reduction in the locality of reference has a negative impact on both caching and batching. In addition, as the

number of videos increases, the ratio of overall cache size to the overall data size decreases. The figure also shows that the percentage improvement of the integrated policy increases with the number of videos. Figure 6.18 plots the impact of the video data rate. The number of videos here is adjusted in order to keep all disks nearly full. Hence, as the data rate increases, the number of videos decreases. The video duration also has a similar impact as the number of videos, and thus the corresponding figure is not shown.

Finally, the impact of the skewness parameter (θ) on the number of concurrent requests is depicted in Figure 6.19. As expected, both the number of concurrent requests and the improvement of the integrated policy diminish as the skewness in access patterns decreases.

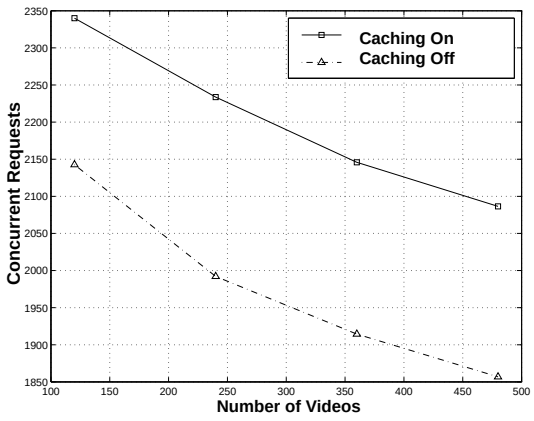


Fig. 6.17: Effect of Number of Videos

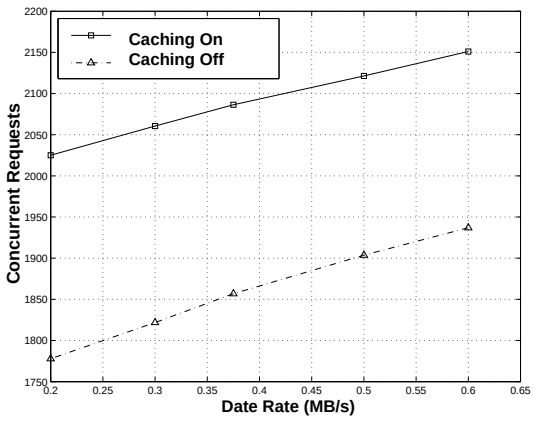


Fig. 6.18: Effect of Video Data Rate

6.5.5 Evaluation of the Effectiveness of the DIC Algorithm

The results in the last subsection demonstrate that the performance improvement achieved by the DIC scheme – with only 30 MB of cache per disk – in the number of

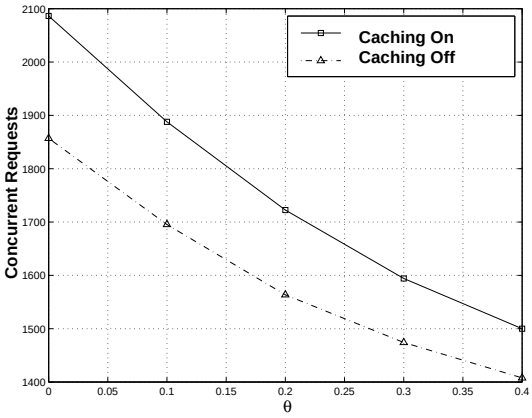


Fig. 6.19: Effect of Skewness Parameter (θ) on Concurrent Requests

concurrent requests ranges from 2% to 9% as the number of disks varies from 30 to 240. Naturally, these improvements are less significant than the improvements achieved by caching in general-purpose systems because of the much larger object size(s) in multimedia servers.

The question arises as to whether we can squeeze additional performance out of NAD-based multimedia servers by polishing the proposed scheme? This subsection answers this question by finding the performance limits on DIC. For this purpose, a model of an *ideal* DIC scheme is required. This scheme minimizes the average cached interval by perfect victimizations and fully utilizes all the on-disk caches. Therefore, all the internal disk caches virtually act as one big cache. Consequently, the analytical model developed in Section 5.3 can be used for this ideal scheme by substituting the sum of all internal cache sizes for the cache size (S).

