EFFICIENT DATA STORAGE AND RETRIEVAL TECHNIQUES FOR HIGH-PERFORMANCE VIDEO SERVERS

By

Jinsung Cho

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
SEOUL NATIONAL UNIVERSITY
SEOUL 151-742, KOREA
DECEMBER 1999

SEOUL NATIONAL UNIVERSITY

DEPARTMENT OF

COMPUTER ENGINEERING

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "**Efficient Data Storage and Retrieval Techniques for High-performance Video Servers**" by **Jinsung Cho** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Dated: December 1999

Chief Examiner: ______________________________

Kim, Chong Sang

Research Supervisor: ______________________________

Shin, Heonshik

Examing Committee: ______________________________

Choi, Yanghee

____

Yeom, Heon Young

____

Choi, Chang Ryeol

To my family with love and respect

Abstract

During the past decade we have paid great attention to a video-on-demand service which provides the combined facilities of a video rental store over high-speed networks. The realization of such services requires the development of video servers that support efficient mechanisms for storing and retrieving video data. The study on video servers is highly motivated by the fact that video-on-demand services are becoming increasingly important and widespread in the entertainment, education, and telecommunications industries. In this thesis we design and/or implement video servers in various environments by providing efficient storage and retrieval of video data.

This thesis consists of three main parts: storage and retrieval in a single server, in a large-scale server, and in a multi-resolution video server. In the first part, we begin by designing and implementing a small-scale single server. The large size and bandwidth of video data require a single server approach to adopt disk arrays for its storage subsystem. Through a simple performance analysis for disk arrays, a prototype of the small-scale video server is developed.

The second part of this thesis explores the issue on designing a large-scale video server. Limitations in scalability of the single server approach lead to a large number of such nodes connected to each other. First, given a large number of nodes, we describe how to cluster them into parallel servers. Next, storage and retrieval in a parallel server are presented including data placement, retrieval scheduling, and communication scheduling between nodes. The data placement is based on the block-striping technique verified in the single server and, by the proposed scheduling algorithms, disk bandwidth in a parallel server can be fully utilized and communication between nodes in a parallel server is guaranteed conflict-free. In addition, a queueing model is proposed with a parallel server being an independent service entity in the large-scale server and its performance is analyzed.

The final part addresses the issue on storage and retrieval of multi-resolution video. To this end, we describe a design framework for multi-resolution video servers and implement a prototype. First, we propose a multi-resolution video stream model which can be implemented by various scalable compression techniques. Second, given the proposed stream model, we devise a data placement scheme to store scalable video data across disks in the server. The scheme exploits both concurrency and parallelism offered by striping data across the disks and achieves the disk load balancing during any resolution video service. Next, the retrieval of multi-resolution video is described. The deterministic access property of the placement scheme permits the retrieval scheduling to be performed on each disk independently and to support interactive operations simply by reconstructing the input parameters of the scheduler. We also present an efficient admission control algorithm which precisely estimates the actual disk workload for the given resolution services. In addition, through the implementation of the multi-resolution video manager, we validate the proposed scheme.

Keywords: video server architecture, storage and retrieval, disk arrays, retrieval scheduling, communication scheduling, multi-resolution video, interactive operations, admission control

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and objective

Recent advances in computer technology and demands of video, audio, and text integration services have provided driving forces behind the emergence of various multimedia applications[Agne96, Bufo94]. Among them, we have paid great attention to a video-on-demand (VOD) service which provides the combined facilities of a video rental store over high-speed networks. VOD services are becoming increasingly important and widespread in the entertainment, education, and telecommunications industries [Andl96, Chan96b]. The architecture for these services consists of video servers connected to client sites via high-speed network. Clients can retrieve video streams from the server for real-time playback. Furthermore, the access may be interactive because clients are likely to stop, pause, and resume playback and, in some cases, to perform fastforward or rewind operations.

The realization of such services requires the development of video servers that support efficient mechanisms for storing and delivering video streams. The fundamental problem in developing video servers is that the delivery and playback of video streams must be performed at real-time rate [Gemm95]. Unlike other types of data, video data is characterized by its large size and bandwidth. Although compressed, a two-hours long MPEG-1 [Gall91] video stream requires 1.5Mbps bandwidth and 1.3GB storage space. Video servers should support for efficient storage and retrieval techniques of video data in order to provide such large bandwidth and space of storage subsystem for a large number of clients.

The three main categories of data storage today are tape, disk, and memory. Tape storage is the least expensive and tape drive throughput is also reasonable at about 1MB/s. However, the latency between accesses to different sessions is typically on the order of seconds or minutes. This latency is too high to support multiple independent clients, and as a result, tape drives are limited to one user at a time. On the other extreme, main memory is very fast and has extremely low latency, but the cost is about two orders of magnitude higher than disk. A reasonable compromise between the two extremes is disk-based storage and retrieval. In addition, disk arrays, or RAID [Patt88], have been proved to provide cost-effective storage and high-bandwidth transfer capabilities and are becoming popular in video-on-demand systems. This thesis targets disk-based data storage and retrieval for video servers.

The objective of this thesis is to design video servers in various environments and to evaluate the effectiveness of them through simulation and implementation. For effectiveness, we define the performance metrics of video servers as *concurrency*, *interactivity cost*, and *service latency*. Video servers should be able to provide services for *as many concurrent clients as possible* while guaranteeing their real-time playback requirements. In addition, *interactive operations* such as pause, resume, fastforward, rewind, and slow playback, should be supported with reasonable cost. The *service latency* upon startup or interactive operations should be acceptable. In summary this thesis aims at designing video servers that can service as many clients as possible by fully utilizing the server resources while providing acceptable interactivity cost and service latency.

1.2 Research contribution

1.2.1 Storage and retrieval in a single server

Video servers range from a standard PC for small-scale systems to massively parallel or distributed computers for large-scale systems [Lee98]. First, we tackle the problems of designing and implementing a small-scale single server which is equipped with disk arrays. The single server approach for video servers needs to adopt disk arrays because video servers are required to provide large storage space and transmission bandwidth.

A simple performance analysis for disk arrays is conducted through simulation and the proper storage architecture for a single server is proposed [Cho95]. Based on the proposed architecture, we implement a software disk array manager which controls SCSI adapters and SCSI disks [Cho96]. The implementation details are described and its performance is empirically evaluated and compared with the simulation result.

On top of the disk array manager, a video server is developed. By integrating the server with a VOD system implemented in [Ahn95], we figure out the behavior of video servers and get some feedbacks.

1.2.2 Storage and retrieval in a large-scale server

The single server approach, however, has limitations in scalability. For the purpose of providing video services for the public, a video server should store thousands of video streams and serve tens of thousands of concurrent clients. For such a large-scale video server, we focus on the parallel server architecture which consists of multiple nodes connected by an interconnection network.

First of all, given a large number of nodes, we describe how to cluster such nodes into parallel servers [Cho97a, Cho97e]. In other words, a large-scale server consists of multiple parallel servers while a parallel server is comprised of multiple nodes.

Next, storage and retrieval in a parallel server are addressed [Cho97a, Cho97e] including data placement, retrieval scheduling, and communication scheduling between nodes. The data placement is based on the block-striping technique (AID5) validated through the work on single server. By the proposed scheduling algorithms, disk bandwidth in a parallel server can be fully utilized and communication between nodes in a parallel server is guaranteed conflict-free.

Then, each parallel server in a large-scale video server provides individual services for clients. A queueing model is proposed with a parallel server being an independent service entity and its performance is analyzed [Cho97b, Cho98a]

1.2.3 Storage and retrieval in a multi-resolution video server

The final issue of the thesis is to design and implement a video server which provides multiple resolution video services. A multi-resolution video stream is a video sequence encoded such that subsets of the full resolution video bit stream can be decoded to recreate lower resolution video streams. Employing the multi-resolution video in video servers provides benefits including heterogeneous client support, storage efficiency, adaptive service, and interactive operations support. We present a design framework for multi-resolution video servers by describing multi-resolution stream model, data storage and retrieval of multi-resolution video, interactive operations support, and admission control [Cho97c, Cho99a, Cho99b].

First, a z-level multi-resolution video stream model is proposed. In the multi-resolution video stream model, each video stream can be provided with z levels of quality and the QoS parameter is represented by the number of components in a segment. We also describe how to construct the proposed multi-resolution video stream model from the current scalable compression techniques.

Second, storage and retrieval of multi-resolution video are explored. The data placement scheme exploits both of concurrency and parallelism offered by striping data across disks and achieves the disk load balancing during any resolution video service. The deterministic access property of the placement scheme permits the retrieval scheduling to be performed on each disk independently and to support interactive operations simply by reconstructing the input parameters of input scheduler. We also present an efficient admission control algorithm which precisely estimates the actual disk workload for the given resolution services.

Finally, we describe implementation experiences of a multi-resolution video server [Cho98b, Cho98c]. For the quick implementation, we extend the software disk array manager mentioned in Subsection 1.2.1 to incorporate the storage and retrieval of multi-resolution video and employ MPEG-1 streams with existing hardware decoder for multi-resolution video streams. MPEG-1 streams are reconstructed into the multi-resolution video stream model in temporal dimension. A prototype of the multi-resolution VOD system exhibits that the visual quality of multi-resolution playback and fastforward playback is acceptable.

1.3 Organization

The rest of this thesis is organized as follows.

Chapter 2 gives background for the thesis. Research issues on video servers are addressed from basics to advanced topics. Related researches on video servers are also described.

Chapter 3 designs and implements a small-scale single server equipped with disk arrays. Given the system model, a simple performance analysis for disk arrays is conducted and implementation details for a software disk array manager follow.

Chapter 4 tackles the problems of designing a large-scale video server. Storage and retrieval in a parallel server which consists of multiple nodes are described including data placement, retrieval scheduling, and communication scheduling between nodes. Given a large number of nodes, the configuration of a large-scale server, that is, how to cluster such nodes into parallel servers is also addressed following a queueing analysis of the large-scale server.

Chapter 5 presents a design framework for multi-resolution video servers by describing multi-resolution video stream model, data storage and retrieval of multi-resolution video, interactive operations support, and admission control. The impacts of mobility on video servers are also identified following support for mobile computing environment of the multi-resolution video server. In addition, a prototype of the multi-resolution video server is developed and its performance is measured and analyzed.

Finally, Chapter 6 summarizes the results obtained from this thesis with some concluding remarks.

Chapter 2

Background

2.1 Research issues on video servers

In this section, we introduce various research issues on video servers from basics to advanced topics. The issues, however, are closely coupled with each other.

2.1.1 Basic design issues

A. Guaranteed retrieval of continuous media

Due to its strict timing constraint of continuous media (digital video and audio), early works focus on its guaranteed retrieval and admission control [Rang91a, Gemm92, Ande92, Loug92, Cho94]. Their works found a framework for designing continuous media servers including round-based scheduling, disk performance analysis, admission control strategy, and buffer management. Most of their works, however, are based on the worst-case disk performance and constant bit rate streams.

B. Disk scheduling

For improving the disk performance, several works are performed on the disk head scheduling. An elevator-type or SCAN disk scheduling [Pete85] receives much attention in the literature [Kand93, Cho94] and its variation, or SCAN-EDF algorithm is proposed in [Redd93]. In order to provide the tradeoff between disk throughput and buffer requirements, Yu *et al.* propose a grouped sweeping scheme by integrating FIFO and SCAN algorithms [Yu92] and extend it to accommodate heterogeneous streams [Chen93].

C. Data placement

Placing video streams on a single disk or multiple disks in an array is another issue for the increased performance of storage subsystem. Rangan and Vin [Rang93] propose a constrained allocation policy of digital continuous media on a single disk. By employing a probabilistic model of video popularity, Little and Venkatesh [Litt93] describe data distribution and replication to balance client requests with available disk I/O bandwidth. In order to place video data across multiple disks, Berson *et al.* [Bers94, Bers95] propose a flexible technique called staggered striping and Vin *et al.* [Vin95] identify two placement policies for optimizing the disk-array performance. Wang *et al.* [Wang97b] formulate the problem of video file allocation over disk arrays and present some heuristic algorithms to find the near-optimal solutions. They argue that the consequence of replication and striping of hot movie titles is the potential increase on the required number of disk arrays.

D. Buffer management

Most of early works assume the sufficiently large buffers. Wu and Yu [Wu96, Wu98] study the issue of dynamically utilizing the spare disk bandwidth and buffer to maximize the system throughput of a video server. They introduce the concept of minimizing buffer consumption to select an appropriate media stream to use the spare disk bandwidth. Ng and Yang [Ng96] study the problem of how to maximize the throughput of a continuous-media system, given fixed amounts of buffer space and disk bandwidth both predetermined at design time. Their approach is to maximize the utilizations of disk and buffers.

In addition, several works explain the issues for video servers in detail [Gemm95, Ozde95, Mour96, Sriv97].

2.1.2 Further issues

A. VBR stream manipulation

Variable bit rate (VBR) video streams generate undeterministic workload, so that a careful manipulation should be conducted in order to fully utilize server resources. Admission control algorithms are proposed for tightly estimating workloads of VBR streams [Vin94, Neuf96, Maka97] and scheduling support for VBR streams are given in [Paek96, Rosa96, Lee97, Pan98].

B. Support for interactive operations

Video servers should provide interactive operations such as stop, pause, resume, fastforward, and rewind. Among them, fast scan operations (fastforward and rewind) may require additional disk and network bandwidth. A lot of works are performed to support them with acceptable cost. Dey-Sircar *et al.* [Dey94] introduce an effective FF/Rew service which provides FF/Rew capabilities with associated statistical QoS guarantees. Chen *et al.* propose a segment skipping scheme in the server's side [Chen94] and a stream conversion scheme in the client's side [Chen96], respectively. Kwon *et al.* [Kwon97] support interactive operations efficiently by a disk placement scheme called PRR. Wu and Shu [Wu97] present two basic scheduling approach, the prefetching approach and the grouping approach for both fine-grain and coarse-grain data blocks. In addition, support for interactive operations in multicast VOD servers is given in [Abra98]

C. Caching and page replacement

Although the performance gains of caching and page replacement schemes are relatively small due to the continuous access property of video streams, several works tackle the problems of caching and page replacement schemes including an interval caching [Dan95] and new basic replacement algorithm and the distance-based replacement algorithm [Ozde96]. Reddy [Redd97] studies several caching strategies for improving the overall performance of the server and shows that request response time can be improved by some of the replacement policies that take size of the request into account.

D. Cost-effective design

The number of clients that can be serviced simultaneously can be increased just by adding disks and/or buffer memory to video servers. Hence, the cost-effective design is a significant issue for video servers. Doganata and Tantawi [Doga93] present an analytical tool which allows a user to perform a cost/ performance analysis of video servers with hierarchical storage. The underlying model comprises multiple systems, main memory, expanded storage, disks and a tape library. They argue that the tool optimally allocates the video files to different storage media based on the system parameters and the video file request probability distribution. Chervenak *et al.* [Cher95] propose that striped disk farms achieve close to full disk utilization, good load balancing, and the lowest cost per video stream and Chang and Zakhor provide cost analyses for VBR video servers in [Chan96a].

E. Zoned disk

Zoned disks represent an emerging trend in disk technology. Several works study the placement of multimedia data on zoned disks that maximizes disk throughput. Tewari *et al.* [Tewa96b] describe an optimal placement of fixed-size blocks on zoned disks that minimizes response time using the differences in popularity and access rates among the multimedia objects. Similarly, Kim *et al.* [Kim97b] propose an efficient video block placement scheme considering the characteristics of zoned disks and clients' skewed access patterns for some popular videos. In [Tong98], Tong *et al.* propose two schemes, free-π and fixed-π schemes, in a unified framework of rearranging the zone layout in a logical manner.

F. Hierarchical storage management

Hierarchical storage structures consisting of memory, disk, and tertiary storage devices provide a cost-effective solution for the large size of multimedia repositories. Ghandeharizadeh and Shahabi [Ghan94] investigate the role of hierarchical multimedia storage managers and describe a piplelining mechanism that overlaps the display of a portion of an object from the disk drive with the materialization of its remainder from the tertiary. Lau and Lui [Lau97] consider a two-tier storage architecture with a robotic tape library as the vast near-line storage and an on-line disk system as the front-line storage. They also propose some tape-scheduling algorithms. Wang *et al.* present techniques for managing disks as a buffer for the tertiary storage of multimedia servers. They propose a new staging technique called SEP in [Wang96] and extend it so called BiHOP in [Wang97c].

G. Real-time scheduling support

Since video data has periodic nature, the existing real-time scheduling theory can be applied to multimedia applications. Tindell *et al.* [Tind93] apply their fixed priority preemptive scheduling theory to multimedia disk traffic for the guaranteed retrieval. Recently, Mok and Chen [Mok96] propose a new multiframe model for the task of which the execution time varies from one instance to another, and apply it to multimedia streams. Kaneko and Stankovic [Kane96] give an integrated scheduling of multimedia and hard real-time tasks. Hamdaoui and Ramanathan [Hamd95] introduce the notion of (m, k)-firm deadlines and propose a distance-based priority assignment technique to reduce the probability of dynamic failures which indicate if fewer than m out of any k consecutive instances meet their deadlines. In [Han95], by raising the priority of the urgent frames and pre-scheduling them with the backwards-EDF algorithm, the urgent frames can meet their deadlines and the normal frames have more room for their execution. We propose a simple but efficient scheduling scheme for multimedia streams with firm deadlines using heuristic functions in [Cho97d].

2.2 Related work

There exist a lot of works related to video servers. In this section, we introduce some well-known works on video servers classified by the issues which this thesis concerns.

2.2.1 Storage and retrieval in a single server

Early works on video servers are founded on a single disk or multiple disks in a single server architecture. Rangan and Vin [Rang91a, Rang91b, Rang92, Vin93] present a model that relates disk and device characteristics to the recording rate, and derive storage granularity and scattering parameters that guarantee continuous access. They also develop admission control algorithms in order for the server to support multiple concurrent clients. A prototype multimedia file system is implemented, in which policies and algorithms for video storage are experimented.

Anderson *et al.* [Ande92] develop a Continuous Media File System (CMFS) to support real-time storage and retrieval of continuous media data (digital video and audio) on a single disk. CMFS addresses several interrelated design issues: real-time semantics of sessions, disk layout, an admission control algorithm for new sessions, and disk scheduling policy.

Lougher and Shepherd [Loug92, Loug93] describe the design of a file server specially optimized for the storage and retrieval of continuous media including disk striping, optimized disk layouts, real-time algorithms, and disk head scheduling. They also implement a prototype of continuous media storage server equipped with multiple disks.

Starlight Networks Inc. introduces a product for a single server equipped with disk arrays. In [Toba93], Tobagi *et al.* describe a video applications server software focusing primarily on its underlying storage management system. The system manages an array of disks and uses a disk access algorithm particularly suitable for video streaming. They also characterize the performance of the system by determining the number of streams that can be supported for a given memory size and a given service latency requirement.

Huynh and Khoshgoftaar [Huyn94] take an engineering approach and give an extensive performance analysis of the subsystem control block architecture of IBM-PC and disk array technology in typical video server environments. They reveal that, with one video data stream, the five-disk RAID-5 array is a little bit better than the non-RAID, four-disk system. However, when the video server has to support multiple, simultaneous video data stream, the token ring network becomes the system bottleneck, so there is not much difference between RAID-5 and non-RAID.