Figure 6.20 compares the proposed DIC algorithm with the ideal scheme in terms of the number of concurrent cached requests for four disk configurations. The figure

shows that the proposed algorithm achieves between 23% and 33% of the upper limit. The performance gap is primarily because the load is not perfectly balanced among all caches with the proposed algorithm, and the victimization in real DIC is highly constrained. The load balancing problem is due to the distributed nature of the algorithm. These results indicate that the upper limit on DIC is very high, so there may be plenty of room for improving the DIC algorithm. This limit, however, is somewhat loose because the constrained victimization is a nature of the problem, not of the algorithm. Future research is needed to approach the upper limit more closely. Improving the performance of DIC, however, should not be attained by sacrificing the overall performance. For instance, applying the *Shortest Interval First* (SIF) criterion can increase the number of cached streams by reducing the penalty of suboptimal victimizations, but as shown earlier, that is detrimental to the overall performance. In future work, two promising ways to improve the DIC algorithm will be investigated: applying hash functions for mapping videos to disks (instead of the simple round-robin scheme) and dividing disks into partitions. The first way should reduce the load imbalance, whereas the second should reduce the waiting times and customer defection probability.

6.6 Conclusions

Conventional multimedia servers waste precious resources in performing store-and-forward copying. This excessive overhead increases cost and limits the scalability of these servers.

This study has proposed using the *Network-Attached Disk* (NAD) architecture to design highly scalable and cost-effective multimedia servers. It has used both caching

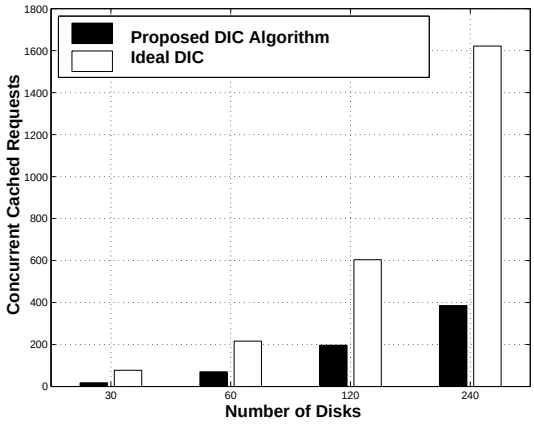


Fig. 6.20: Comparison between the Ideal DIC Algorithm and the Proposed Algorithm

and intelligent scheduling in an *Integrated Resource Sharing policy* to enhance the performance of NAD-based multimedia servers. For caching, it has proposed a *Distributed Interval Caching* (DIC) scheme, which utilizes the on-disk caches (buffers) in caching intervals between successive streams. For scheduling, it has proposed a *Multi-Objective Scheduling* (MOS) scheme, which schedules the waiting requests based on four predefined criteria. The integrated policy is highly configurable; by tuning certain parameters, the administrator can make any necessary tradeoff. These tradeoffs include the number of customers that can be serviced concurrently, the average request waiting time, fairness, and implementation complexity. Certain parameters can also be tuned adaptively to fit the current server workload.

The effectiveness of the integrated policy and the interactions among various parameters have been studied through extensive simulation. The simulation results show that the integrated policy can improve performance significantly and that the performance benefits are greater in larger servers. In particular, it increases the number

of concurrent requests by about 12% in a 60-disk to about 17% in a 240-disk server. Similarly, the policy reduces the average request waiting time by about 8% in a 60-disk server to about 40% in a 240-disk server. Moreover, the results indicate that the MOS scheme can add 4% improvement in the number of concurrent requests and 20% - 65% improvement in the average request waiting time with respect to the best performer of existing scheduling policies.

This study has also used the analytical model presented in Section 5.3 to estimate the upper bound on the performance of DIC. The model shows that the theoretical limit is very high and that there may still be plenty of room for further improvements. It is unclear, however, how much of this limit is in fact achievable.

Chapter 7

Conclusions

The interest in *Multimedia-on-Demand* MOD services has grown dramatically. *Video-on-demand* (VOD) is the most common MOD application and is the application of interest in this thesis.

The design of MOD servers faces significant challenges to support large numbers of concurrent customers and to scale easily and cost-effectively in order to cope with the increasing demands. The number of requests that can be serviced concurrently is highly constrained by the requirements of the real-time playback and the high transfer rates. These requirements impose heavy burdens on server resources, especially the storage subsystem bandwidth and the network bandwidth. The scalability challenge, however, arises primarily because conventional multimedia servers waste precious resources in performing store-and-forward copying.

This thesis has addressed these two challenges. The next two sections summarize the main contributions of the thesis and outline future work.