Bell Lab. implements a multimedia storage system $\mathcal{F}$*ellini* by providing the software APIs for it. In [Chun96], Martin *et al.* describe the architecture of $\mathcal{F}$*ellini*. $\mathcal{F}$*ellini* supports the storage and retrieval of both continuous media data as well as conventional data such as text, binary, and image. They argue that the algorithms for retrieving data from disks provide high throughput by reducing the seek latency time and that the buffer management scheme exploits the sequential access patterns for continuous media data in order to determine the buffer pages to be replaced from the cache.

2.2.2 Storage and retrieval in a large-scale server

The IBM Almaden research center implements a network file server *Shark* for digital video and other continuous media data [Hask93]. *Shark* scales from small desktop machines to the SP-2 parallel supercomputer. *Shark*'s primary features are support for continuous-time data, scalablility, high availability, and manageability, all of which are crucial in its role in large-scale video servers.

Ghanderharizadeh and Ramos [Ghan93] describe a parallel multimedia information system and the key technical ideas that enable it to support a real-time display of multimedia objects. They adopt a shared-nothing architecture, so that the client stations are independent of the backend processors that contain multimedia data. In order to support simultaneous retrieval of an object for different clients, they suggest two alternative approaches (disk multitasking and data replication) and investigate the tradeoffs associated with each approach using a simulation model.

In order to support access to all types of conventional data stored in Oracle relational and text databases, Orcale develops an Oracle Media Server providing consumer based interactive access to multimedia data [Laur94]. The media server supports storage and playback of real-time audio and video data. The server provides a platform for distributed client-server computing and access to data over asymmetric real-time networks. A service mechanism allows applications to be split such that client devices can focus on presentation, while backend services running in a distributed server complex provide access to data via messaging or lightweight RPC.

Freedman and DeWitt [Free95] perform a detailed simulation analysis of the SPIFFI scalable video-on-demand system. They introduce and analyze the performance of video server algorithms for real-time disk scheduling, page replacement, and prefetching, and show that the love prefetch page replacement and delayed prefetching algorithms substantially reduce the memory requirements, and thus, reduce the cost of a video server. They also demonstrate that while the non-real-time elevator disk scheduling algorithm can function well in a relatively small video server (16 disks) with plenty of memory at the terminals, it does not scale to larger systems.

Bell Lab. develops a distributed multimedia server *Calliope* constructed from personal computers. In *Calliope*, each PC provides independent service for client; *i.e.* a video file is not striped across multiple nodes. Heybey *et al.* [Heyb96] show from their preliminary performance measurements that *Calliope* can be scaled from a single PC producing about 22 MPEG-1 video streams to hundreds of PCs producing thousands of streams. They argue that *Calliope* is cost-effective because it requires only commodity hardware and portable because it runs under Unix. In similar architecture to *Calliope*, Kim *et al.* [Kim97a] design and implement a scalable, multi-purpose multimedia-on-demand system.

Reddy [Redd95] addresses the problem of distributing and scheduling videos on a multiprocessor video server and the issue of communication scheduling over the multiprocessor switch for the playback of the scheduled videos. The proposed solution minimizes contention for links over the switch and makes video scheduling very simple. He exploits the network topology of the multiprocessor to derive such a sequence that guarantees freedom from communication conflicts.

Microsoft Corp. develops a distributed, fault-tolerant real-time file server *Tiger* [Bolo96] which provides data streams at a constant, guaranteed rate to a large number of clients, in addition to supporting more traditional file system operations. *Tiger* runs on a collection of personal computers connected by an ATM switch. Bolosky *et al.* discuss that the fundamental problem of the design of *Tiger* is that of efficiently balancing user load against limited disk, network, and I/O bus resources. They also argue that *Tiger* accomplishes this balancing by striping file data across all disks and all computers in the distributed system, and then allocating data streams in a schedule that rotates across the disks.

Tewari *et al.* [Tewa96a] investigate the suitability of clustered architectures for designing scalable multimedia servers. Specifically, through an analytic model of clustered multimedia servers, they evaluate the effcts of architectural design of the cluster, the size of the unit of data interleaving, and read-ahead buffering and scheduling on the real-time performance guarantees provided by the server. They also implement a prototype based on the results of their analysis.

The IBM Watson research center designs and implements a scalable collection of heterogeneous multimedia servers (Research Multimedia Server Complex) which uses a variety of communications protocols and network types to deliver multimedia objects to clients [Dan97]. The Research Server Complex Manager (RSCM) provides a uniform external interface to applications hiding the heterogeneity and making the server complex appear as individual requests. Dan *et al.* justify the particular functions of the RSCM and explains the design decisions and tradeoffs.

Buddhikot *et al.* [Gros97] suggest the Massively-parallel And Real-time Storage (MARS) architecture for the design and prototype implementation of a large-scale video server. MARS exploits some of the well-known techniques in parallel I/O, such as data striping and an innovative ATM based interconnect inside the server to achieve a scalable architecture that transparently connects storage devices to an ATM-based broadband network. The ATM interconnect within the server employs a custom ASIC called ATM Port Interconnect Controller (APIC). The architecture relies on innovative data striping and real-time scheduling to allow a large number of guaranteed concurrent accesses and uses separation of meta data from real data to achieve a direct flow of the media streams between the storage devices and the network. They argue that the system architecture is scalable in terms of the number of supported clients and the throughput.

2.2.3 Storage and retrieval in a multi-resolution video server

Although a number of works have been done on multi-resolution or scalable coding of video data [Chia94, Laza94, Lian97, Tan96, Wang97a] and on transmission of multi-resolution video in communication network [Delg94, Hunt, Mc96], a relatively small number of works address the issue on multi-resolution video servers.

Chiueh and Katz [Chiu93] employ the specific multi-resolution video representation coded in a Laplacian or Gaussian pyramid and lay out video data on a two-dimensional disk array. The result of a simulation study shows that under synthetic workload the multi-resolution scheme performs significantly better in terms of I/O rate, average waiting time, and average physical data bandwidth requirement as compared with full-rate single resolution video.

Keeton and Katz [Keet93] propose a systems approach to providing video service which integrates the multi-resolution data generated by scalable compression algorithms with the high-bandwidth, high-capacity storage provided by disk arrays. They argue from their simulation results that the storage of multi-resolution video permits service to considerably more clients than the storage of single-resolution video and that retrieval of data striped across the disks of an array can be performed much more efficiently than retrieval from a single disk.

Chang and Zakhor [Chan94, Chan96a, Chan97] consider the placement of scalable video data on single and multiple disks for storage and retrieval. For the single-disk case, they explore the principle of constant frame grouping from scalable video data. They also examine the qualities of video reconstructions obtained from a real disk video server and find the scalable video more visually appealing.

Considering the multiple disk scenario, they prove that periodic interleaving results in lower system delay.

Paek *et al.* [Paek95] present a flexible data placement strategy for independent parallel disk arrays. The trade-off between utilization efficiency and interactive delay is investigated for the data placement strategy. Based on the trade-off, they show the advantage of video servers supporting a range of interactivity QoS. They also argue that using scalable video improves the utilization and interactivity performance of a video server. They use three-layer scalable MPEG2 digital video to support resolution QoS.

Chen *et al.* [Chen95] suggest an idea of staggering scalable data blocks considering video data corresponding to different rates of the video clip are not required to reside in the same disk. On the basis of the idea, they propose and explore the approach of rate staggering, *i.e.* staggering video data in the disk array based on data rates. They argue that the advantages of the proposed rate staggering method include: (1) minimizing the intermediate buffer space required at the server, (2) achieving better load balancing due to finer scheduling granularity, and (3) alleviating the disk bandwidth fragmentation. However, their argument about (1) has some technical flaws.

Shenoy and Vin [Shen95, Shen98] present a placement algorithm that interleaves multi-resolution video streams on a disk array and enables a video server to efficiently support playback of these streams at different resolution levels. They then combine this placement algorithm with a scalable compression technique to efficiently support interactive scan operations. They present an analytical model for evaluating the impact of the scan operations on the performance of disk-array-based servers. The experiments demonstrate that exploiting the characteristics of video streams and human perceptual tolerance enables a server to support interactive operations without any additional overhead.

Beckmann *et al.* [Beck98] describe an admission control policy in which the quality of service is negotiated at stream initiation and is a function of both the desired quality of service and the available bandwidth resources. They argue that the advantage of their approach is the ability to robustly service large numbers of clients, while providing increased quality of service during low usage periods. Several simple algorithms for implementing the policy are proposed and evaluated via simulation.

Chapter 3

Storage and retrieval in a single server

In this chapter, we design and implement a video server equipped with a disk array for the storage subsystem. The single server approach for video servers needs to adopt disk arrays because video servers are required to provide large storage space and transmission bandwidth. Disk arrays reveal large differences in performance according to their organization and data distribution across disks [Cata95]. This chapter analyzes the performance of disk arrays for storage architecture of video servers and implement a software-based disk array for the video server on the basis of the performance analysis.

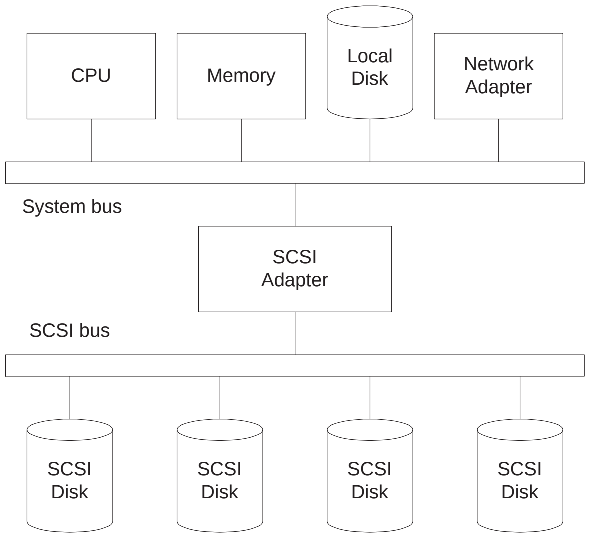


Figure 3.1: Architecture of a single server

3.1 System model

First of all, we begin by taking a small-scale single server approach in this chapter. The server is based on a personal computer (PC) equipped with a SCSI adapter and multiple SCSI hard disk drives. The server architecture is shown in Figure 3.1. The server consists of processor, memory system, system bus, disk subsystem, and network subsystem. A SCSI adapter and SCSI hard disk drives comprise the disk subsystem and we take a software-based approach for the disk array subsystem. We assume that meta data for video streams are stored on a separate local disk in order to guarantee the large bandwidth of disk array subsystem. On the other hand, the network subsystem may be another performance bottleneck of the video server, this problem is not considered throughout the thesis assuming that the network subsystem provides large bandwidth enough to support the transmission of video streams.

The major mission of video servers is to convey data through I/O path: disk to network. Hence, the CPU utilization is not so high. Although CPU controls disk arrays, the tasks include distributing client requests to disks, scheduling requests on each disk queue, issuing commands to SCSI adapter, and sending the retrieved data to clients via network subsystem. So, we can find that CPU does not cause the performance bottleneck in video servers. In addition, since the bandwidth of system bus is far larger than that of SCSI bus, we conclude that the performance bottleneck of video servers occurs in the disk subsystem [Cho94].

Disk arrays provide large bandwidth by activating multiple disks concurrently or in parallel. Multiple disks in a disk array are connected to the system through a SCSI bus which may cause the performance bottleneck. The trends of SCSI disk and bus technologies show that the bandwidth of SCSI bus can support four disks analytically without any performance bottleneck in the bus[1]. So, we consider only up to four disks in a SCSI bus. On the other hand, although a hardware-based disk array controller may be used, we take the software-based approach as mentioned earlier because it is flexible to implement disk arrays suitable for video servers.

3.2 Performance analysis of disk arrays

There exist two straightforward strategies which explore different aspects of parallelism and concurrency offered by striping data across disks [Keet93]. The degree of concurrency is defined as the number of outstanding request at one time and the

[1]The current transfer bandwidth of a SCSI disk is about 10MB/s while the wide SCSI bus supports 40MB/s.

parallelism describes the number of disks that service a single request. Among different levels of RAID (Redundant Arrays of Inexpensive Disks), RAID3 exploits the parallelism while RAID4 and RAID5 do the concurrency. We consider only data striping: redundancy such as parity for on-line recovery is beyond the scope of the thesis and is not described further. So, we call the 'parallelism' scheme and 'concurrency' scheme, AID3 and AID5, respectively.

In AID3, a request is serviced on all the disks while all the disks in the array are synchronized. In video servers, however, since client requests arrives periodically, the synchronization unit in AID3 may be the period of client requests. So, AID3 can be implemented by software-based approach without difficulty. On the contrary, in AID5, a request is serviced on a single disk but multiple requests may be serviced across disks concurrently. Therefore, scheduling support is required to evenly distribute requests across disks.

The major performance metric of video servers is the number of clients that can be serviced simultaneously as mentioned in Chapter 1. Since we identified that the performance bottleneck of video servers is the disk subsystem in Section 3.1, the number of clients that can be serviced simultaneously depends upon the performance of disk subsystem.

Before analyzing the performance of disk subsystem, we first describe the basic behavior of video servers. Let us assume that client i ($1 \leq i \leq N$) requests a video stream V_i. Then, the server must retrieve N disk blocks and transmit them to clients periodically. The period is called *round*. The round length T_{round} is determined by the playback rate of V_i ($1 \leq i \leq N$) and the disk block size. In this chapter, we assume that the playback rate of V_i ($1 \leq i \leq N$) is equal to each other for convenience; so, $T_{round} = T_{play}$, where T_{play} denotes the playback time of a disk block. This assumption is broken in the next chapter.

First, we analyze the performance of AID5 in terms of the number of clients that can be serviced simultaneously in video servers. Video streams are divided into striping units and are striped across disks. The size of client requests is the same as the striping unit or one disk block. From the view point of each disk, disk requests for a client arrive with the period of d rounds in AID5, where d denotes the number of disks in the array, because a request is serviced in a disk and the next request is serviced in the adjacent disk, or in round-robin manner. Thus, during d rounds, total N disk blocks must be retrieved in each disk. This condition can be written by

$$T_{overhead} + N \times \frac{B_{AID5}}{R} \;\leq\; d \times T_{play}^{AID5}, \tag{3.1}$$

where R denotes the transfer rate of a disk and B is the disk block size. $T_{overhead}$ represents the time for the disk head to arrive at the desired position including seek time and rotational latency. If the disk block size is determined by the unit of track, or given by multiple tracks, the rotational latency can be ignored due to the track buffer in disk drives. The SCAN disk scheduling minimizes $T_{overhead}$ because all the requests can be serviced while the disk head moves to one direction. Since recent disks reveal non-linear seek time for the seek distance, however, it is difficult to estimate the exact value of $T_{overhead}$. From Eq. (3.1), we obtain

$$N \;\leq\; \frac{T_{play}^{AID5} \cdot d - T_{overhead}}{B_{AID5}/R}. \tag{3.2}$$

When $d = 4$, $B = 36KB$, $R = 2.4MB/s$, and $T_{play} = 192$ms[2], for example, the shaded region in Figure 3.2 represents the feasible values for N and $T_{overhead}$. In

[2]We assume 1.5Mbps MPEG-1 video streams.

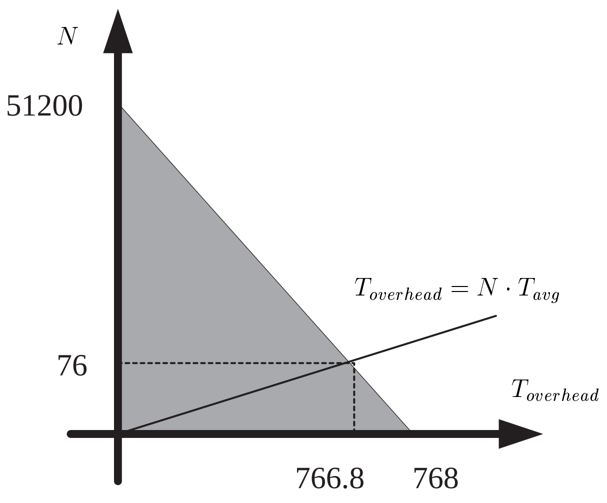


Figure 3.2: Relationship between $T_{overhead}$ and N

Figure 3.2, assuming that the average overhead for one disk request is 10ms and thus $T_{overhead} = 10N$, we get $N = 76$ when $T_{overhead} = 766.8ms$. If we assume that the average overhead for a disk request is 15ms, $N = 51$ is obtained while $T_{overhead} = 767.2ms$.

On the other hand, in AID3, since a disk block for client request is distributed across all the disks, the striping unit is $1/d$ of a block, or $\frac{B_{AID3}}{d}$. Then, from the viewpoint of each disk as similar in AID5, the server must retrieve N striping units for each round yielding:

$$T_{overhead} + \frac{N}{d} \times \frac{B_{AID3}}{R} \leq T_{play}^{AID3}. \tag{3.3}$$

In Eq. (3.1) and Eq. (3.3), if we let the striping unit sizes of two schemes be the same, we obtain $B_{AID3} = d \cdot B_{AID5}$ and $T_{play}^{AID3} = d \cdot T_{play}^{AID5}$. Applying them to Eq. (3.3), Eq. (3.1) becomes to be equivalent to Eq. (3.3). This indicates that the performance of two schemes are equal to each other when the striping unit sizes are given by the same value. In that case, however, since the block size of AID3 is d times larger than that of AID5, the buffer requirement of AID3 is much larger than that of AID5. On the condition that the disk block sizes of two schemes are equal, AID5 performs better than AID3. As compared with Figure 3.2, for instance, AID3 can provide services for only 19 concurrent clients assuming the average overhead for a disk request is 10ms.

The simple analysis given above reveals that the disk seek time and rotational latency, or $T_{overhead}$ greatly affect on the performance of disk arrays. In order to reduce the left-term of Eq. (3.1) and Eq. (3.3), we should employ SCAN disk scheduling algorithm and the disk block size s should be large enough. As mentioned earlier, the striping unit should be multiple tracks, so that the rotational latency is ignored due to the track buffer of disk drives. The larger the disk block size, the larger the buffer requirement. So, the cost-effective disk block size should be determined. We explore components which affect on the performance of video servers through simulation in the next section.

3.3 Simulation studies

Figure 3.3 depicts the simulation model. During each round with the period of T_{round}, total N disk block requests are generated for N clients and they are distributed across disks according to the striping policies. Each disk scheduler reorders the requests by SCAN disk scheduling algorithm and services them. All the requests should be serviced during a round of T_{round}. Simulation proceeds by increasing N and determines the maximum number of streams that can be serviced simultaneously. T_{round} is calculated by the disk block size assuming the

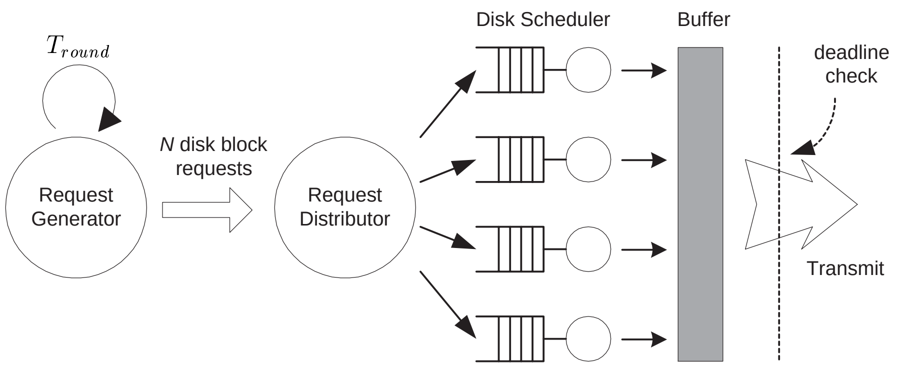


Figure 3.3: Simulation model

video stream rate is given by 1.5Mbps. For the purpose of modeling the behavior of disk drives exactly, we employ a study of Ruemmler and Wilkes [Ruem94] which analyzes the characteristics of modern disk drives. We choose HP 97560 disk drives for the simulation of which the performance parameters are given in Table 3.1.