7.1 Main Contributions

This thesis has proposed an *Adaptive Block Rearrangement* policy, which significantly improves the performance of multimedia storage subsystems by exploiting the high skewness in video access patterns. This policy monitors the accesses to various

videos and keeps the videos with comparable access frequencies close to each other. Two rearrangement algorithms have been analyzed: *Centered-Layout* and *Sequential-Layout*. The centered-layout algorithm places the blocks of the more popular videos nearer to the center of the disk, whereas the sequential algorithm places these blocks nearer to the edge of the disk. The simulation results indicate that the adaptive block rearrangement policy can achieve significant performance benefits. The centered-layout algorithm reduces the mean seek distance by approximately 50% when the number of disks is 4 and by about 60% when the number of disks is 20. The corresponding reductions in the mean disk access time are 13% and 15%, respectively. Similarly, the sequential-layout algorithm can enhance performance, but it performs worse than the centered. In particular, the mean seek distance is about 25% to 45% longer than that of the centered.

This thesis has also proposed a new class of scheduling policies, called *Next Schedule Time First* (NSTF), which provides hard time of service guarantees. Providing such guarantees enhances customer-perceived QoS and server throughput. NSTF provides customers with schedule times, performs scheduling based on these schedule times, and guarantees that customers will be serviced no later than and accurately as scheduled. The thesis has presented alternative implementations of NSTF and has shown that it delivers outstanding performance benefits. The simulation results show that NSTF always meets the time of service guarantees and produces very accurate schedule times. The average deviation of the actual times of service from the schedule times can be within 0.2 second. By contrast, FCFS may violate its time of service guarantees, and these guarantees differ from the actual times of service by 20 seconds to more than 4.5 minutes on the average! The results also show that NSTF can achieve higher throughput

and, in certain situations, shorter waiting times than FCFS that may provide limited time of service guarantees, even under statistically identical models of waiting tolerance.

In addition, this thesis has developed an analytical model for interval caching and has validated it by simulation. It then has used that model to study the impacts of various system and workload parameters and to experiment with new design alternatives. In particular, it has evaluated the effectiveness of a hybrid caching scheme. This scheme caches a small set of the most popular videos in their entireties and employs interval caching for the rest. The results show that the hybrid scheme outperforms interval caching when the cache is sufficiently large.

Moreover, this thesis has proposed using the emerging *Network-Attached Disk* (NAD) architecture for designing large-scale, cost-effective, and scalable multimedia servers. The NAD architecture removes all store-and-forward copying, which severely limits the scalability of conventional multimedia servers.

Furthermore, this thesis has proposed an *Integrated Resource Sharing* policy that enhances the performance of NAD-based multimedia servers through caching and intelligent scheduling. This policy combines two schemes that have also been proposed here: *Distributed Interval Caching* (DIC) and *Multi-Objective Scheduling* (MOS). The DIC scheme exploits the internal caches of NADs in caching intervals between successive streams. The MOS scheme schedules waiting requests for service intelligently based on four performance criteria. The simulation results show that the integrated policy increases the number of concurrent requests by about 12% in a 60-disk to about 17% in a 240-disk server. It also reduces the average request waiting time by about 8% in a

60-disk server to about 40% in a 240-disk server. The MOS scheme can add 4% improvement in the number of concurrent requests and 20% - 65% improvement in the average request waiting time. This thesis has also studied the performance limit of DIC through analytical modeling.

7.2 Future Work

The work on the adaptive block rearrangement policy will be elaborated in two main ways. First, a higher level study will be conducted to evaluate how the improvements in disk performance translate to the overall performance in terms of the number of requests that can be serviced concurrently while maintaining a "reasonable" QoS. Second, the impacts of other factors on the proposed rearrangement algorithms will be examined, including caching, admission control, interactive operations (such as pause, resume, fast forward, and fast rewind), and variable levels of QoS.

The study of the provision of time of service guarantees will be extended by investigating the impacts of additional models of customer waiting tolerance and by exploring new ways to bridge the performance gap in terms of the mean request waiting time between NSTF and MQL/MFQL.

The research on the design NAD-based multimedia servers offers two major opportunities for future work. First, new ways to enhance the DIC algorithm so as to approach the theoretical limit more closely will be explored. One such way is to use hash functions instead of the simple round-robin scheme. Efficient hashing can reduce the imbalance in cache utilization. Second, the effectiveness of dividing disks into partitions (compared with merely striping videos across all disks) will be evaluated. In

addition to improving the reliability, partitioning should reduce the waiting times before requests find empty slots. Reducing such delay boosts performance by shortening the average cached interval.

References

[1] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. On Optimal Piggyback Merging Policies for Video-On-Demand Systems. In *Proceedings of the ACM SIGMETRICS Conference on Measurements and Modeling of Computer Systems*, pages 200-209, May 1996.