The major concern is how the disk block size and striping techniques affect the performance of video servers. We first compare the performance of striping techniques: AID3, AID5, and the hybrid technique of AID3 and AID5. We also consider 'no striping' scheme in which data are not striped across disks but are stored in disks sequentially. Figure 3.4 shows the results. As analyzed in Section 3.2, the performances of AID3, AID5, and the hybrid scheme do not reveal large differences but the absolute values of the maximum number of streams that can be serviced simultaneously show some differences with those of the analysis performed in the previous section. This occurs because we underestimate $T_{overhead}$ and include additional overhead such as controller overhead and head switch time in the simulation.

Table 3.1: Disk parameters used in the simulation (HP 97560)

Capacity	1.3 GB
Cylinders	1,962
Tracks per cylinder	19
Track size	36 KB
Revolution speed	4,002 RPM
Seek time	$3.24 + 0.400\sqrt{d}\ (0 < d \leq 383)$
	$8.00 + 0.008d\ (383 < d \leq 1962)$
Controller overhead	2.2 ms
Head switch time	1.6 ms

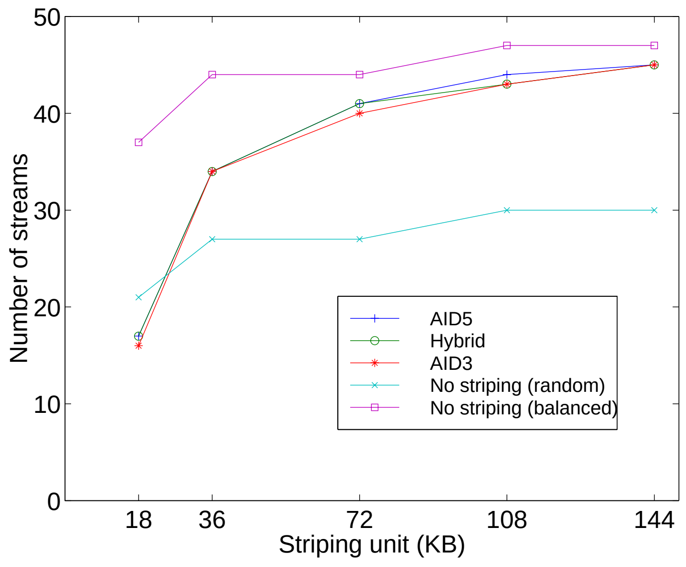


Figure 3.4: Comparison of striping policies

Although AID3, AID5, and the hybrid striping techniques show similar performance when the striping unit is equal to each other, the buffer requirement of AID5 is far smaller than those of the other schemes, because the disk block size of AID5 is the smallest. In addition, the large disk block leads to the large T_{round} and thus results in the large start-up latency for the service. We can conclude that AID5 is suitable for the storage architecture of video servers.

Figure 3.4 also reveals that the number of streams increases as the striping unit size increases. It is noteworthy that allocating one more track for striping unit increases the performance significantly. This is because the large striping unit reduces the portion of $T_{overhead}$ in Eq. (3.1) and Eq. (3.3). However, when the striping unit is greater than three tracks, the performance gain reduces. Therefore, it can be concluded that the best choice for the striping unit would be 1 or 2 tracks.

In Figure 3.5, the effects of the number of disks are presented when AID5 is applied. Figure 3.5 reveals the linear increase in the number of streams as the number of disks increases. However, as mentioned earlier, 5 or more disks may cause a bottleneck in SCSI bus.

Finally, we explore the placement policy in a disk. As compared with the random allocation scheme, the contiguous allocation scheme is simple to implement and requires no meta data for video streams but may cause the fragmentation in disk storage space when video streams are created, edited, and deleted frequently. Figure 3.6 compares the performance of two schemes. The contiguous allocation scheme performs better slightly. In VOD application where read operations are dominant and thus no fragmentation occurs, the contiguous allocation scheme is competitive approach for the placement policy.

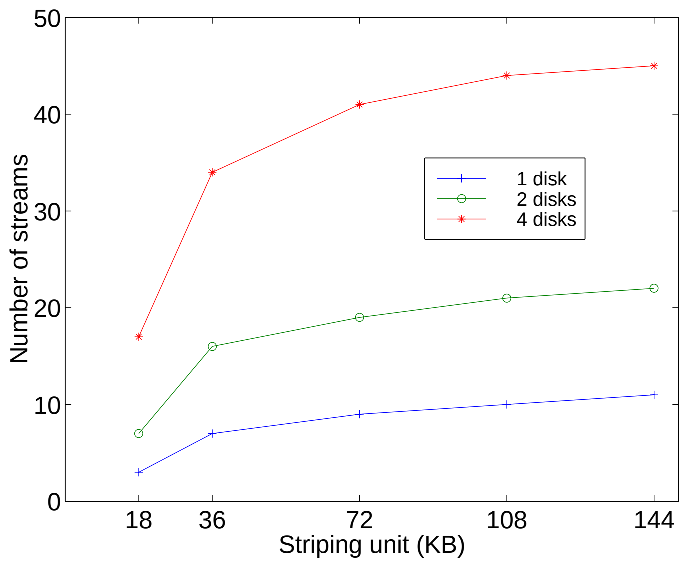


Figure 3.5: Effect of the number of disks (AID5, $s = 36KB$)

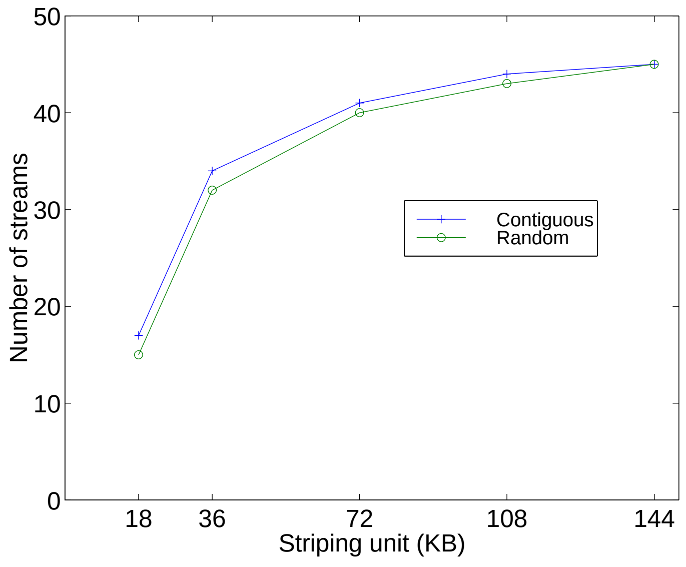


Figure 3.6: Comparison of placement schemes (AID5, $s = 36KB$)

Table 3.2: The proposed storage architecture for a single server

Number of disks	4
Disk scheduling	SCAN
Disk striping	AID5
Striping unit	$1 \sim 2$ tracks
Placement policy	contiguous

In summary, we suggest a storage architecture for video servers equipped with disk arrays in Table 3.2 from the analysis and simulation conducted in Section 3.2 and Section 3.3. We implement the proposed storage architecture in the next section.

3.4 Implementation of a disk array manager: DAman

On the basis of the studies conducted in the previous sections, we implement a disk array for video servers and measure its performance in this section. As mentioned in Section 3.1, Figure 3.1 depicts the hardware platform of the server. The server is a Pentium 100MHz PC equipped with a AHA-1540CP SCSI adapter and four Quantum 850MB SCSI disks. We choose the QNX real-time microkernel operating systems [QNX93] for the software platform and implement a disk array manager, which we name DAman[3], as a server process on QNX. Since QNX is a microkernel OS, device drivers run as server processes[4]. So, it is convenient to develop and debug device drivers. In addition, QNX supports real-time facilities

[3]Source codes for DAman can be accessed at `http://cselab.snu.ac.kr/~cjs/proj/DAman/DAman.html`.

[4]The priorities of device drivers is higher than those of application programs.

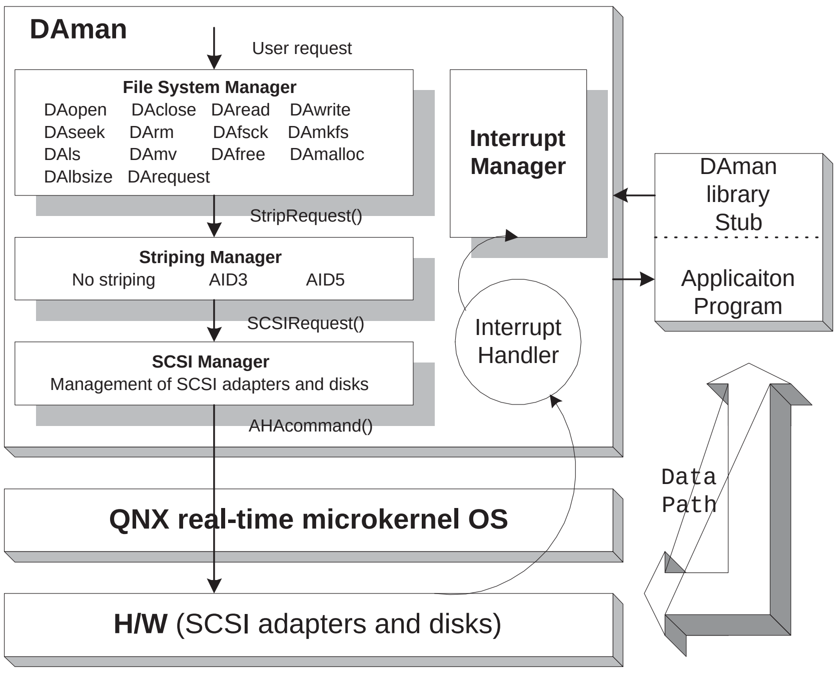


Figure 3.7: Overall architecture of DAman

including prioritized message and process scheduling. Consequently, we believe that QNX is an appropriate OS for DAman and video servers.

3.4.1 Overall architecture

DAman consists of four functional managers: file system manager, striping manager, SCSI manager, and interrupt manager, as shown in Figure 3.7. By implementing functional managers, it is easy to add, update, and delete each function. However, the functional managers are not implemented as independent processes, because QNX does not support thread facilities. If the managers are implemented as independent processes, large overhead of IPC may lead to low performance. DAman is invoked from messages given by applications or by interrupt handler.

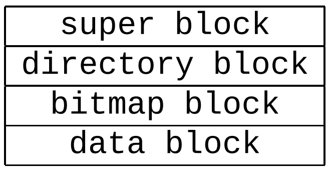


Figure 3.8: DAman file system structure

Since the messages of interrupt handler have higher priorities than those of applications, the interrupt manager has the highest priority among the functional managers.

The control flow between each managers is also depicted in Figure 3.7. Application programs are linked with DAman libraries and the library stub sends messages to DAman for data service and receives results. The DAman library stub packs a message for the given system call and send it to DAman. The file system manager processes the message and hand it over to the striping manager by `StripRequest()` function call. Then, according to the striping policy, the striping manager packs a SCSI request and call `SCSIRequest()`. the SCSI manager controls SCSI devices including adapters and disks through `AHACommand()`. When a SCSI request service is done, SCSI adapters notify it to CPU by interrupt. The interrupt handler invokes DAman for the interrupt processing.

DAman provides a separate file system from QNX of which the structure is given in Figure 3.8. Since DAman is designed for video servers, DAman file system is optimized on continuous read operations. That is, it has no index structure such as `inode` in UNIX while each block of a video file is placed contiguously. So, all the blocks can be accessed directly. In Figure 3.8, the super block contains hardware information including the number of SCSI adapters, the number of

```
int           da_open(char *file_name, int flag);
int           da_close(int handle);
int           da_read(int handle, char __far *buf, int max_size);
int           da_write(int handle, char __far *buf, int size);
int           da_request(int n, Req_Blk_t *req_blk);
int           da_rewind(int handle);
int           da_mkfs(int logical_block_size);
int           da_fsck(void);
int           da_lbsize(void);
int           da_ls(DirEnt_t *dir, char *name);
int           da_rm(char *file_name);
int           da_mv(char *source, char *dest);
char __far *  da_malloc(int size);
int           da_free(char __far *pointer);
char *        far2near(char __far *pointer, int size);
int           da_errmsg(void);
```

Figure 3.9: Run-time libraries for DAman

SCSI disks, IRQ number, base address for adapters, etc. It also has the striping type. DAman initializes itself with the information stored in the super block. The directory block stores directory entries for each video file such as file name, file size, created time, and owner. The bitmap block indicates whether each block is allocated or not. It is used on compaction and file system check operation.

The file system manager handles the logical address space on file systems while the mapping from logical block to physical block is performed by the striping manager. The striping manager supports the following three policies: no striping, AID3, and AID5. The SCSI manager and the interrupt manager control the SCSI adapters and disks. The SCSI manager packs SCSI commands from the information given by the striping manager and send them to the adapter. The interrupt manager is invoked by proxy messages which are delivered from interrupt handler when SCSI commands are done.

Table 3.3: Utilities for DAman

Command	Description
`cp2da`	copy a file to DAman
`cpda2`	copy a file from DAman
`mkdafs`	make a DAman file system
`dals`	list directory entries
`darm`	remove a file
`damv`	move (rename) a file
`dafsck`	check DAman file system

As mentioned earlier, the disk blocks can be accessed directly without index structure and are placed contiguously. This is because DAman is designed for video servers in which read operations are dominant. However, when video files are deleted, inserted, and updated frequently, the fragmentation in file system address space may occur [Vin95]. We believe that this problem does not occur in video servers and can be solved simply by replacing only the file system manager with a new one. Another optimization for read operations is that DAman has no buffer cache. In other words, data are transferred to user address space directly through DMA mechanism. Hence, there is no data copying overhead from kernel address space to user address space which takes large portion in read operations of video servers [Coul94]. The performance gain of buffer cache in video servers is not so large unless multiple clients arrive within a short time interval [Dan95].

DAman supports 14 system calls as shown in Figure 3.7. Based on them, 16 run-time libraries are given for applications in Figure 3.9. We also provide some utilities for the maintenance of DAman in Table 3.3. When the fragmentation problem in DAman is severe, we can eliminate it by `dafsck` command. Finally, we develop a video-on-demand system by integrating DAman with a video server

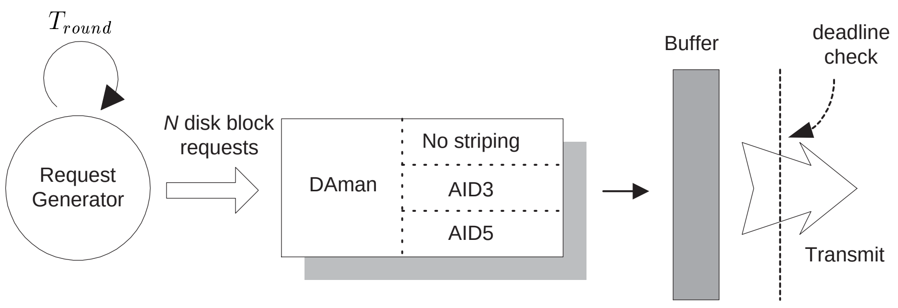


Figure 3.10: Experiment model

given in [Ahn95]. The prototype demonstrates that DAman well supports multiple concurrent streams[5].

3.4.2 Empirical evaluation

In this subsection, we empirically evaluate the performance of DAman. Figure 3.10 depicts the experimental model similar to that of the simulation study given in Section 3.3. A request generator is an application program linked with DAman libraries. It generates N disk requests periodically with the period of T_{round}. DAman stores 20 video files each of which has 150MB size. We assume that the stream rate of a video stream is 160KB/s[6]. Since T_{round} is calculated from the block size and the stream rate of video streams, T_{round} is given by 0.1 second when the disk block size is 16KB. We derive the number of streams that can be serviced simultaneously by measuring the number of disk blocks which can be retrieved within a round, or T_{round}. Meanwhile, we measure the elapsed time using

[5]Due to the lack of hardware MPEG-1 decoders, we have ascertained two concurrent streams to be played back hiccup-free in the prototype. It is expected, however, that DAman can support as many as concurrent streams presented in the next subsection through simulation.

[6]$\approx$ 1.5Mbps.

Table 3.4: Execution time of DAman (μs)

Striping policy	No striping	AID3	AID5
Request to DAman	132.7	130.8	130.0
File system manager	23.3	22.7	25.6
Striping manager	14.9	26.3	15.1
SCSI manager	17.6	25.7	18.7
Interrupt manager	109.2	407.8	107.2

a timer board which has 0.1μs granularity.

First, Table 3.4 represents the execution time of each functional manager of DAman. The execution time is much less than the disk access time which is tens of milli-seconds. We can find from Table 3.4 that the time for request to DAman and the execution time of the interrupt manager are relatively large. This is because some messages are passed between an application to DAman, and between the interrupt handler to DAman, respectively. It is well known that message passing leads to large overhead in micro-kernel operating systems.

As for the comparison of striping policies, while ‘no striping’ and AID5 reveal the similar execution time to each other, the execution times of the striping manager, the SCSI manager, and the interrupt manager in AID3 are larger than those in ‘no striping’ and AID5. This occurs because a logical block service in AID3 is divided into four requests to each disk which should be handled in the striping manager, the SCSI manager, and the interrupt manager. The total service time for a disk access in AID3 is expected to be far larger than those in ‘no striping’ and AID5 because: (1) the service time depends on the largest one among all the disks, and (2) a logical block is divided into multiple physical disk blocks yielding relatively large portion of disk seek time.