[2] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. The Maximum Factor Queue Length Batching Scheme for Video-on-Demand Systems. *IEEE Transactions on Computers*, 50(2): 97-110, February 2001.

[3] D. Aksoy and M. J. Franklin. Scheduling for Large-Scale On-Demand Data Broadcasting. In *Proceedings IEEE INFOCOM*, pages 651-659, April 1998.

[4] S. Akyurek and K. Salem. Adaptive Block Rearrangement. *ACM Transactions on Computer Systems*, 13(2): 89-121, May 1995.

[5] J. Almeida, D. Eager, and M. Vernon. A Hybrid Caching Strategy for Streaming Media Files. In *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, January 2001.

[6] D. P. Anderson, Y. Osawa, and R. Govindan. A File System for Continuous Media. *ACM Transactions on Computer Systems*, 10(4): 311-337, November 1992.

[7] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McMaus. *Requirements for Traffic Engineering over MPLS*. Internet Draft, October 1998.

[8] D. Awduche, D. Gan, T. Li, G. Swallow, and V. Srinivasan. *Extension to RSVP for Traffic Engineering*. Internet Draft, August 1998.

[9] Y. Bernet et al. *A Framework for Differentiated Services*. Internet Draft, May 1998.

[10] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. *An Architecture for Differentiated Service*. RFC 2475, December 1998.

[11] W. J. Bolosky, M. B. Jones, and R. F. Rashid. The Tiger video file server. In *Proceedings of Workshop on Network and Operating System Support for Digital Audio and Video*, pages 97-104, April 1996.

[12] R. Braden, D. Clark, and S. Shenker. *Integrated Services in the Internet Architecture: an Overview*. Internet RFC 1633, June 1994.

[13] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin.*Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*. RFC 2205, September 1997.

[14] Y. Cai and K. A. Hua. An Efficient Bandwidth-Sharing Technique for True Video on Demand Systems. In *Proceedings of the ACM International Conference on Multimedia*, pages 211-214, October 1999.

[15] Y. Cai, K. A. Hua, and K. Vu. Optimizing Patching Performance. In *Proceedings of the IS&T/SPIE Conference on Multimedia Computing and Networking*, pages 204-215, January 1999.

[16] S. R. Carter, J.-F. Pâris, S. Mohan, and D. D. E. Long. A Dynamic Heuristic Broadcasting Protocol for Video-on-Demand. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, pages 657-664, April 2001.

[17] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A Large-scale and Decentralized Application-level Multicast Infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC)* 20(8): 100 -110, October 2002.

[18] A. L. Chervenak. *Tertiary Storage: An Evaluation of New Applications*. Ph.D. Thesis, University of California at Berkeley, December 1994. Also Available as Technical Report UDB/CSD 94/847.

[19] A. L. Chervenak, D. A. Patterson, and R. H. Katz. Choosing the Best Storage Systems for Video Service. In *Proceedings of the ACM Conference on Multimedia*, pages 109-119, November 1995.

[20] M. Chesire, A. Wolman, G. Voelker, and H. Levy. Measurement and Analysis of a Streaming Media Workload. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*, pages 1-12, March 2001.

[21] C. Chou, L. Golubchik, J. C .S. Lui. Striping Doesn't Scale: How to Achieve Scalability for Continuous Media Servers with Replication. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS 2000)*, pages 64-71, April 2000.

[22] Y. Chu, S. Rao, and H. Zhang. A Case for End System Multicast. In *Proceedings of the ACM SIGMETRICS Conference on Measurements and Modeling of Computer Systems*, pages 1-12, June 2000.

[23] D. Clark, S. Shenker, and L. Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism. In *Proceedings of*

SIGCOMM Symposium on Communications Architectures and Protocols, pages 14-26, August 1992.

[24] E. Crawley, R. Nair, B. Jajagopalan, and H. Sandick. *A Framework for QoS-based Routing in the Internet*. RFC 2386, August 1998.

[25] A. Dan, and D. Sitaram. *Buffer Management Policy for an On-Demand Video Server*. Technical Report RC 19347, IBM Watson Research Center, January 1994.

[26] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling Policies for an On-Demand Video Server with Batching. In *Proceedings of the ACM Conference on Multimedia*, pages 391-398, October 1994.

[27] A. Dan, D. M. Dias, R. Mukherjee, D. Sitaram, R. Tewari. Buffering and Caching in Large-Scale Video servers. In *Proceedings of IEEE International Computer Conference*, pages 217-225, March 1995.

[28] A. Dan, P. Shahabuddin, D. Sitaram, and D. Towsley. Channel Allocation under Batching and VCR Control in Movie-on-Demand Servers. *Journal of Parallel and Distributed Computing*, 30(2): 168-179, November 1995.

[29] A Dan and D. Sitaram. *Multimedia Caching Strategies for Heterogeneous Application and Server Environments*. Technical Report RC 20670, IBM Watson Research Center, December 1996.

[30] A. Dan and D. Sitaram. A Generalized Interval Caching Policy for Mixed Interactive and Long Video Workloads. In *Proceedings of Multimedia Computing and Networking Conference (MMCN)*, pages 344-351, January 1996.

[31] H. D. Dykeman, M. H. Ammar, and J. W. Wong. Scheduling Algorithms for Videotex Systems under Broadcast Delivery. In *Proceedings of IEEE International Conference on Communication*, pages 1847-1851, June 1986.

[32] EMC, *Delivering a World of Content Through Rich Media Solutions*. On-line article at `http://www.emc.com`, 2000.

[33] D. Ferrari. Client Requirements for Real-Time Communication Services. *IEEE Communications Magazine*, 28(11):65-72, November 1990.

[34] R. Flynn, and W. Tetzlaff. Disk Striping and Block Replication Algorithms for Video File Servers. In *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 590-597, June 1996.

[35] G. Ganger, and Y. Patt. The Process-Flow Model: Examining I/O Performance from System's Point of View. In *Proceedings of the ACM SIGMETRICS Conference on Measurements and Modeling of Computer Systems*, pages 86-97, May 1993.

[36] G. Ganger. Generating Representative Synthetic Workloads: An Unsolved Problem. In *Proceedings of Computer Measurement Group (CMG) Conference*, pages 1263-1269, December 1995.

[37] G. Ganger, B. Worthington, and Y. Patt. *The DiskSim Simulation Environment Version 2*. Reference Manual, Carnegie Mellon University, December 1999.

[38] G. Ganger and J. Schindler. *Database of Validated Disk Parameters for DiskSim*. Available at: `http://www.ece.cmu.edu/~ganger/disksim/diskspecs.html`.

[39] D. J. Gemmell and J. Han. Multimedia Network File Servers: Multi-Channel Delay Sensitive Data Retrieval. *Multimedia Systems*, 1(6):240-252, April 1994.

[40] D. J. Gemmell, H. M. Vin, D. D. Kandlur, P. V. Rangan, and L. A. Rowe. Multimedia Storage Servers: A Tutorial. *IEEE Computer*, 28(5): 40-49, May 1995.

[41] S. Ghandeharizadeh and D. Kim. *On-line Re-organization of Data in Scalable Continuous Media Servers*. Technical Report USC-CS-TR96-634, University of Southern California, 1996.

[42] G. A. Gibson, D. F. Nagle, K. Amiri, F. W. Chang, E. Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, and D. Rochberg. *A Case for Network-Attached Secure Disks*. Technical Report CMU-CS-96-142, Carnegie Mellon University, September 1996.

[43] G. A. Gibson, D. F. Nagle, K. Amiri, F. W. Chang, E. M. Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, D. Rocherg, and J. Zelenka. File Server Scaling with Network-Attached Secure Disks. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 272-284, June 1997.

[44] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobio , C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. *Filesystems for Network-Attached Secure Disks*. Technical Report CMU-CS-97-118, Carnegie Mellon University, July 1997.

[45] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. A Cost-Effective, High-Bandwidth Storage Architecture. In *Proceedings of the Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS)*, October 1998.

[46] L. Giuliano. *Deploying Native Multicast across the Internet*. Online White Paper at `http://www.sprintlink.net/multicast/whitepaper.html`.

[47] H. Gobioff. *Security for a High Performance Commodity Storage Subsystem*. Ph.D. Thesis, Carnegie Mellon University, July 1999. Also Available as Technical Report CMU-CS-99-160.

[48] L. Golubchik, J. Lui, and R. Muntz. Adaptive Piggybacking: A Novel Technique for Data Sharing in Video-On-Demand Storage Servers. *ACM Multimedia Systems Journal*, 4(3): 140-155, 1996.

[49] L. Golubchik, J. C. S. Lui, and R. Muntz. Reducing I/O Demand in Video-On-Demand Storage Servers. In *Proceedings of the ACM SIGMETRICS Conference on Measurements and Modeling of Computer Systems*, pages 25-36, May 1995.

[50] D. D. Grossman, and H. F. Silverman. Placement of Records on a Secondary Storage Device to Minimize Access Time. *In Journal of the ACM* 20(3): 429-438, 1973.