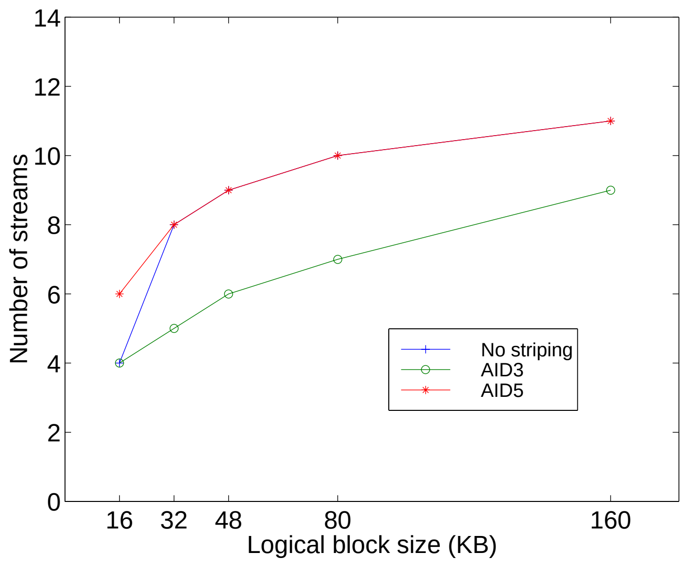


Figure 3.11: Comparison of striping policies (4 disks)

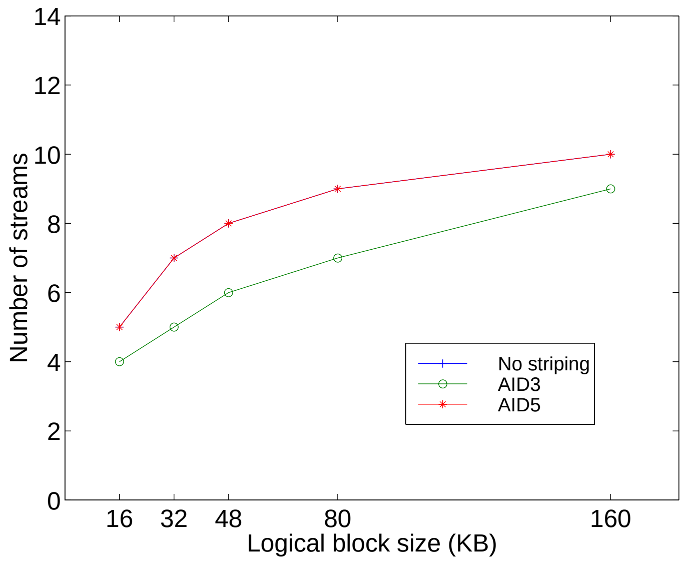


Figure 3.12: Comparison of striping policies (2 disks)

Figure 3.11 reflects the analysis described above. Given a logical block size, AID3 provides the smallest number of concurrent streams. One more thing to point out in Figure 3.12 is that the number of streams increases as the logical block size increases. This is a straightforward result taking disk seek time overhead into account. However, the increament ratio decreases when the logical block size is greater than two tracks[7]. This coincides with the simulation studies in Section 3.3. The similar results are derived in Figure 3.12 when only two disks comprise the disk array.

Observe in Figure 3.11 and Figure 3.12 that the performance of ‘no striping’ scheme is equivalent to AID5. This occurs because we let the disk workload be evenly distributed across disks, which may not occur in real world. AID5 itself evenly distributes disk workload across disks. In addition, Figure 3.13 reveals that the average service time of AID5 is less than that of ‘no striping’ scheme. Consequently, AID5 is the most promising architecture for video servers, which also coincides with the simulation studies.

Figure 3.14 shows the effect of the number of disks in the array. Although the number of streams increases as the number of disks increases, the linear increase is not obtained due to the contention on system bus and/or SCSI bus, which is not reflected on the simulation studies. In addition, the actual number of streams that can be serviced in DAman shows large differences from that of simulation studies. This indicates that: (1) the actual disk access time including seek time, rotational latency, and transfer time is greater than the analytic value, and (2) the other overhead of DAman takes large portion of the total service time. Nevertheless, the

[7]The track size of our disk is 32KB.

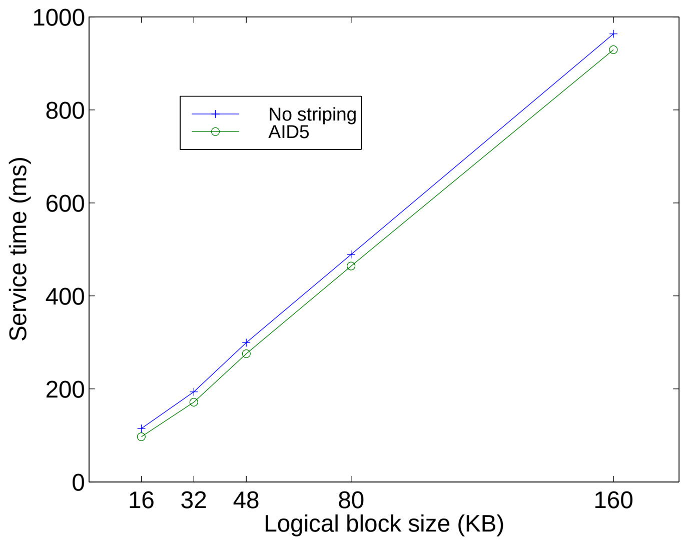


Figure 3.13: Comparison between no striping and AID5 (4 disks)

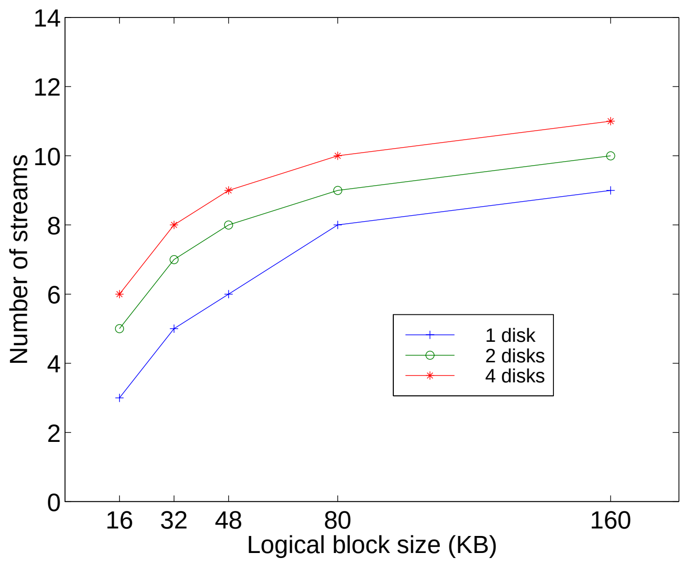


Figure 3.14: Effect of the number of disks (AID5)

shapes of the graphs (*i.e.* the tendency of the effects of parameters on performance) are similar; the proposed storage architecture in Table 3.2 obtained by the simulation studies in Section 3.3 is validated through the performance evaluation of DAman.

Chapter 4

Storage and retrieval in a large-scale server

The single server approach in Chapter 3 has limitations in scalability [Lee98]. For the purpose of providing video services for the public over high-speed network, a video server should store thousands of video streams and serve tens of thousands of concurrent clients. Assuming that MPEG-1 video streams require a playback rate of 1.5Mbps, a 100-minute long video requires about 1.1GB storage, and two thousand videos require a capacity of 2.2TB, or 220 disks of 10GB. If we exploit the perfect parallelism or concurrency of disks in such a system and assume that the effective bandwidth of a disk is 10MB/s, 11733 clients can be serviced simultaneously. Such a large-scale server should be designed and implemented based on parallel video server architecture. The parallel server consists of storage nodes which store and provide video data and network nodes which deliver data to clients in a timely fashion. The storage and network functions may reside in a node or be isolated physically in different nodes. The storage node should be equipped with high-performance storage subsystems such as disk arrays. Communication between nodes also demands high bandwidth interconnections.

In this chapter we address the problems in designing a large-scale video server which consists of a large number of nodes connected by a high performance interconnection network. The design problems narrow down to how to cluster such nodes into parallel servers and how to distribute and schedule video streams in a parallel server.

4.1 System model

As similar to the single server equipped with disk arrays in Chapter 3, each video stream can be divided into logical blocks and then distributed among multiple storage nodes, which referred to as *data-striping* in this chapter. Data-striping implicitly achieves higher disk bandwidth and load balancing [Tewa96a]. In a distributed server, however, a video stream may be stored and serviced in a single storage node [Heyb96], which referred to as *no-striping*.

Although the data-striping technique has the aforementioned advantages, it has the following disadvantages: First, a distributed scheduling among storage nodes is required. This imposes clock synchronization problems among all nodes in a parallel server. Second, service latency is relatively larger than the no-striping case on account of scheduling problems. Service latency means the time elapsed since a request is made to initiate a new stream until the stream is serviced. Third, data-striping lacks scalability. If disks or storage nodes are added to a parallel

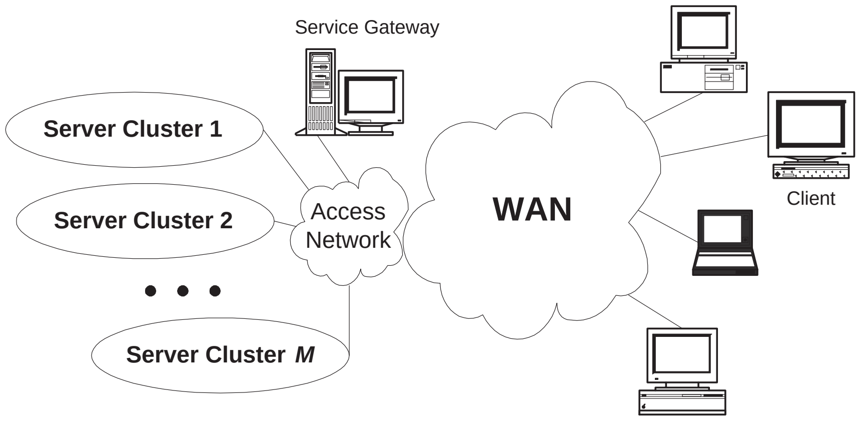


Figure 4.1: Configuration of a large-scale video server

server, the whole data must be redistributed among storage nodes. Finally, the popularity of video streams cannot be considered. Since there is no improvement in performance when more than one copy of the same video is placed on a disk [Litt93], replicating popular videos in a parallel server fails to increase the number of clients that can be serviced simultaneously. Hence, a hybrid technique of data-striping and no-striping is required.

For a given set of nodes, we divide it into server clusters as shown in Figure 4.1. Considering the advantages of data-striping, video streams are striped across all the storage nodes in a server cluster while a server cluster has the parallel server architecture[1]. Video streams are allocated and replicated among server clusters with respect to their popularity. Server clusters provide video streams for clients independently. A series of video streams stored in server clusters is serviced at a service gateway. Considering the load balance among server clusters, the gateway provides clients with the address of the server cluster that stores the requested

[1]In Chapter 4, the terms server cluster and parallel server are used interchangeably.

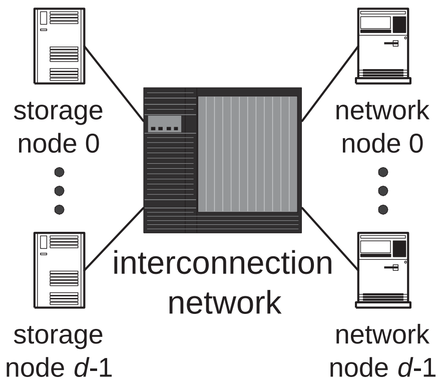


Figure 4.2: Architecture of a parallel server (server cluster)

video. For a given set of nodes, the appropriate number of server clusters or the number of nodes within a server cluster (the size of a server cluster) will be described in Section 4.3. If the size of a server cluster equals the number of nodes in the server, video streams are striped across the server, that is, data-striping occurs in the server. On the other hand, if the size of a server cluster is equal to one, no-striping scheme is employed in the server.

Figure 4.2 shows the architecture of a server cluster (parallel server) which consists of *storage nodes* and *network nodes*. Storage nodes are responsible for storing video data and delivering the required bandwidth to this data while network nodes are for delivering data blocks from storage nodes to clients. Each request stream would originate at one of the network nodes in the server cluster. This network node should deliver the video stream without violating the continuity requirement of the stream. Although storage and network functions can reside on the same node by connecting node i to input i and output i of the network, we will treat storage nodes and network nodes separately in the rest of this chapter.

We consider multistage interconnection networks (MIN) for interconnecting storage nodes and network nodes. There exist static connection networks and dynamic connection networks for the interconnection networks. Static networks such as mesh, torus, and hypercube networks are suitable for building computer systems where the communication patterns are predictable or implementable with static connections [Hwan93]. While static networks are used for special-purpose applications such as scientific parallel computations, for multipurpose or general-purpose applications, we need to use dynamic connection networks which can implement all communication patterns based on program demands.

Dynamic connection networks include backplane bus systems, crossbar switch networks, and multistage networks. Bus systems and crossbar switch networks are limited to small or medium-size systems due to bus bandwidth and large cost, respectively, while multistage interconnection networks can be extended to larger systems [Hwan93]. We consider Omega interconnection network [Bern93] as the communication network between storage nodes and network nodes. However, some blocking networks are equivalent after graph transformations, so that the communication scheduling algorithm proposed in Section 4.2.3 is applicable to other multistage interconnection networks which are topologically equivalent to Omega network [Wu80] such as Baseline and Banyan networks.

4.2 Storage and retrieval in a parallel server

The parallel server architecture imposes the following problems: data distribution and retrieval scheduling at storage nodes, communication scheduling between storage nodes and network nodes, and admission control for deterministic service guarantee. The following subsections focus on these issues in a server cluster, that is, a parallel server.

4.2.1 Data placement

Data placement refers to distributing the blocks of video streams across storage nodes. This involves the order in which the blocks are striped across the storage nodes. Data organization determines bandwidth available to a video, load balance across storage nodes, and communication patterns. Since we verified in Chapter 3 that AID5 is the most appropriate storage architecture for video servers although the analysis proceeds in the single server approach, we consider only a data striping technique in which successive blocks are interleaved across storage nodes. Successive blocks of a video stream may be allocated to storage nodes either using a round-robin or a random placement algorithm [Tewa96a].

With random placement, successive blocks are placed on storage nodes using a random permutation. Although the random placement technique adapts to incremental growth, it may require more meta data and cause the load of storage nodes to be unbalanced. On the other hand, the round-robin placement scheme places successive blocks of a video stream on adjacent storage nodes and allows the streams to access storage nodes deterministically thus generating deterministic communication patterns between storage nodes and network nodes. However, this could cause large service latency if start blocks of video streams are placed at the same storage nodes. Distributing the starting points of video streams across storage nodes decreases the average service latency. Figure 4.3 illustrates an example of data placement, where $A.3$ denotes the 3rd block of stream A.

node 0	node 1	node 2	node 3
$A.0$	$A.1$	$A.2$	$A.3$
$A.4$	$A.5$	$A.6$	$A.7$
$\ldots$	$\ldots$	$\ldots$	$\ldots$
$B.3$	$B.0$	$B.1$	$B.2$
$B.7$	$B.4$	$B.5$	$B.6$
$\ldots$	$\ldots$	$\ldots$	$\ldots$
$C.2$	$C.3$	$C.0$	$C.1$
$C.6$	$C.7$	$C.4$	$C.5$
$\ldots$	$\ldots$	$\ldots$	$\ldots$
$D.1$	$D.2$	$D.3$	$D.0$
$D.5$	$D.6$	$D.7$	$D.4$
$\ldots$	$\ldots$	$\ldots$	$\ldots$

Figure 4.3: An example of data placement

We now analyze the worst case service latency. As an illustration, suppose that three clients request video streams A, B, and C respectively and that storage nodes 0, 1, and 2 serve the first blocks of A, B, and C, respectively, as shown in Figure 4.3. After the playback time T_{play} of a disk block, storage nodes 1, 2, and 3 schedule the next block. If the fourth client requests video stream D at this moment, the schedule for block $D.0$ will be delayed until storage node 3 is idle; so the loads are balanced across the storage nodes. Hence, the scheduling penalty is $3T_{play}$. Supposing there are d storage nodes (as is in Chapter 3), the worst case service latency will be modeled as follows:

$$(d-1)T_{play} + T_{read}(1) + T_{comm} + T_{net} \;\leq\; T^{max}_{latency}, \tag{4.1}$$

where $T_{read}(k)$ denotes the time to read k disk blocks, T_{comm} is the time to deliver a block from storage node to network node, and T_{net} is the time to deliver a block to clients.

By rewriting Eq. (4.1), the number of nodes in a server may be bounded to

$$d \leq (T^{max}_{latency} + T_{play} - T_{read}(1) - T_{comm} - T_{net})/T_{play}. \quad (4.2)$$

4.2.2 Retrieval scheduling

The performance of video servers is limited by their relatively low disk bandwidth as mentioned in Chapter 3. This section describes a scheduling technique that fully utilizes the disk bandwidth of storage nodes while satisfying the continuity requirement of video streams.

We first consider a process involved in serving a single client. A disk block must be retrieved for the client every T_{play} seconds. From the standpoint of a storage node, it must retrieve a disk block every $d \times T_{play}$ seconds, where d is the number of storage nodes. Thus, the retrieval of a disk block must be completed within $d \times T_{play}$[2].

We now consider N client requests, r_1, r_2, $\cdots$, r_N. In each storage node, N retrievals for r_1, r_2, $\cdots$, r_N constitutes a round. That is, in a storage node, b^1_j, b^2_j, $\cdots$, b^N_j are retrieved in the jth round, where b^i_j denotes the jth disk block of r_i in the storage node. Each disk block, b^i_j, must be retrieved within its deadline, $d \times T^i_{play}$. The deadlines of b^i_j, $1 \leq i \leq N$, will be met if the period of a round T_{round} is given by the shortest playback time among N disk blocks, $d \times T^{min}_{play}$, where $T^{min}_{play} = min_{1 \leq i \leq N}(T^i_{play})$. In order to service N clients without violating the

[2]In Chapter 3, we described more in detail.

round:	(1)	(2)	(3)	(4)	(5)	(6)	(7)	$\cdots$
r_1 :	b_1^1	b_2^1	b_3^1	b_4^1	b_5^1	b_6^1	b_7^1	$\cdots$
r_2 :	b_1^2		b_2^2		b_3^2		b_4^2	$\cdots$
r_3 :	b_1^3			b_2^3			b_3^3	$\cdots$
r_4 :	b_1^4	b_2^4		b_3^4	b_4^4		b_5^4	$\cdots$
time:	0	1	2	3	4	5	6	$\cdots$

Figure 4.4: A scenario of simple round scheduling in a storage node

continuity of streams, the following admission control criteria must be satisfied[3].

$$T_{read}(N) \;\leq\; d \times T_{play}^{min} \tag{4.3}$$

When clients access heterogeneous streams, T_{play}^{i}'s of client request r_i, $1 \leq i \leq N$, are different from each other with respect to the playback rate of video streams. If T_{play}^{1} = T_{play}^{2} = $\cdots$ = T_{play}^{N} = T_{play} then every $d \times T_{play}$ seconds, a block for each client request is retrieved and consumed; thus not accumulated. If $T_{play}^{i} > T_{play}^{min}$, however, data accumulation will occur for r_i. To avoid this untoward effect, we opt to schedule the blocks to be retrieved in each round. We call such a procedure *round scheduling*. Figure 4.4 illustrates a simple scenario where $d \times T_{play}^{1} = 1$, $d \times T_{play}^{2} = 2$, $d \times T_{play}^{3} = 3$, $d \times T_{play}^{4} = 1.5$.

As shown in this example, b_1^1, b_1^2, b_1^3, b_1^4 are retrieved in the first round; b_2^1 and b_2^4 in the second round. As only two blocks are retrieved in the second round, there exists disk idle time. Hence, the client requests that have failed the admission control can be serviced during the idle time. For instance, if we assume that four clients can be simultaneously serviced or that four blocks can be retrieved in a round, and if r_5 with $d \times T_{play}^{5} = 1.5$ and r_6 with $d \times T_{play}^{6} = 2$ arrive, then the

[3]Eq. (4.3) is a generalized form of Eq. (3.1) including heterogeneous streams.

round:	(1)	(2)	(3)	(4)	(5)	(6)	(7)	$\cdots$
r_1 :	b_1^1	b_2^1	b_3^1	b_4^1	b_5^1	b_6^1	b_7^1	$\cdots$
r_2 :	b_1^2		b_2^2		b_3^2		b_4^2	$\cdots$
r_3 :	b_1^3			b_2^3			b_3^3	$\cdots$
r_4 :	b_1^4	b_2^4		b_3^4	b_4^4		b_5^4	$\cdots$
r_5 :		b_1^5	b_2^5		b_3^5	b_4^5		$\cdots$
r_6 :		b_1^6		b_2^6		b_3^6		$\cdots$
time:	0	1	2	3	4	5	6	$\cdots$

Figure 4.5: A scenario of efficient round scheduling in a storage node

admission control for these two requests will fail. Requests r_5 and r_6, however, can be serviced during the disk idle time. Figure 4.5 shows a schedule for this scenario. Further analysis of these cases results in the following theorem. Let $Q = \{i | i = 1, 2, \cdots;$ index for client requests$\}$ and $k_i = T_{play}^{min} / T_{play}^{i}$.

Theorem 4.1 *It is possible to service a new request r_a even when the admission control fails, if the following relationships hold:*

$$\sum_{i \in Q} k_i \ \leq \ 1 \ \textit{and} \ \ a \in Q. \tag{4.4}$$

Proof: k_i for r_i is the number of blocks retrieved in a round and $k_i \leq 1$. If there exists Q such that $\sum_{i \in Q} k_i \leq 1$ then the blocks for $\{r_i | i \in Q\}$ can be retrieved. That is, for r_i, $i \in Q$, k_i/k_{min} blocks are retrieved once every $1/k_{min}$ rounds, where $k_{min} = \min_{i \in Q}(k_i)$. When the values of k_i/k_{min} is not an integer value, the values are toggled between $\lceil k_i/k_{min} \rceil$ and $\lfloor k_i/k_{min} \rfloor$, so that k_i/k_{min} blocks are retrieved once every $1/k_{min}$ rounds on the average. Thus r_a can be serviced. $\square$

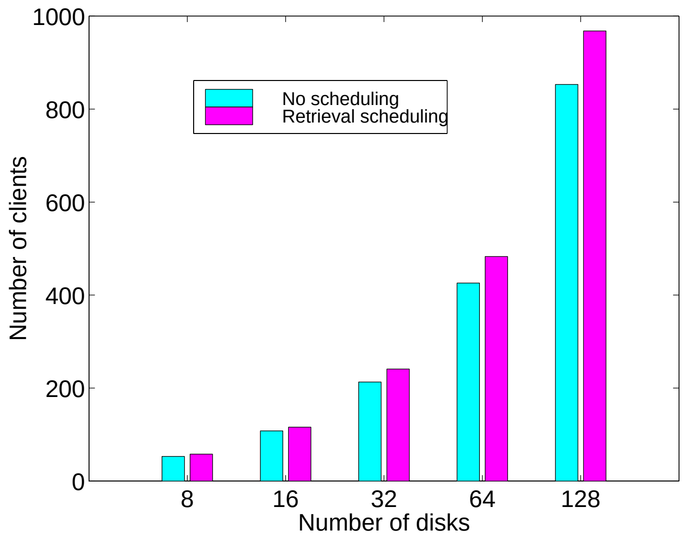


Figure 4.6: Effect of the retrieval scheduling

Let us apply Theorem 4.1 to r_5 and r_6 of Figure 4.5. Since $k_2 = 1/2$, $k_3 = 1/3$, $k_5 = 2/3$, $k_6 = 1/2$, there exist Q_1 and Q_2 for r_5 and r_6, respectively, such that $Q_1 = \{3, 5\}$, $Q_2 = \{2, 6\}$. Hence r_2 and r_6 are serviced every other round. A block for r_3 and two blocks for r_5 are retrieved in three rounds, and these three rounds are repeated.

In consequence, Eq. (4.3) and Eq. (4.4) can be integrated into the final admission control criteria. Figure 4.6 shows the effect of the proposed retrieval scheduling algorithm quantitatively[4]. We can find from Figure 4.6 that integrating Eq. (4.4) to the admission control criteria increases significantly the number of clients that can be serviced simultaneously (about $1 \sim 10\%$).

[4]We assume the following parameters: $T_{seek_max} = 25$msec, $B = 60$KB, $R = 2$MB/sec, T^i_{play} =uniform (200,440)msec.

4.2.3 Communication scheduling

In the previous section, we described the guaranteed retrieval of disk blocks in storage nodes. These blocks must also be transmitted to network nodes in a deterministic fashion over an interconnection network. For further discussion we choose Omega network [Lawr75] as a candidate network in the parallel server. This multistage interconnection network has a property that each data block to be sent through the network involves a unique path between source and destination. Thus, for a given set of blocks it may not be transmitted simultaneously because some of the blocks may conflict with one another. To resolve such conflicts we may need to partition a set S of conflicting data blocks into k subsets, $S_1, \cdots, S_k$, such that each subset is conflict-free [Bern93].

First, we consider communication patterns in the server. The parallel server distributes video streams across all storage nodes and proceeds in periodic rounds at storage and network nodes. During a round, each storage node must transmit to network nodes m data blocks prefetched in the previous round. While delivering video data to network nodes, d storage nodes generate $m \times d$ (src, $dest$) pairs, where src and $dest$ denote a source and a destination, respectively. In our environment, source means storage node and destination does network node. Upon clients' arrival at network nodes, requests are evenly distributed and then sent to storage nodes for service; so the amount of data blocks that a network node receives in return from storage nodes is the same as that of the other network nodes. Therefore, in $m \times d$ (src, $dest$) pairs, each value of src and $dest$ occurs m times exactly whereas the value of src and $dest$ ranges from 0 to $d - 1$.

We now describe the communication scheduling algorithm. A round is divided into $l \times d$ slots, and a data block is transmitted within a slot. For source node i, m data blocks are scheduled as follows:

$$(i, i), (i, i+1) \cdots, (i, d-1), (i, 0), \cdots, (i, i-1), \cdots$$

It can be shown that all data blocks are transmitted during a round without conflict based on the following theorem.

Theorem 4.2 *For a set S of $d \times d$ (src, $dest$), $0 \leq src < d$, $0 \leq dest < d$, pairs of communication paths, S can be partitioned into the following S_k's which are conflict-free*[5]*:*

$$S_k = \{(i, [i+k]_d) \mid 0 \leq i < d\},\ 0 \leq k < d$$

Proof: S can be partitioned into S_k's and it can be obtained from [Lawr75] that S_k, $0 \leq k < d$, is conflict-free. Therefore, the theorem holds. □

To our knowledge, there exist only one study on communication scheduling between storage nodes and network nodes for video servers. Reddy [Redd95] addresses the issue of scheduling communication over the multiprocessor switch for the playback of video streams. He argues that the proposed solution including disk placement makes video scheduling very simple: If the first block of the video is schedulable without network contention, the solution guarantees that there will be no network contention during the entire duration of video playback. Compared with the solution, our communication scheduling is much simpler: We need not

[5]*In the following equation, We define $y = [x]_d$ if $x = a \cdot d + y$, $0 \leq y < d$, for all integer values.*

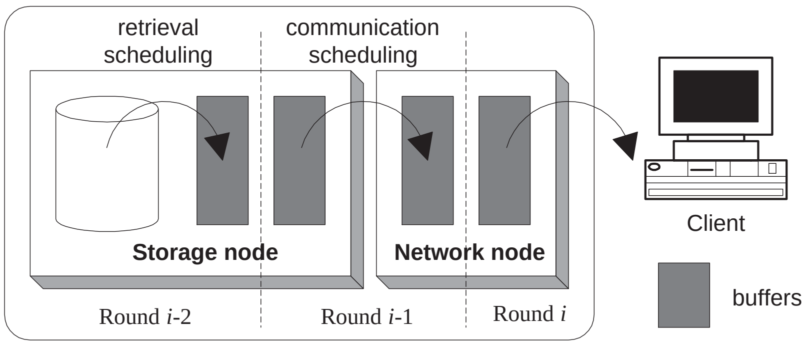


Figure 4.7: Data flow in a parallel server

check if even the first block of the video is conflict-free because the slot-based schedule always guarantees freedom from conflict. Furthermore, the proposed communication scheduling is designed for heterogeneous streams of which the playback rates are different from each other, while the solution in [Redd95] is not.

In general, considering that the link bandwidth of interconnection networks is larger than that of disks, a storage node can transmit more than m blocks to network nodes, or $ld \geq m$. When $ld \geq m$, the admission control criteria we described in Section 4.2.2 can be applied. If $ld < m$, communication network becomes a bottleneck in the parallel server, and a new client who demands that a storage node retrieve more than ld blocks should be rejected. When m equals ld, the utilization of all links in the network reaches 100%.

Since one of the major objectives of designing video servers suggests that they service as many clients as possible, sufficiently large buffers have been assumed for the scheduling algorithms. We now observe the buffer requirement of parallel server. A data flow in parallel server is depicted in Figure 4.7. Since the schedules generated by disk scheduling and communication scheduling are different from each other, we employ the double buffer scheme at both storage node and network

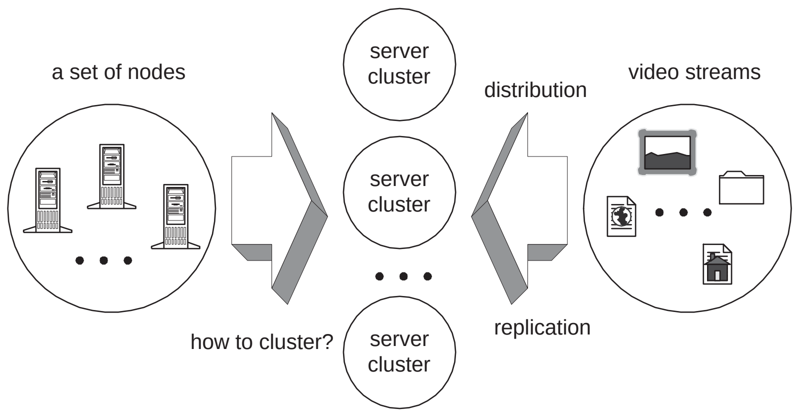


Figure 4.8: Problem description on the configuration of a large-scale server

node. For this reason 4 buffers per client are required in the parallel server for deterministic service guarantees. The effective management of shared buffers, however, will decrease buffer requirements.

4.3 Configuration of a large-scale server

We now examine the configuration of a large-scale video server for given nodes. Figure 4.8 depicts the given problem. When a large number of storage nodes are given, we intend to find the optimal configuration of the large-scale video server through a simulation study. The simulation model is based on the parameters listed in Table 4.1 that are considered suitable for the proposed large-scale server: For 640 storage nodes given, data striping across all the 640 storage nodes causes several problems, as described in Section 4.1. Especially, if we assume the block size to be 256KB[6], service latency will be larger than 300 seconds from Eq. (4.1).

[6]In Chapter 3, we encouraged one or two tracks ($30 \sim 70$KB) for the disk block size. It is based on the single server architecture, however, in which a disk block is striped across single disks. In the parallel server architecture, since each storage node is equipped with high-performance storage

Table 4.1: Parameters used in the simulation

Number of storage nodes	640
Number of stored video streams	2,000
Stream rate of a video stream	0.5 MBps
Length of a video stream	$80 \sim 120$ min.
Block size	256 KB
Disk bandwidth of storage node	20 MB/s
Link bandwidth of network	20 MB/s

Hence, we need to group storage nodes into server clusters.

While Table 4.1 represents the parameters used in the analysis, performance analysis is based on a set of various alternatives for server configuration as listed in Table 4.2. By capacity of a server cluster in Table 4.2 we mean the number of clients that can be serviced simultaneously in a server cluster and by service latency the value in the worst case. In average case, the value is much smaller. All the alternatives in Table 4.2 can service 25,600 concurrent clients when the loads are perfectly balanced across server clusters.

On the basis of video store rental patterns, it is known that access to video streams in the server will be highly localized, with a small number of videos receiving most of the accesses [Cher95]. According to Zipf's Law [Cher95] the probability of choosing the nth most popular one from M videos is C/n, where $C = 1/(1 + 1/2 + 1/3 + \cdots + 1/M)$. Thus, replicating popular video streams in server clusters can keep the load of server clusters balanced. In this experiment we allocate 1,000 video streams to server clusters in the round-robin manner

subsystem, a larger block leads to its high performance.

Table 4.2: The alternatives in the configuration

Number of server clusters	5	10	20	40	80
Number of storage nodes	128	64	32	16	8
Capacity of a server cluster	5,120	2,560	1,280	640	320
Service latency (sec)	64	32	16	8	4

according to their ranking and replicate top ranking videos in all the server clusters. For example, when there are 10 server clusters, each server cluster has 100 unreplicated videos and the top 100 replicated video streams.

First, we carry out the simulation under the worst case assumption that 25,600 clients request concurrently. Video requests are localized according to the Zipf's distribution. Simulation results are given in Figure 4.9 and Figure 4.10. The analysis of the graphs results in the following assertions: (1) Replicating popular videos performs better. It is possible to service 50 to 250 more clients. (2) Until the number of server clusters becomes 20, the average utilization[7] of server clusters is close to 100%. (3) When there are large numbers of server clusters, there exist hot spots among server clusters, that is, server clusters which client requests center around.

Second, we simulate the actual video service with replication. It is assumed that clients arrive at the server according to a Poisson distribution with mean interarrival time, $1/\lambda$, and that the running time of each video is uniformly distributed between 80 and 120 minutes. Figure 4.11 shows similar results to Figure 4.9 and Figure 4.10. When the load becomes smaller in Figure 4.11, the average utilization of server clusters decreases. This is because there exist hot spots among

[7]# of clients being serviced/ # of clients can be serviced

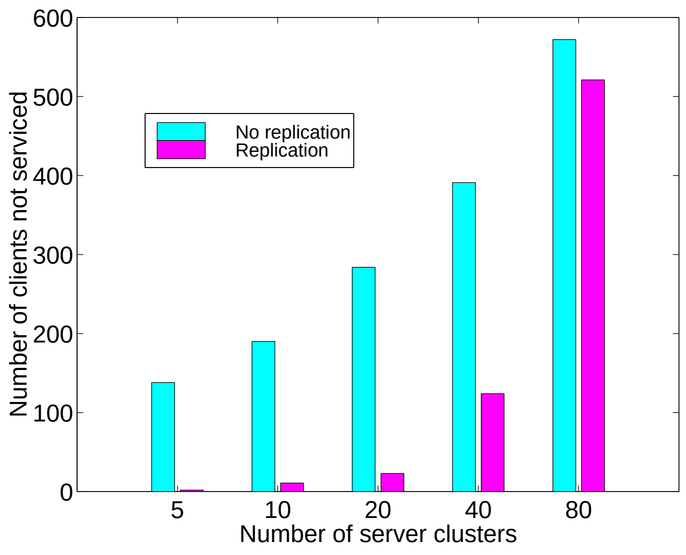


Figure 4.9: Number of clients not serviced (worst case)

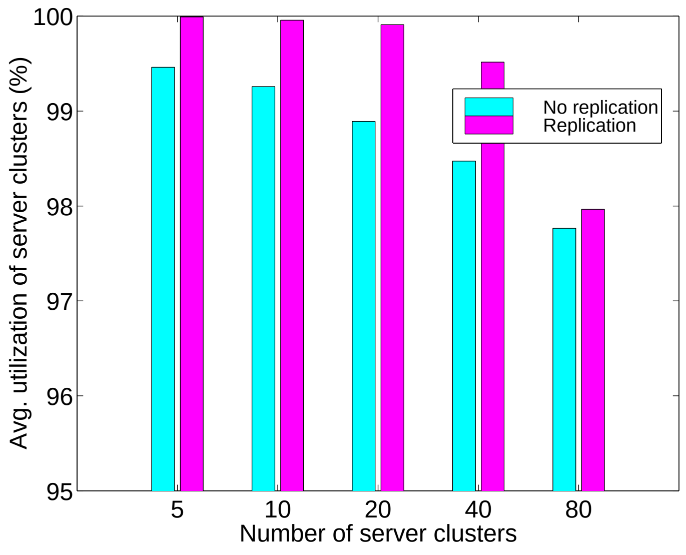


Figure 4.10: Average utilization of server clusters (worst case)

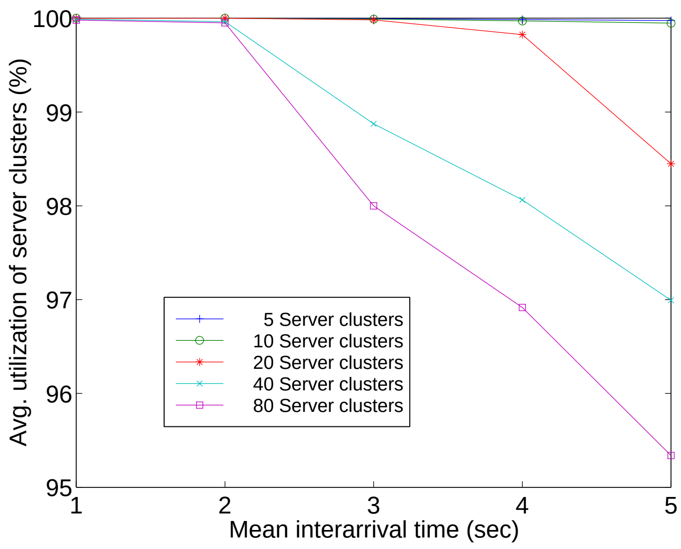


Figure 4.11: Average utilization of server clusters (average case)

server clusters whereas the other server clusters (non-hot spots) are under-utilized. In Figure 4.10, the larger the number of server clusters is, the longer the average waiting time[8] becomes. It also results from hot spots where the waiting time is longer. In addition, since the running time of videos is relatively long, the average waiting time under heavy load ($1/\lambda = 1, 2$) is too long as shown in Figure 4.10.

In summary, given a large number of nodes, clients experience relatively large service latency when the number of server clusters is small, that is, the size of a server cluster is large. On the other hand, when the number of server clusters is large, client requests are not balanced among server clusters, that is, there exist hot spots, even though popular videos are replicated. Consequently, the tradeoff of large versus small clusters provides a basis for the design of the most effective server configuration. As to the size of a server cluster, it can be determined based on the analysis of tradeoff between the utilization and service latency. In the

[8]The waiting time includes only queueing delay here.

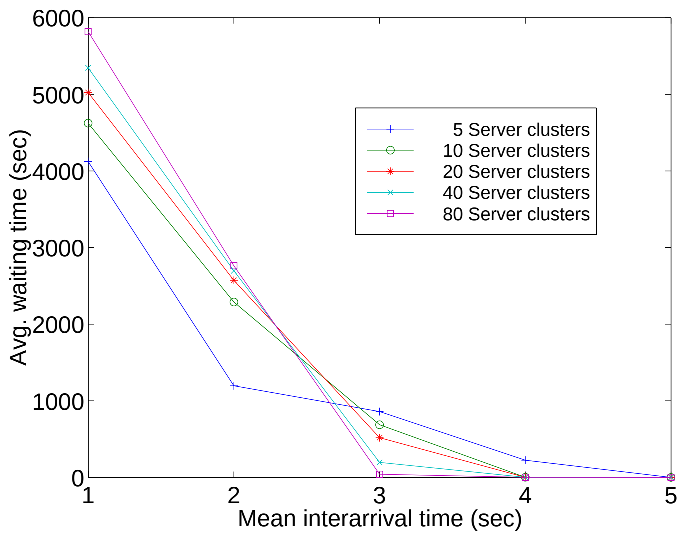


Figure 4.12: Average waiting time (average case)

example given above, the simulation reveals that the most appropriate size of a server cluster is 32 storage nodes, since the average utilization of server clusters is close to 100% while the service latency is relatively small. This value is feasible by current technologies.