[51] R. Guerin, S. Blake, and S. Herzog. *Aggregating RSVP-based QoS Requests*. Internet Draft, November 1997.

[52] G. H. Hardy, J. E. Littlewood, and G. Polya. *Inequalities*. Cambridge University Press, Cambridge, England, 1952.

[53] R. Haskin. The Shark Continuous Media File Server. In *Proceedings of IEEE 1993 Spring COMPCON*, pages 12-17, February 1993.

[54] R. Haskin and F. Stein. A System for the Delivery of Interactive Television Programming. In *Proceedings of IEEE 1995 Spring COMPCON*, pages 209-214, March 1995.

[55] A. Hu. Video-on-Demand Broadcasting Protocols: A Comprehensive Study. In *Proceedings of IEEE INFOCOM*, Vol 1, pages 508-517, April 2001.

[56] K. A. Hua, S. Sheu. Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand System. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'97)*, pages 89-100, September 1997.

[57] K. A. Hua, Y. Cai, and S. Sheu. Patching: A Multicast Technique for True Video-on-Demand Services. In *Proceedings of ACM Multimedia*, pages 191-200, September 1998.

[58] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI)*, pages 197-212, October 2000.

[59] L. Juhn and L. Tseng. Harmonic Broadcasting for Video-on-Demand Service. *IEEE Transactions on Broadcasting*, 43(3): 268-271, September 1997.

[60] S. Jamin, S. Shenkar, L. Zhang, and D. D. Clark. An admission Control Algorithm for Predictive Real-Time Service. In *Proceedings of the International Workshop on*

Network and Operating System Support for Digital Audio and Video, pages 349-356, November 1992.

[61] K. Keeton, A. L. Drapeau, D. A. Patterson, and R. H. Katz. Storage Alternatives for Video Service. In *Proceedings of IEEE Symposium on Mass Storage Systems* pages 100-105, June 1994.

[62] K. Keeton, D. A. Patterson, and J. M. Hellerstein. *The Intelligent Disk (IDISK): A Revolutionary Approach to Database Computing Infrastructure*. White Paper, University of California at Berkeley, May 1998.

[63] S. Kim, A. Sivasubramaniam, and C. R. Das. Analyzing Cache Performance for Video Servers. In *Proceedings of the International Conference on Parallel Processing Workshops on Architectural and OS Support for Multimedia Applications*, pages 38-47, August 1998.

[64] S. Kim and C. R. Das. A Reliable Statistical Admission Control Strategy for Interactive Video-On-Demand Servers with Interval Caching. *In Proceedings of the International Conference on Parallel Proceedings*, pages 135-142, August 2000.

[65] S. Kim, C. R. Das, and A. Sivasubramaniam. Performance Analysis of A Buffer Management Technique for Interactive Video-on-Demand. In *Proceedings of the International Conference on Multimedia Modeling*, November 2000.

[66] S. Kumar and P. Mohapatra. *A Disk Scheduling Algorithm for Multimedia Storage Servers*. Technical Report TR-ACAR-96-04, Iowa State University, 1996.

[67] J. Kurose and K. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*, Addison Wesley, 2001.

[68] T. Li and Y. Rekhter. *Provider Architecture for Differentiated Services and Traffic Engineering (PASTE)*. RFC 2430, October 1998.

[69] F. Y. S. Lin. Optimal Real-time Admission Control Algorithms for Video-On-Demand (VOD) Service. *IEEE Transactions on Broadcasting*, 44(4): 402-408, December 1998.

[70] P. Lougher, D. Shenker, L. Zhang, and D. D. Clark. The Design and Implementation of a Continuous Media Storage Server. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 69-80, November 1992.

[71] G. Ma, A. Khaleel, and N. Reddy. Performance Evaluation of Storage Systems Based on Network-Attached Disks. *IEEE Transactions on Parallel and Distributed Systems*, 11(9): 956-968, September 2000.

[72] E. L. Miller, D. D. E. Long, W. E. Freeman, and B. C. Reed. Strong Security for Network-Attached Storage. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, pages 1-14, January 2002.

[73] D. Nagle, G. Ganger, J. Butler, G. Goodson, and C. Sabol. Network Support for Network-Attached Storage. In *Proceedings of Hot Interconnects*, August 1999.

[74] J. Nieh and M. S. Lam. The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 184-197, October 1997.

[75] J.-F. Pâris, S. W. Carter, and D. D. E. Long. Efficient Broadcasting Protocols for Video on Demand. In *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 127-132, July 1998.

[76] J.-F. Pâris. A Fixed-Delay Broadcasting Protocol for Video-on-Demand. In *Proceedings of the International Conference on Computer Communications and Networks*, pages 418-423, October 2001.

[77] Quantum Corporation. *A New Standard of Performance*. On-line article at: `http://www.quantum.com/products/hdd/article_atlas10k.htm`, 2000.

[78] P. V. Rangan, and H. M. Vin. Designing File Systems for Digital Video and Audio. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 81-94, October 1991.

[79] A. L. N. Reddy, J. Wyllie. Disk Scheduling in a Multimedia I/O System. In *Proceedings of the ACM Conference on Multimedia*, pages 225-233, August 1993.

[80] A. L. N. Reddy and J. C. Wyllie. I/O Issues in a Multimedia System. *IEEE Computer Magazine*, 27(3): 69-74, March 1994.

[81] E. Riedel, M. Kallahalla, and R. Swaminathan. A Framework for Evaluating Storage System Security. In *Proceedings of the USENIX Conference on File and Storage Technology (FAST)*, pages 15-30, January 2002.

[82] E. Rosen, A. Viswanathan, and R. Callon. *Multiprotocol Label Switching Architecture*. Internet Draft, March 1998.

[83] Internet Engineering Task Force. *RTP: A Transport Protocol for Real-Time Applications*. Internet Draft, July 2000.

[84] H. Schulzrinne, A. Rao, and R. Lanphier. *Real Time Streaming Protocol (RTSP)*. RFC 2326, April 1998.

[85] C. Reummler, and J. Wilkes. *Disk Shuffling*. Technical Report HPL-91-156, Hewlett-Packard Laboratories, October 1991.

[86] C. Ruemmler, and J. Wilkes. An Introduction to Disk Drive Modeling. *IEEE Transactions on Computer*, 27(3): 17-29, March 1994.

[87] N. J. Sarhan. *An Investigation of Scheduling Policies for Multimedia Systems*, Master of Science Thesis, Pennsylvania State University, May 2003.

[88] N. J. Sarhan and C. R. Das. An Integrated Resource Sharing Policy for Multimedia Storage Servers Based on Network-Attached Disks. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, pages 136 - 143, May 2003.

[89] N. J. Sarhan and C. R. Das. Providing Time of Service Guarantees in Video-On-Demand Servers. In *Poster Proceedings of the International World Wide Web Conference (WWW)*, May 2003.

[90] N. J. Sarhan and C. R. Das. A Simulation-Based Analysis of Scheduling Policies for Multimedia Servers. In *Proceedings of the Annual Simulation Symposium (part of the Advanced Simulation Technologies Conference)*, pages 183 - 190, March 30 - April 2, 2003.

[91] N. J. Sarhan and C. R. Das. *A Detailed Study of Request Scheduling in Multimedia Systems*. Technical Report CSE-03-002, The Pennsylvania State University, January 2003.

[92] N. J. Sarhan and C. R. Das. *Caching and Scheduling Techniques for Multimedia Storage Servers Based on Network-Attached Disks*. Technical Report CSE-02-013, The Pennsylvania State University, September 2002.

[93] N. J. Sarhan and C. R. Das. Adaptive Block Rearrangement Algorithms for Video-On-Demand Servers. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, pages 452 - 459, September 2001.

[94] N. J. Sarhan and C. R. Das. *Proposing Block Rearrangement for VOD Servers*. Technical Report CSE-01-019, The Pennsylvania State University, June 2001.

[95] J. Schindler and G. Ganger. Automated Disk Drive Characterization. In *Proceedings of the ACM SIGMETRICS Conference on Measurements and Modeling of Computer Systems*, pages 112-113, June 2000.

[96] S. Sen, L. Gao, J. Rexford, and D. Towsley. Optimal Patching Schemes for Efficient Multimedia Streaming. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, June 1999.

[97] H. Shachnai and P. S. Yu. Exploiting Wait Tolerance in Effective Batching for Video-on-Demand Scheduling. *Multimedia Systems*, 6(6): 382-394, 1998.

[98] P. Shenoy, and V. Harric. Efficient Striping Techniques for Variable Bit Rate Continuous Media File Servers. *Performance Evaluation Journal*, 38(2): 175-199, December 1999.

[99] Sprint Corporation. *SprintLink Multicast.* Online Article at `http://www.sprintlink.net/multicast/whitepaper.html`.

[100] C. Staelin, and H. Garcia-Molina. Smart Filesystems. In *Proceedings of the Winter 1991 USENIX Conference,* pages 45-51, 1991.