4.4 Queueing analysis of a large-scale server

Section 4.2 described the architecture, data placement, retrieval and communication scheduling in each server cluster (parallel server) and we suggested how to cluster a large number of storage nodes into parallel servers in Section 4.3. Then a big picture of large-scale video server can be re-depicted as shown in Figure 4.1. Each server cluster in Figure 4.1 provides individual services for each client. This section propose a queueing model of the large-scale video server with a parallel server being an independent service entity and analyze its performance.

4.4.1 Queueing model

As mentioned above, Figure 4.1 represents the system which we intend to model. A large-scale video server in Figure 4.1 can be modeled as an open queueing network model in Figure 4.13. The queueing network consists of $M + 2$ queues (M server clusters and access network input/output) and N input processes. The proposed model has two assumptions for easy calculation. First, the packet request pattern of client C_i is a Poisson process with parameter λ_i ($1 \leq i \leq N$). Variable bit rate streams such as MPEG are modeled as Poisson processes in the literature [Tewa96a]. Second, the service time in each queue has exponential distribution. Then all the queues in the network are M/M/1 queues as follows: N Poisson processes are superposed on network input queue into a new Poisson process with parameter $\lambda = \sum_{i=1}^{N} \lambda_i$. Next, the output process of network input queue is also a Poisson process with parameter λ according to the Burke's theorem [Harr93]. The Poisson process is decomposed and arrives at server cluster S_i with probability p_i ($1 \leq i \leq M$). The output processes of server cluster queues are also superposed on network output queue into a new Poisson process with parameter λ. In summary, each queue in Figure 4.13 is modeled as follows:

- network input: M/M/1, $\lambda = \sum_{i=1}^{N} \lambda_i$, μ_{ni}, $\rho_{ni} = \lambda / \mu_{ni}$
- server cluster S_i: M/M/1, $p_i\lambda$, μ_{S_i}, $\rho_{S_i} = p_i\lambda / \mu_{S_i}$
- network output: M/M/1, λ, μ_{no}, $\rho_{no} = \lambda / \mu_{no}$

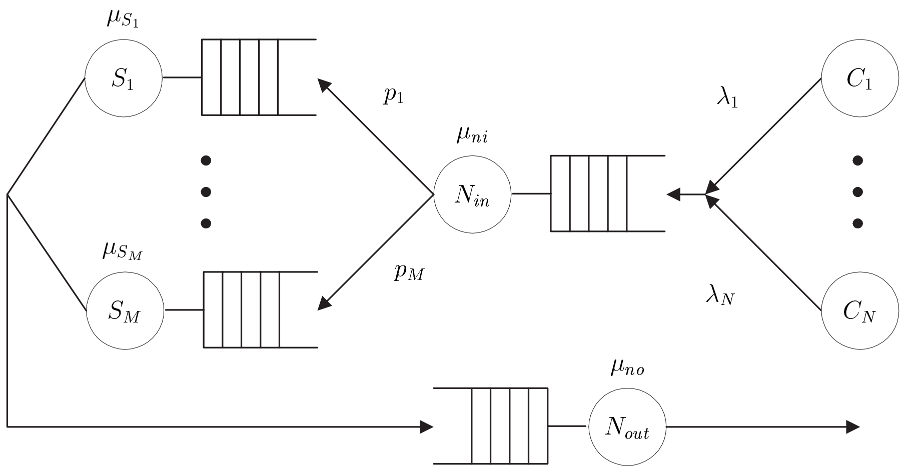


Figure 4.13: Queueing model of a large-scale server

We now intend to derive the distribution of response time in the network of queues for each request of clients and to examine whether each data packet meets its deadline. In other words, we calculate the probability that each packet request is not serviced within its deadline, which is given by

$$P[\text{packet loss}] = \int_{D}^{\infty} f_W(t)dt, \tag{4.5}$$

where $f_W(t)$ denotes the probability density function (pdf) of response time W in the network of queues and D represents the deadline of each packet request, or the playback time of a data packet. Let W_{ni}, W_{S_i}, and W_{no} denote the response time in network input queue, server cluster S_i, and network output queue, respectively. From the analysis of M/M/1 queue [Alle90], we obtain the pdf's of response time of each queue in the network:

$$\begin{aligned} f_{W_{ni}}(t) &= \mu_{ni}(1-\rho_{ni})e^{-\mu_{ni}(1-\rho_{ni})t} \\ f_{W_{S_i}}(t) &= \mu_{S_i}(1-\rho_{S_i})e^{-\mu_{S_i}(1-\rho_{S_i})t} \\ f_{W_{no}}(t) &= \mu_{no}(1-\rho_{no})e^{-\mu_{no}(1-\rho_{no})t} \quad (t \geq 0) \end{aligned} \tag{4.6}$$

The total response time in the network is given by

$$W = W_{ni} + W_{S_i} + W_{no}. \tag{4.7}$$

Then, $f_W(t)$ is calculated by convolving $f_{W_{ni}}(t)$, $f_{W_{S_i}}(t)$, and $f_{W_{no}}(t)$:

$$f_W(t) = f_{W_{ni}}(t) \odot f_{W_{S_i}}(t) \odot f_{W_{no}}(t) \tag{4.8}$$

The complexity of convolution operation is very high. So, we now derive $f_W(t)$ from the Laplace transformation technique [Klei75]. The Laplace transform $F_W^*(s)$ of $f_W(t)$ is obtained from Eq. (4.8) as follows:

$$F_W^*(s) = F_{W_{ni}}^*(s) \cdot F_{W_{S_i}}^*(s) \cdot F_{W_{no}}^*(s), \tag{4.9}$$

where $F_{W_k}^*(s)(k = ni, S_i, no)$ is given from Eq. (4.6) as

$$F_{W_k}^*(s) = \int_0^\infty f_k(t)e^{-st}dt = \mu_k(1-\rho_k)/(\mu_k(1-\rho_k)+s) \tag{4.10}$$

Finally, we get $f_W(t)$ by the inverse-Laplace transformation[9] of Eq. (4.9):

$$f_W(t) = B_1 e^{-a_1 t} + B_2 e^{-a_2 t} + B_3 e^{-a_3 t} \tag{4.11}$$

$$\begin{array}{lll} a_1 = \mu_{ni}(1-\rho_{ni}) & a_2 = \mu_{S_i}(1-\rho_{S_i}) & a_3 = \mu_{no}(1-\rho_{no}) \\ B_1 = \frac{a_1 a_2 a_3}{(a_2-a_1)(a_3-a_1)} & B_2 = \frac{a_1 a_2 a_3}{(a_1-a_2)(a_3-a_2)} & B_3 = \frac{a_1 a_2 a_3}{(a_1-a_3)(a_2-a_3)} \end{array}$$

The correctness of Eq. (4.11) can be validated from Eq. (4.12).

$$\begin{aligned} E(W) &= \frac{B_1}{a_1^2} + \frac{B_2}{a_2^2} + \frac{B_3}{a_3^2} \\ &= \frac{1}{a_1} + \frac{1}{a_2} + \frac{1}{a_3} \\ &= E(W_{ni}) + E(W_{S_i}) + E(W_{no}) \end{aligned} \tag{4.12}$$

[9]We employ the inspection technique [Klei75] here.

Table 4.3: Parameters used in the analysis

Symbol	Description	Values
N	Number of clients	
M	Number of server clusters	10
R_s	Stream rate	1.5Mbps
R_n	Access network bandwidth	$1 \sim 10$Gbps
R_d	Disk bandwidth in a server cluster	$10 \sim 100$MB/s
B_{req}	Request packet size	1KB
B_{data}	Data packet size	256KB
p_i	Probability that a client is serviced at S_i	$1/M$
k	read-ahead	2

Consequently, from Eq. (4.5), the probability that each packet request is not serviced within its deadline is derived as

$$\begin{aligned} P[\text{packet loss}] &= \int_D^\infty f_W(t)dt \\ &= \frac{B_1}{a_1}e^{-a_1 D} + \frac{B_2}{a_2}e^{-a_2 D} + \frac{B_3}{a_3}e^{-a_3 D} \end{aligned} \tag{4.13}$$

4.4.2 Performance analysis

This subsection analyzes the performance of large-scale video servers based on the queueing model described in Subsection 4.4.1. Table 4.3 represents the parameters which affect the performance of video servers. Parameters in Table 4.3 are applied to the queueing model as follows:

$$\begin{aligned} \mu_{ni} &= \frac{R_n}{B_{req}},\ \mu_{no} = \frac{R_n}{B_{data}},\ \mu_{S_i} = \frac{R_d}{B_{data}} \\ \lambda &= \sum_{i=1}^{N} \lambda_i = N\lambda_s = N \cdot \frac{R_s}{B_{data}} \end{aligned} \tag{4.14}$$

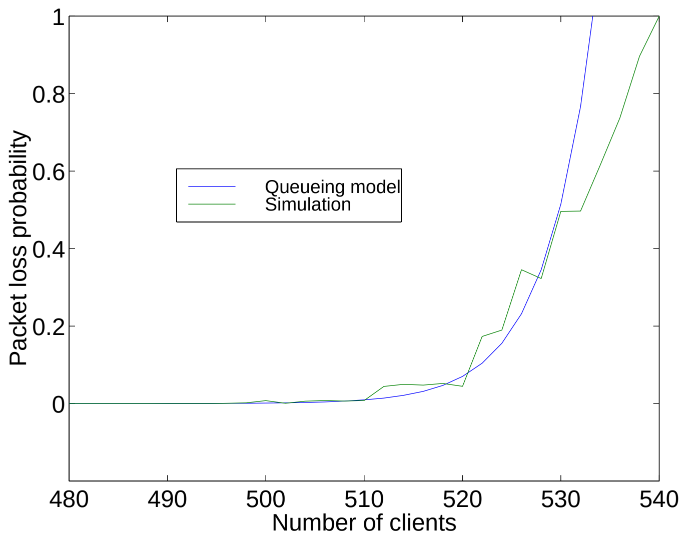


Figure 4.14: Validation of queueing model

$$D = k \cdot \frac{1}{\lambda_s} = k \cdot \frac{B_{data}}{R_s}$$

In Eq. (4.14), we assume that all the clients request the same stream rate of video streams and that all the server clusters are equivalent to each other. It is also assumed that the load balancing between server clusters is achieved; $p_i = 1/M$ for $1 \leq i \leq M$. This must be treated carefully on designing the server.

First, we validate the proposed queueing model through simulation. Given the same value for each parameter, Figure 4.14 shows that the queueing model is quite correct. Especially, it is meaningless when the packet loss probability is greater than 0.2. We describe the results of queueing analysis in the rest of this subsection.

The parameters which greatly affect the performance of large-scale video servers are the disk bandwidth (R_d) and the access network bandwidth (R_n). Since the access network bandwidth is closely related with the number of server clusters,

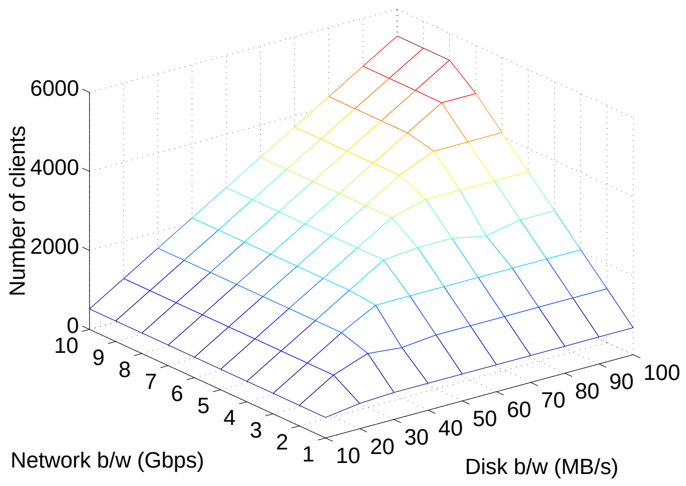


Figure 4.15: Effect of disk and network bandwidth

it is considered carefully on the design of large-scale video servers. That is, the following condition should be met:

$$R_n \geq R_d \times M \tag{4.15}$$

The effect of disk and network bandwidth is given in Figure 4.15. Z-axis represents the number of clients that can be serviced simultaneously while the packet loss probability is less than 0.05. As shown in Figure 4.15, the more clients can be serviced as the disk and network bandwidth increase. However, either of them may be the performance bottleneck; so, the proper values for them should be derived, which can be known from Figure 4.15.

Figure 4.16 depicts the effect of read-ahead parameter k. The deadline D can be increased by reading k data blocks ahead; the packet loss probability decreases. As expected, the larger k performs better, but the performance gain decreases as k increases. Since the buffer requirement also increases as k increases, we believe that 2 or 3 read-ahead is the most appropriate. On the other hand, Figure 4.17

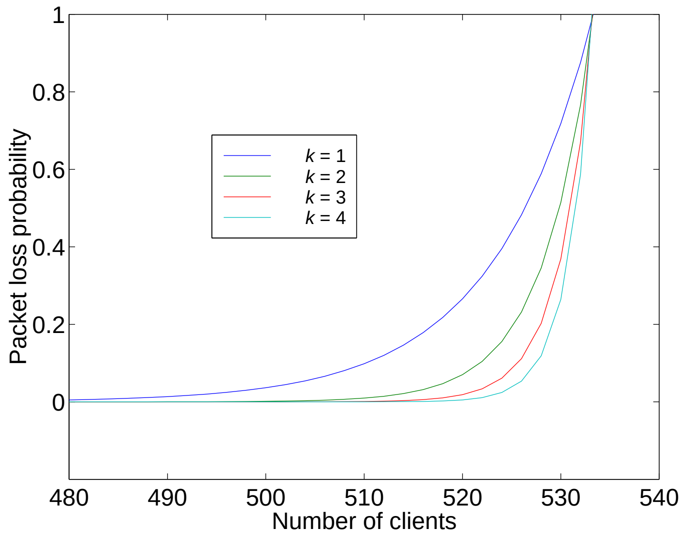


Figure 4.16: Effect of read-ahead

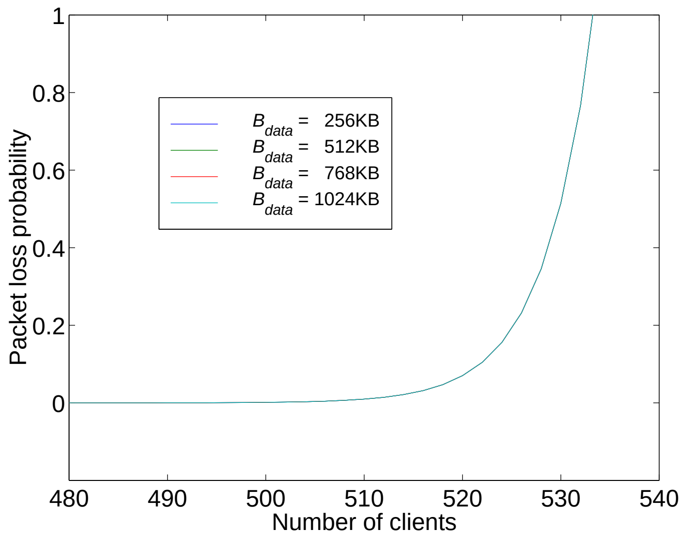


Figure 4.17: Effect of data block size

demonstrates that data block size does not affect the performance of video servers in the analysis. This is because the disk bandwidth is fixed. As described in Chapter 3, data block size is closely related with the disk bandwidth. We can conclude from Figure 4.16 and Figure 4.17 that, if the disk bandwidth is not dependent upon data block size, the large read-ahead is desirable rather than the large block size.

Chapter 5

Storage and retrieval in a multi-resolution video server

Video can be encoded into multiple-resolution format in nature. Recent advances in video coding technology make it possible to create a multi-resolution or scalable video stream. In general, a multi-resolution video stream permits the extraction of lower resolution subsets of the full resolution stream that may be decoded independently. Employing the multi-resolution video in video servers provides the following benefits: *heterogeneous client support, storage efficiency, adaptive service, and interactive operations support*.

First, clients in a video service are likely to request various QoS parameters, such as color depth, window size, and frame rate, because they have different decoding capabilities and network bandwidth connected to the server. Second, servicing single (full) resolution video streams for a wide range of clients results in wasting server resources such as disk and network bandwidths. Since multiple versions with different resolutions for each video stream lead to storage inefficiency, it is required to employ scalable video. Third, on the transient overloaded condition, the server is able to provide adaptive services by gracefully degrading the resolution levels. Furthermore, even when the admission of new clients fails, the storage of scalable video permits the server to gracefully degrade the resolution levels of existing clients in order to service new clients. In addition, the server can provide adaptability to the fluctuation in network bandwidth, which is one of fundamental problems in mobile computing environment [Cho97c, Cho99a]. Finally, the lower resolution streams enable the server to efficiently support interactive operations such as fastforward and rewind.

In this chapter, we present a design framework for video servers which provide multiple resolution services: multi-resolution video model, server model, data placement and retrieval of multi-resolution video, interactive operations support, and admission control. We also describe implementation experiences of multi-resolution video server.

5.1 System model

5.1.1 Multi-resolution video stream model

In general the notion of video resolution is defined in three dimensions: chroma, spatial, and temporal. In these dimensions, video streams can be compressed into

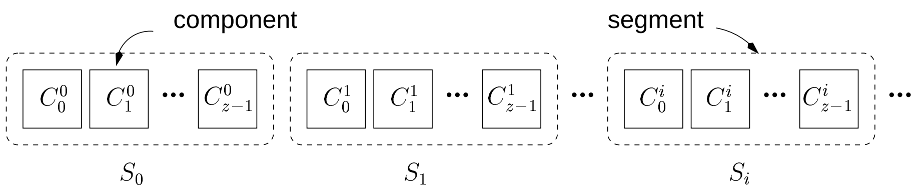


Figure 5.1: z-level multi-resolution video stream model

multiple-resolution format by various scalable compression algorithms. A multi-resolution or scalable video stream is a video sequence encoded such that subsets of the full resolution video bit stream can be decoded to recreate lower resolution video streams.

For the purpose of modeling multi-resolution video streams, we propose a z-level multi-resolution video stream model in Figure 5.1. A multi-resolution video stream is a set of segments in which a segment consists of z components. In other words, for a video stream V,

$$\begin{aligned} V &= \{S_s \mid S_s \text{ is a segment},\ 0 \le s < l\} \\ S_s &= \{C_c^s \mid C_c^s \text{ is a component},\ 0 \le c < z\}, \end{aligned}$$

where s and c denote the segment number and the component number, respectively, and l is the number of segments, or the length of V. The k-level resolution can be obtained by integrating k components from the lowest one; so, $\{C_c^s \mid 0 \le s < l,\ 0 \le c < k\}$ are serviced. In the multi-resolution video model, each video stream can be provided with z levels of quality and the QoS parameter is represented by the number of components in a segment, or k. Full resolution quality dictates the use of all the components.