[101] R. Tewari, H. M. Vin, A. Dan, and D. Sitaram. Resource-based Caching for Web Servers. In *Proceedings of the SPIE Conference on Multimedia Computing and Networking*, pages 191-204, January 1998.

[102] D. A. Tran, K. A. Hua, and T. T. Do. Layered Range Multicast for Video on Demand. In *Proceedings of IEEE International Conference on Computer Communications and Networking*, pages 210-215, October 2002.

[103] A. K. Tsiolis and M. K. Vernon. Group-Guaranteed Channel Capacity in Multimedia Storage Servers. In *Proceedings of the ACM SIGMETRICS Conference on Measurements and Modeling of Computer Systems*, pages 285-297, June 1997.

[104] H. M. Vin and P. V. Rangan. Admission Control Algorithms for Multimedia-On-Demand Servers. In *Proceedings of the International Workshop for Digital Audio and Video*, pages 56-68, November 1992.

[105] H. M. Vin and P. V. Rangan. Designing a Multi-User HDTV Storage Server. *IEEE Journal on Selected Areas in Communications*, 11(1): 153-164, January 1993.

[106] H. Vin, P. Goyal, A. Goyal, and A. Goyal. An Observation-Based Admission Control Algorithm for Multimedia Servers. In *Proceedings of International Conference on Multimedia Computing and Systems*, pages 234-243, May 1994.

[107] H. M. Vin, P. Goyal, and A. Goyal. A Statistical Admission Control Algorithms for Multimedia Servers. In *Proceedings of the ACM Multimedia*, pages 33-40, October 1994.

[108] H. M. Vin, A Goyal, and P. Goyal. Algorithms for Designing Multimedia Servers. *Computer Communications*, 18(3): 192-203, March 1995.

[109] P. Vongsathorn, and S. D. Carson. A System for Adaptive Disk Rearrangement. *Software – Practice and Experience*, 20(3): 225-242, March 1990.

[110] P. P. White. RSVP and Integrated Services in the Internet: A Tutorial. *IEEE Communications Magazine* 35(5): 100-106, pages 100-106, May 1997.

[111] C. K. Wong. Minimizing Expected Head Movement in One-Dimensional and Two-Dimensional Mass Storage Systems. *ACM Computing Surveys*, pages 167-178, June 1980.

[112] J. W. Wong. Broadcast Delivery. *In Proceedings of the IEEE*, 76(12): 1566-1577, December 1988.

[113] J. W. Wong and M. H. Ammar. Analysis of Broadcast Delivery in a Videotex System. *IEEE Transactions on Computers*, 34(9): 863-866, September 1985.

[114] B. Worhington, G. Ganger, and Y. Patt. Scheduling Algorithms for Modern Disk Drives. In *Proceedings of the ACM SIGMETRICS Conference on Measurements and Modeling of Computer Systems*, pages 146-156, May 1994.

[115] Worthington, B. *Aggressive Centerized and Distributes Scheduling of Disk Requests*. Ph.D. Thesis, University of Michigan at An Arbor, June 1995. Also Available as Technical Report CSE-TR-244-95.

[116] W. E. Wright. An Efficient Video-on-Demand Model. *IEEE Computer* 34(5): 64-70, May 2001.

[117] X. Xiao and L. M. Ni. Internet QoS: A Big Picture. *IEEE Network Magazine* 13(2): 8-18, March-April 1999.

[118] G. K. Zipf. *Human Behavior and Principles of Least Effort: An Introduction to Human Ecology*. Addison Wesley, Cambridge, Massachusetts, 1949.

Vita

Nabil J. Sarhan was born in Irbid, Jordan. He received his B.S. degree in Electrical Engineering from Jordan University of Science and Technology in 1995. He worked as a teaching assistant at the same University for about one year and a half. He received his M.S. degree in Computer Science and Engineering from the Pennsylvania State University in 2003. The title of his M.S. thesis is *An Investigation of Scheduling Policies for Multimedia Systems*. Currently, he is a Ph.D. candidate in Computer Science and Engineering at the Pennsylvania State University. While at the Pennsylvania State University, he has worked as a teaching assistant, a research assistant, and an instructor. He received a Computer Science and Engineering Teaching Award in 2001. His main research areas are multimedia systems and networking, storage subsystems, multiprocessor systems, computer architecture, and performance evaluation. His research contributions are published in the International Conference on Distributed Computing Systems (ICDCS 2003), the International World Wide Web Conference (WWW 2003), the Annual Simulation Symposium (ANSS 2003), and the International Conference on Parallel Processing (ICPP 2001).