The multi-resolution video stream described above can be implemented by various coding technologies. The current scalable video compression techniques include DCT-based schemes, subband (wavelet) schemes, fractal-based schemes, and object-based schemes [Hunt]. First, of the standard codecs, only MPEG-2 [ISOb] addresses scalable video streams. Four techniques, namely data partitioning, SNR scalability, spatial scalability, and temporal scalability, can be used. Since a frame consists of multiple layers, a frame can be mapped to a segment in the proposed video stream model and the sub-layers of the frame constitute components of the segment. Alternatively, multiple frames may be mapped to a segment because a large storage/retrieval unit is beneficial to the disk performance [Chan97]. For example, Paek *et al.* [Paek95] implement a three layer MPEG-2 video stream. In their scheme, the base layer provides the initial resolution video while an additional spatial enhancement layer allows for the upsampling and hence increases in frame size of the base layer. A further SNR enhancement layer provides increase in the visual quality of the base+spatial enhancement layers of video. One possible mapping from their video stream to our model is that a group-of-picture (GOP) corresponds to a segment, that is,

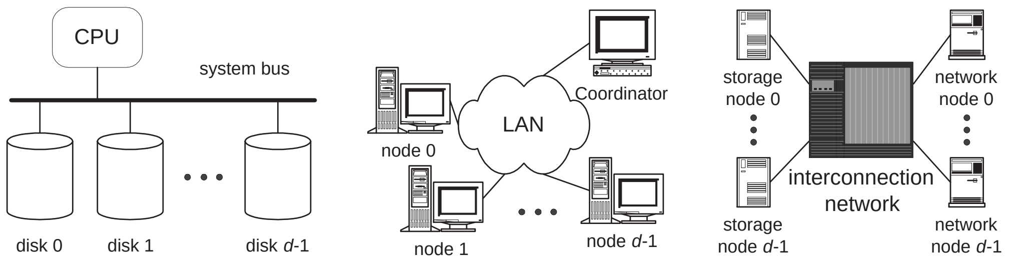
$$\begin{aligned} V &= \{S_s \mid S_s = \text{GOP}^s,\ 0 \le s < l\} \\ S_s &= \{C_c^s \mid C_c^s = \text{GOP}_c^s,\ 0 \le c < 3\}, \end{aligned}$$

where GOP^s denotes the s-th GOP. The components GOP_0^s, GOP_1^s, and GOP_2^s are the base layer, the spatial enhancement layer, and the SNR enhancement layer for GOP^s, respectively.

MPEG-1, which is another DCT-based coding scheme, can also exploit scalability techniques such as data partitioning with slight modification in existing codecs [Shen95]. In addition, without modifying codecs, we can reconstruct MPEG-1 video into the multi-resolution video model in temporal dimension as follows: (1) A GOP is mapped to a segment. (2) An I frame is the first component in a segment. (3) P frames constitute the next one or more components. (4) B frames constitute the rest of the components in the segment. This is similar to a rearrangement scheme of Chang and Zakhor [Chan94] which stores the frames within a GOP in a specific order.

Taubman and Zakhor [Taub94] propose and implement a scalable codec capable of generating bit rates from tens of kilo bits to several mega bits per second with fine granularity of available bit rates. The codec is based on 3-D subband coding and multi-rate quantization of subband coefficients, followed by arithmetic coding. Chang and Zakhor [Chan97] use 11-layer scalable video streams produced by the codec which range from 190 Kbps and 1330 Kbps in their work for storage and retrieval of scalable video. We can reconstruct the video streams into the proposed video model in the same way as the scalable MPEG-2 which is described above.

Bogdan [Bogd94] proposes a multi-scale fractal video coding. The scheme combines the still image pyramid coding and the ITT (iterated transformation theory) inter-frame video coding methods to generate a hierarchy of bit-streams. MPEG-4 is scalable in the sense that multiple objects can be added or removed to compose a frame. The fractal-based and object-based coding schemes are also consistent with the proposed model. Consequently, we can conclude that we can utilize 'off-the-shelf' technology in order to implement the multi-resolution video stream model.



(a) single server (b) distributed server (c) parallel server

Figure 5.2: Architecture of multi-resolution video server

5.1.2 Server model

As described in Chapter 3 and Chapter 4, video servers range from a standard PC for small-scale systems to massively parallel or distributed computers for large-scale systems (see Figure 5.2). The architecture in Figure 5.2 can be modeled into a disk array model where the server has d disks and video data are striped across the disks. In distributed or parallel servers, a disk corresponds to a disk subsystem of each node. In this chapter, we assume the disk array model for the multi-resolution video server architecture and consider the large-scale case (distributed or parallel server) for system parameters. Many works are also founded on the model but most of them conduct the worst case analysis for the performance of a disk (i.e. the maximum seek time and rotational latency). This may underestimate the disk performance. Furthermore, the analysis is not directly applicable to the RAID disk subsystem in distributed or parallel servers. For flexibility, we consider only the effective bandwidth Φ for the performance of a disk subsystem. The value can be measured from a calibration program that determines the maximum number of blocks that can be read within the given time interval [Maka97]. For convenience's sake, we use term 'disk' instead of 'disk subsystem' in the rest of this chapter.

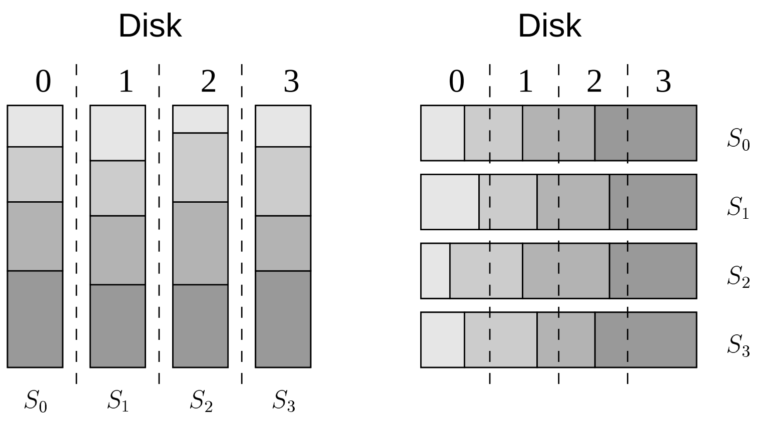
The operation of video servers is re-described briefly. A video server proceeds in periodic rounds due to its periodic nature. In each service round of which the length is T_{round}, a video server retrieves the required amount of data with respect to its playback duration and transmits them to remote clients. A double buffer scheme enables the disk and network bandwidths to be effectively utilized. In other words, in each round, data are retrieved to maximize the disk performance and the transmission of data retrieved in the previous round is performed to ensure the real-time playback capability considering the buffer space of each client. Assuming that the network bandwidth is large enough for the transmission, we are concerned about the effective disk bandwidth management for multi-resolution video data.

5.2 Data placement for multi-resolution video

The performance of video servers is closely related with data placement. A data placement scheme should explore the followings: First, it should provide deterministic access for simple retrieval scheduling. Second, the performance efficiency should be considered such as throughput and service latency. Third, it should support interactive operations with reasonable cost. Next, the disk load balancing should be achieved so that the server may be able to fully utilize the aggregate disk bandwidth.

Before placing data on disks, we first have to determine storage units by which data are written to or read from disk. Constant bit rate (CBR) video streams require the equal amount of data in each round, but variable bit rate (VBR) streams do not. There exist two methods for VBR streams [Chan97]. The constant time length (CTL) method is to store and retrieve video data in unequal amounts with respect to its real-time playback duration. In contrast, the constant data length (CDL) is to store and retrieve data in equal-sized blocks while utilizing buffer memory to provide real-time playback. The former provides advantages in buffer usage and disk throughput but has the fragmentation problem. The latter is consistent with the current disk storage technology, but requires large buffer space and complicated retrieval scheduling. In order to alleviate the problems, we can employ a hybrid method in which data are stored in fixed-size blocks, but the number of blocks to be retrieved varies with the playback duration. The CTL method is more efficient in a read-only environment such as VOD because it reads a large chunk of data contiguously while there exist seek operations in the hybrid method [Chan97]. On the other hand, the hybrid method is a viable approach for the design of integrated multimedia file system where multimedia data are created, edited, and deleted frequently [Vin95].

We model the hybrid approach in consideration of flexibility and allocate a variable number of fixed-size blocks for a component, that is, $size(C_c^s) = b_c^s B$, where B denotes the disk block size. However, if we choose the smallest allocation unit for B (one sector, or 512 bytes) and place blocks in a component contiguously, it results in the CTL scheme. We follow this assumption for the analysis in the rest of this chapter, because we target a video server where read requests are dominant. As for the component size $size(C_c^s)$ in the multi-resolution video stream model, we construct a segment based on the round length T_{round},



(a) concurrency (b) parallelism

Figure 5.3: Striping strategies: concurrency vs. parallelism

so that a segment is serviced in each round. This leads to a large and logically contiguous data chunks, and hence, the high disk throughput can be achieved.

We now intend to place multi-resolution video data on a disk array. There exist two straightforward strategies which explore different aspects of the concurrency and parallelism offered by striping data across disks, as depicted in Figure 5.3. The degree of concurrency is defined as the number of outstanding requests at one time and the parallelism describes the number of disks that service a single request. Chang and Zakhor [Chan94] propose the periodic interleaving scheme using the concurrency of multiple disks and Paek *et al.* [Paek95] define the second strategy (parallelism) as the balanced placement scheme. The periodic interleaving scheme accesses only one disk in a round for a segment and, in the balanced placement, a segment is divided into d equal amounts of data and placed over all d disks.

Two extremes of data placement show a tradeoff of disk throughput versus service latency. The periodic interleaving scheme achieves high disk throughput due to large and logically contiguous data chunks, but the worst case service latency is d rounds [Paek95] because a service should be delayed considering the load balancing of disk bandwidth[1]. The service delay consists of the waiting time plus one round for filling a buffer in the double buffer system. On the contrary, in the balanced placement, Paek *et al.* argue that the service latency is one round all the time although relatively small data chunks cause to lower disk throughput. They also present a hybrid multiple segmentation scheme, on the basis of the tradeoff analysis. In the scheme, they define a segmentation level S which represents the degree of parallelism. Each segmentation group (S disks) are performed in parallel and d/S disks concurrently.

However, all the schemes are based on full-resolution services. For lower resolution services, a small quantity of data are retrieved in the periodic interleaving and a subset of disks participate in the retrieval in the balanced placement scheme. Hence, both schemes cannot guarantee their advantage (i.e. throughput and service delay, respectively) in lower resolution services. In addition, they do not consider the disk access boundaries for each component, so that each component in a segment is not accessible independently. Furthermore, the load balancing issue of disk bandwidth for VBR streams is not described.

To take advantage of both of concurrency and parallelism for each resolution services, we place each segment of a video stream in parallel but each component in a segment concurrently. In other words, since the independent access unit is a component, we place each component contiguously in a disk and components in a segment are striped across disks. The finer storage granularity provides the advantages over the periodic interleaving scheme: better load balancing and less

[1]We also mentioned in Chapter 4.

Table 5.1: Advantages of the proposed data placement scheme

over the periodic interleaving	over the balanced scheme
finer storage granularity	sequential and independent access
better load balancing	to each component
less bandwidth fragmentation	load balancing on lower resolution services

disk bandwidth fragmentation. Disk bandwidth fragmentation refers to a situation where the available bandwidth in each disk is not sufficient to accommodate an incoming request, although there is sufficient aggregate bandwidth across disks in the array [Chen95]. On the other hand, the proposed placement scheme guarantees the sequential and independent access to each component, which the balanced scheme does not provide. The balanced scheme incurs the load imbalance problem on lower resolution services because the lower resolution components are placed on the same disk (see Figure 5.3).

In summary, by taking the hybrid approach of two strategies which explore different aspects of the concurrency and parallelism offered by striping data across disks, the proposed placement scheme has the advantages over the periodic interleaving scheme and the balanced scheme, respectively, as shown in Table 5.1.

We begin by introducing an example of data placement in Figure 5.4. Three cases are identified according to the resolution level of stream z and the number of disks d. First, in case of $z \leq d$, components in a segment are distributed across all the disks and successive components are placed on adjacent disks[2]. Next, for the disk load balancing, the first components of successive segments S_i and S_{i+1} (i.e., C_0^i and C_0^{i+1}) are assigned on adjacent disks(disk j and disk $j+1$), as

[2]The adjacent disk of disk $d-1$ is disk 0.

disk	0	1	2	3
	C_0^0	C_1^0	C_2^0	C_3^0
V_1	C_3^1	C_0^1	C_1^1	C_2^1
	C_2^2	C_3^2	C_0^2	C_1^2
	C_1^3	C_2^3	C_3^3	C_0^3
	$\cdots$	$\cdots$	$\cdots$	$\cdots$
	C_3^0	C_0^0	C_1^0	C_2^0
V_2	C_2^1	C_3^1	C_0^1	C_1^1
	C_1^2	C_2^2	C_3^2	C_0^2
	C_0^3	C_1^3	C_2^3	C_3^3
	$\cdots$	$\cdots$	$\cdots$	$\cdots$

(a) $z = d$

disk	0	1	2	3	4
	C_0^0	C_1^0	C_2^0	C_3^0	
V_1		C_0^1	C_1^1	C_2^1	C_3^1
	C_3^2		C_0^2	C_1^2	C_2^2
	C_2^3	C_3^3		C_0^3	C_1^3
	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
		C_0^0	C_1^0	C_2^0	C_3^0
V_2	C_3^1		C_0^1	C_1^1	C_2^1
	C_2^2	C_3^2		C_0^2	C_1^2
	C_1^3	C_2^3	C_3^3		C_0^3
	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

(b) $z < d$

disk	0	1	2
	C_0^0	C_1^0	C_2^0
V_1	C_3^0	C_0^1	C_1^1
	C_2^1	C_3^1	C_0^2
	C_1^2	C_2^2	C_3^2
	$\cdots$	$\cdots$	$\cdots$
	C_2^0	C_0^0	C_1^0
V_2	C_1^1	C_3^0	C_0^1
	C_0^2	C_2^1	C_3^1
	C_3^2	C_1^2	C_2^2
	$\cdots$	$\cdots$	$\cdots$

(c) $z > d$

Figure 5.4: An example of data placement for multi-resolution video

shown in Figure 5.4(a) and 5.4(b). In addition, we distribute the starting point (the first component in the first segment, or C_0^0) of each stream across disks for load balancing when multiple streams are requested concurrently. For the service of V_1 in Figure 5.4(a) with the second level resolution, for example, the first segment is retrieved from disk 0 and disk 1 (C_0^0 and C_1^0, respectively) and the second from disk 1 and disk 2 (C_0^1 and C_1^1). Next, when $z > d$, multiple components in a segment may be placed on a disk. However, the strategy is similar to the case of $z \leq d$. That is, successive components in a segment are placed on adjacent disks and the first components of successive segments are assigned on consecutive disks as depicted in Figure 5.4(c). The data placement scheme allows deterministic access to disks. For $V = \{C_c^s \mid 0 \leq s < l,\ 0 \leq c < z\}$, the disk which contains a component C_c^s is calculated as follows[3]:

$$D(C_c^s) = [s + c + StartDisk_V]_d, \tag{5.1}$$

where $StartDisk_V$ indicates the disk in which the starting point (C_0^0) of V is

[3]In Eq. (5.1), we define $y = [x]_d$ if $x = a \cdot d + y$, $0 \leq y < d$, for all integer values.

stored.

We now show the disk load balancing property of the placement scheme. Since the number of concurrent clients in a video server with multiple disks depends on the most heavily loaded disks [Vin95], this issue should be examined carefully. Let $\mathcal{V}_{i,k}$ denote a set of components retrieved in disk i during k-level service of V. From Eq. (5.1), we obtain

$$\mathcal{V}_{i,k} = \{C_c^s \mid D(C_c^s) = i, 0 \le s < l, 0 \le c < k\}. \tag{5.2}$$

Theorem 5.1 *Given the parameters above, $|\mathcal{V}_{i,k}|$ is given as follows:*

$$|\mathcal{V}_{i,k}| = \left\lfloor \frac{l}{d} \right\rfloor \times k + \alpha \;\; (0 \le \alpha < d) \tag{5.3}$$

Proof: See Appendix A. □

Theorem 5.1 indicates that, regardless of the resolution level of video service, components are evenly distributed across all the disks. Disk load balancing in CBR video streams can be directly derived from Theorem 1 because CBR streams have equal-sized components of $size(C_c^s) = bB$, $0 \le s < l$, $0 \le c < z$.

When a VBR scalable coding algorithm is employed to obtain the compression efficiency, the size of each component varies, that is, $size(C_c^s) = b_c^s B$ is not constant. This may lead to the load imbalance. In [Shen98], Shenoy and Vin suggest a scheme in which the block size can vary across sub-streams (lower resolution streams) but is fixed for a given sub-stream to maximize performance. This can be realized in the multi-resolution video stream model by fixing the number of blocks in the same component level, or $b_c^s = b_c$, $0 \leq s < l$. The number of blocks for storing components in $\mathcal{V}_{i,k}$, or $n(\mathcal{V}_{i,k})$ is calculated from Eq. (5.3).

$$\begin{aligned} n(\mathcal{V}_{i,k}) &= \sum_s \sum_c \sum_{C_c^s \in \mathcal{V}_{i,k}} b_c^s = \sum_{c=0}^{k-1} \left(\sum_{s, C_c^s \in \mathcal{V}_{i,k}} b_c \right) \\ &\approx \sum_{c=0}^{k-1} \left\lfloor \frac{l}{d} \right\rfloor b_c \end{aligned} \tag{5.4}$$

From Eq. (5.4), we can observe that the disk workload in VBR streams is also evenly distributed for any resolution video stream. Even when b_c^s is variable in the same component level, it is expected that the following equation holds statistically:

$$n(\mathcal{V}_{i,k}) \approx \sum_{c=0}^{k-1} \left\lfloor \frac{l}{d} \right\rfloor E[b_c], \tag{5.5}$$

where $E[b_c]$ is the mean of b_c^s for $0 \leq s < l$, or $E[b_c] = \frac{1}{l} \sum_{s=0}^{l-1} b_c^s$. We will validate Eq. (5.5) through experiments in Section 5.4.

Observe that we achieve the disk load balancing for a given resolution video service. However, we have to consider another load balancing issue for the workload induced by concurrent clients. The issue should be treated in the retrieval scheduling upon startup or interactive operations.

5.3 Data retrieval for multi-resolution video

As mentioned earlier, a video data retrieval proceeds in periodic rounds. In the multi-resolutoin video server, multi-resolution video data are constructed such that a segment is played back for T_{round}. So, for each video stream V_j, a segment is serviced in a round, and hence, k components are retrieved in parallel across

```
Procedure Scheduler$_i$
input: $V_j = (k_j, s_j, K_j, StartDisk_{V_j}, direction), 1 \le j \le N$
output: $\mathcal{D}_i$, a set of components to be retrieved from disk $i$ in a round
begin
1:    clear $\mathcal{D}_i$
2:    for $j := 1$ to $N$
3:        for $m := 1$ to $K_j$
4:            for $c := 0$ to $k_j - 1$
5:                if ($[s_j + c + StartDisk_{V_j}]_d = i$)
6:                    insert $C_c^{s_j}$ into $\mathcal{D}_i$
7:                end if
8:            end for
9:            $s_j = s_j + direction$
10:       end for
11:   end for
end
```

Figure 5.5: Scheduler at disk i

disks for k-level resolution service. Since the data placement scheme proposed in Section 5.2 allows deterministic access for each component, each disk can retrieve data independently.

In Figure 5.5, we present a simple retrieval scheduling procedure performed at disk i in each round. The input parameters of a video stream consist of its resolution level (k_j), the current segment number (s_j), the number of segments retrieved in a round (K_j) which is one in normal playback, $StartDisk_{V_j}$, and playback $direction$ which is set to 1 or -1 according to forward and reverse playbacks, respectively. The scheduler generates a set of components $\mathcal{D}_i$ to be retrieved from its disk in each round. In Line 6 of Figure 5.5, we can incorporate a disk scheduling algorithm to optimize the performance of its disk subsystem.

While each disk performs data retrieval independently, we have to schedule the requests of clients globally (*request scheduling*), as mentioned in Section 5.2, so that the disk workload induced by concurrent clients may be evenly distributed across the disks. The strategy is to delay the start point of service considering the disk load balancing. Since the workload in a disk is shifted to the next disk in the next round, we can calculate the average workload in each disk for the next r rounds ($r \leq d$). One observation is that the maximum number of blocks retrieved in a disk should be minimized, because the most heavily loaded disks determine the number of clients that can be serviced simultaneously. We delay the start point of service until the maximum number of blocks retrieved in each disk is minimized. The worst case service latency is r rounds. The look-ahead parameter r presents a tradeoff between disk load balancing and service latency. We assume $r = d$ in the rest of this chapter because it is worthwhile to increase the number of concurrent clients at the expense of acceptable service latency in video servers. We present an experimental result in Section 5.4 which shows that the number of concurrent clients increases significantly at the expense of acceptable service latency.

5.3.1 Support for interactive operations

Interactive operations are essential for video services. Clients are likely to perform VCR-like operations on video they are watching, such as pause, resume, fastforward, rewind, and slow playback. Fast scan operations, namely fastforward and rewind, should be treated carefully because they require additional server resources. In general, two approaches support them: encode separate streams and skip frames. The first approach needs extra storage space and the second approach may lead to load balancing problems [Lee98].

The multi-resolution video server supports fast scan operations without any additional overhead by degrading the resolution level and retaining the data rate of the video stream. For example, let us assume that a fastforward operation is requested for a video stream with the fourth level resolution. If we lower the resolution level to the second level and the first level, the two-times-fastforward and the four-times-fastforward can be accomplished, respectively, without any additional disk and network bandwidth. Shenoy and Vin [Shen98] validate this idea by a scalable encoding technique in which the low-resolution-based sub-stream provides acceptable video quality for scan operations. For the case of low resolution level where it is impossible to degrade the level, a segment skipping scheme [Chen94] can be integrated with our scheme.

The scheduler in Figure 5.5 can be updated in order to support interactive operations as follows. We assume that an interactive operation is requested to $V_j = (k_j, s_j, K_j, StartDisk_{V_j}, direction)$. All interactive operations are supported simply by reconstructing the input parameters of V_j.

Fastforward The new input parameters are given by $V_j^{'} = (k_j^{'}, s_j, K_j^{'}, StartDisk_{V_j}, direction^{'})$, where $k_j^{'} = k_j/m$, $K_j^{'} = mK_j$, and $direction^{'} = \alpha$. In this case, we can achieve $m \cdot \alpha$-times fastforward[4]. When $\alpha > 1$, the segment skipping scheme is employed. For instance, when four-times fastforward

[4]In more detail, $\frac{R_{V_j,k_j}}{R_{V_j,k_j^{'}}} \cdot \alpha$-times fastforward is achieved, where $R_{V,k}$ denotes the average playback rate of V with k resolution service.

is requested to $V_j = (4, 1000, 1, 0, 1)$ that is being played back with forth-level resolution, we can obtain $V_j^{'} = (1, 1000, 4, 0, 1)$. In addition, when $V_j = (1, 2000, 1, 0, 1)$ (*i.e.* normal playback with first-level resolution), the segment skipping scheme is incorporated by $V_j^{'} = (1, 2000, 1, 0, 5)$ for five-times fastforward. As mentioned above, the segment skipping scheme leads to disk load imbalance when α and d have the least common multiple (LCM). For example, consider a video server having four disks in Figure 5.4(a). For two-times fastforward of $V_1 = (1, 0, 1, 0, 1)$, $V_1^{'}$ is given to $(1, 0, 1, 0, 2)$. Then the server retrieves a sequence of C_0^0, C_0^2, C_0^4, C_0^6, $\cdots$, so that disk 0 and disk 2 will handle all the retrievals. This problem can be solved by selecting α such that α is relatively prime to d [Kwon97].

Rewind This is equivalent to fastforward except $direction^{'} = -\alpha$.

Slow playback Reducing the number of segments K_j accomplishes the slow playback, or $K_j^{'} = K_j/m$ for m-times slow motion. When $K_j^{'} < 1$, V_j is excluded from the input list of the scheduler until $L_j \geq 1$, where L_j (initially $K_j^{'}$) is increased by $K_j^{'}$ in each round and decreased by one when $L_j \geq 1$.

Pause and resume The scheduler excludes V_j on pause and includes V_j again on resume.

5.3.2 Admission control

A video server must employ an admission control algorithm to decide whether a new client can be serviced without violating the real-time requirements of clients already being serviced. Since a CBR video stream V_j produces a constant disk workload (n_j blocks) in each round, we can employ a simple admission control algorithm which checks if all the blocks ($\sum_{j=1}^{N} n_j$) for N streams can be retrieved in a round. For VBR streams, the simple algorithm may use $n_j = max_{0 \leq i < s}(n_j^i)$ or $avg_{0 \leq i < s}(n_j^i)$, where n_j^i is the number of blocks to be retrieved in the i-th round. However, this causes to under-utilize or over-utilize the server resources, respectively.

Admission control algorithms for VBR streams may be classified into two categories: statistical and deterministic. The first approach exploits the bit rate statistics of video streams and the second approach does the specific knowledge of the bit traces of video streams. Vin *et al.* [Vin94] propose a statistical admission control algorithm with a mechanism enforcing statistical service guarantees. They compute the overflow probability, which is the probability that the service time for a single disk access exceeds the round duration, by determining the total number of blocks in a round statistically and empirically measuring a distribution function for the service time. Chang and Zakhor [Chan97] calculate the probability of overload by integrating the probability density function of the aggregated resource required by all clients beyond a given threshold limit. The threshold limit is computed from a single disk performance analysis on their data placement schemes. Makaroff *et al.* [Neuf96, Maka97] propose a deterministic admission control algorithm based on the stream block schedule which contains the number of blocks to be retrieved in each round. The admission of a new stream is accomplished by merging the stream block schedule with the existing one and checking a system overflow during the length of the request. The deterministic admission control algorithm provides a tight and safe bound for the admission, but its complexity is relatively high.

We now describe an admission control algorithm in the multi-resolution video server. Assume that client j for $1 \leq j \leq N-1$ is being serviced with k_j-level resolution of V_j and a new client N requests V_N with k_N-level resolution service. Since each disk must retrieve all the components scheduled in Figure 5.5 every round for the deterministic service guarantee, the following inequality must be satisfied in each disk:

$$n(\mathcal{D}_i) \times B \leq T_{round}\Phi, \quad 0 \leq i < d, \tag{5.6}$$

where $n(\mathcal{D}_i)$ denotes the number of blocks required to store $\mathcal{D}_i$. In Eq. (5.6), $n(\mathcal{D}_i)$ can be calculated deterministically in CBR multi-resolution video streams, but $n(\mathcal{D}_i)$ varies from round to round in VBR case. For VBR streams, we attempt to estimate an upper bound n_{upper} of $n(\mathcal{D}_i)$ statistically such that

$$P_{overflow} = P[n(\mathcal{D}_i) > n_{upper}] < \varepsilon. \tag{5.7}$$

A new client N is admitted if the following inequality holds.

$$n_{upper} \times B \leq T_{round}\Phi \tag{5.8}$$

For the statistical estimation of the total number of blocks retrieved in a round for all clients, Vin *et al.* use the central limit theorem and Chang and Zakhor compute the probability density function (pdf) by the convolution of each individual pdf for a video request. They assume that, however, all the blocks are serviced in a single disk. In a disk array environment where data blocks are serviced across multiple disks, their approach may be incorrect. We further describe how to estimate $n(\mathcal{D}_i)$ in the next section along with experiments.

The inherent feature of the multi-resolution video server enables the server to renegotiate the service resolution level with clients failed in the admission control. The server can present a lower resolution level which satisfies Eq. (5.8). Furthermore, if it is permissible to degrade the resolution level of existing clients, more clients can be serviced. Transient degradation may be required for the rounds in which the actual number of blocks to be retrieved is greater than n_{upper}. If the scheduler in each disk detects the overflow, it degrades the service level uniformly across all the clients until Eq. (5.8) is satisfied.

It is noteworthy that according to Eq. (5.8) the buffer requirement of the server is $2n_{upper}B$ per disk. This value is much smaller than that of the static policy which allocates the worst-case fixed-size buffer to each client.

5.4 Experimental evaluation

In this section, we evaluate the proposed schemes through experiments with trace data generated from actual scalable video streams. As mentioned in Subsection 5.1.1, we consider three VBR scalable compression techniques: MPEG-1 with temporal scalability, MPEG-2 with spatial and SNR scalability, and 3-D subband coding scheme. Table 5.2 shows the average bit rate of each resolution level for three kinds of trace data. To construct the multi-resolution video stream, the round length T_{round} should be determined first. The round length provides trade-off between disk throughput and buffer requirement. Chang and Zakhor [Chan97] suggest that the total system cost is minimized at T_{round} of one second from cost analysis and many other works assume one second for T_{round} [Vin94, Bolo96].

Table 5.2: Average bit rate (Mbps) of each resolution level for trace data

Resolution level	1	2	3	4/8	5/9	6/10	7/11
MPEG-1 (30fps)	0.18	0.8	1.5				
MPEG-2 (24fps)	0.32	1.152	3.008				
3-D subband (24fps)	0.190	0.253	0.316	0.380	0.506	0.633	0.760
				0.887	1.013	1.140	1.330

We also choose one second for T_{round}.

5.4.1 Disk load balancing

First, we validate Eq. (5.5) which indicates that the disk workload for any resolution service for a given video stream is evenly distributed across all the disks even for VBR case. Figure 5.6 presents the number of blocks retrieved in each disk for 30 minutes ($l = 1800$). The value of the right-most bar in each graph is calculated from Eq. (5.5). As shown in Figure 5.6, the proposed data placement scheme guarantees the disk load balancing for a video service.

Next, to explore the actual behavior of the multi-resolution video server, we have created an event-driven simulator written in C with SMPL [Mac87] libraries. The simulator models the server including data placement and retrieval. Along with three types of trace data for multi-resolution video streams in Table 5.2, the server is assumed to have eight disks and to store 24 video streams (eight for each type in Table 5.2). The video streams are placed on disks according to the data placement scheme with different starting points ($StartDisk_V$). We assume that each client randomly chooses a video stream and resolution level.

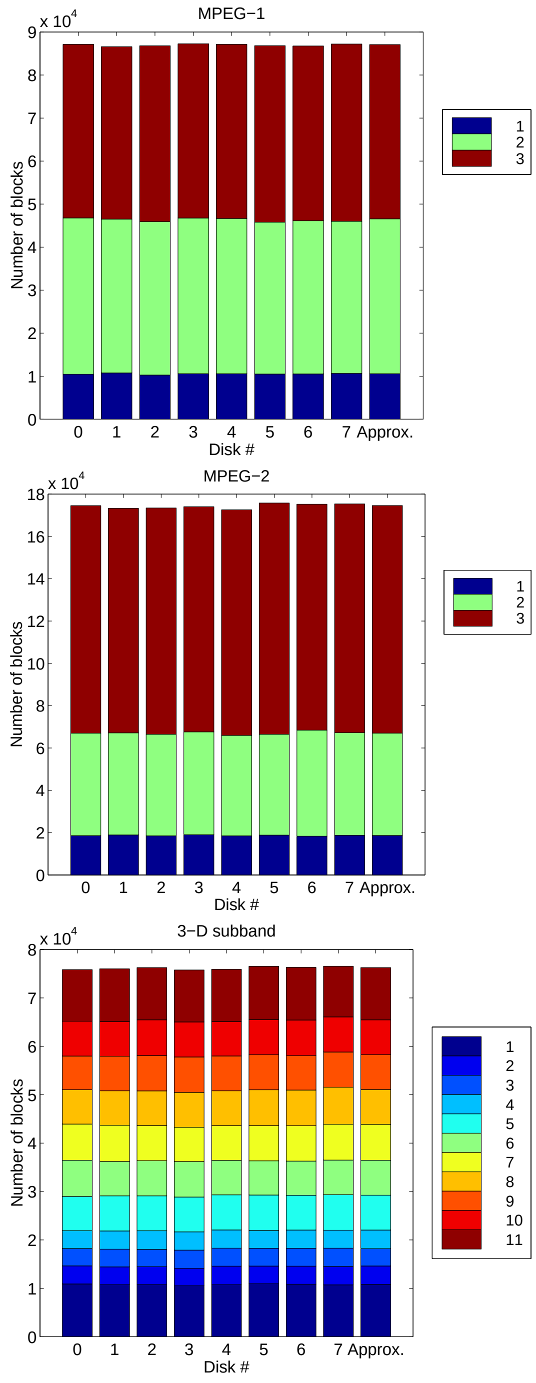
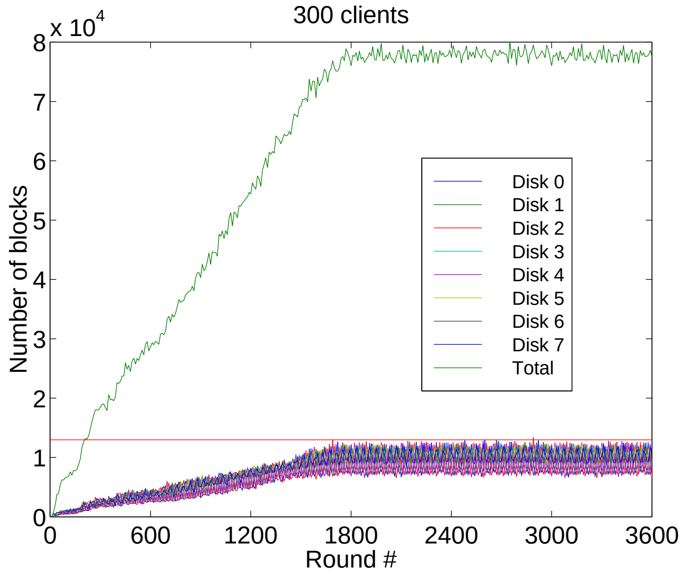
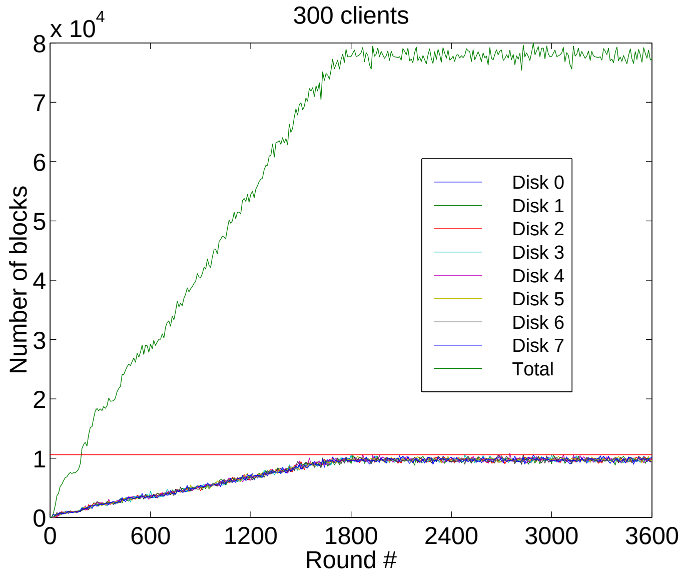


Figure 5.6: Distribution of disk workloads for a given resolution service



(a) No request scheduling



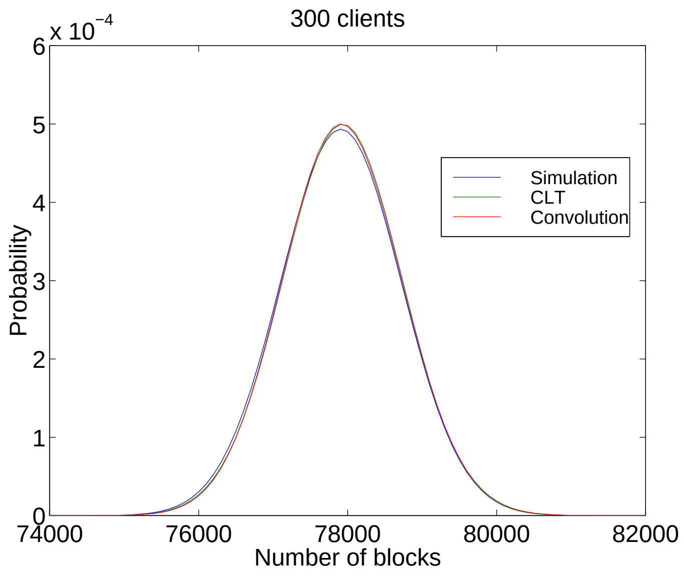
(b) Request scheduling

Figure 5.7: Distribution of disk workload for 300 clients

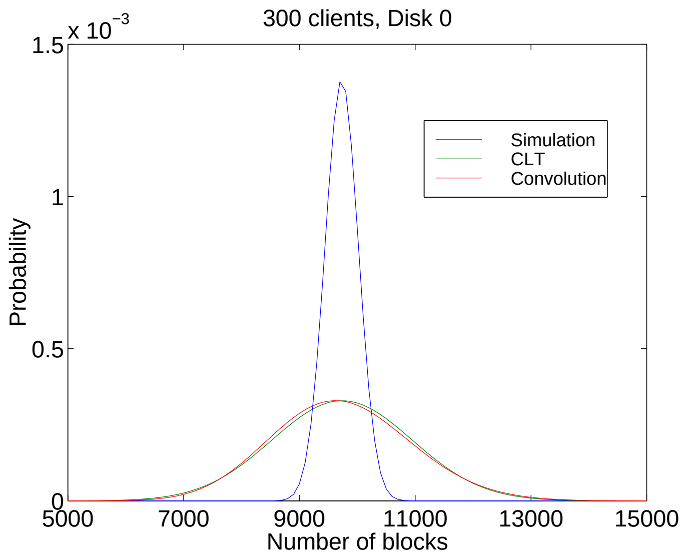
Figure 5.7 presents the number of disk blocks retrieved in a disk in each round for 300 concurrent clients. For the first 30 minutes (1800 rounds), clients arrive, while the services continue for the next 30 minutes. The fluctuation of workloads in each disk is very large from round to round in Figure 5.7(a) since we do not apply the request scheduling. This occurs because the disk workload is not evenly distributed across the disks in a given time point. At the expense of service latency, we schedule the start point of services to minimize the maximum number of blocks retrieved in a disk. By the request scheduling, we can evenly distribute the disk workloads in each round, so that the variation of workloads in a disk decreases significantly, as shown in Figure 5.7(b). It should be noted that the variation of workloads in a disk is smaller than that of the total workloads. This indicates that more clients can be serviced by the request scheduling since the most heavily loaded disks determine the number of concurrent clients in video servers.

5.4.2 Admission control

In Subsection 5.3.2, we described the admission control strategy. Since the number of blocks to be accessed at disk i in each round, or $n(\mathcal{D}_i)$, varies in VBR streams as shown in Figure 5.7, we intend to estimate an upper bound n_{upper} from Eq. (5.7) for the statistical service guarantee. First, we take two existing approaches in a single disk system: central limit theorem [Vin94] and convolution [Chan97]. Let a random variable n_j denote the number of blocks to be accessed in each round for client j. The total number of blocks for N clients is given by $n = \sum_{j=1}^{N} n_j$. Using the central limit theorem, Vin, *et al.* [Vin94] estimate



(a) single disk



(b) multiple disks

Figure 5.8: Estimation with existing schemes

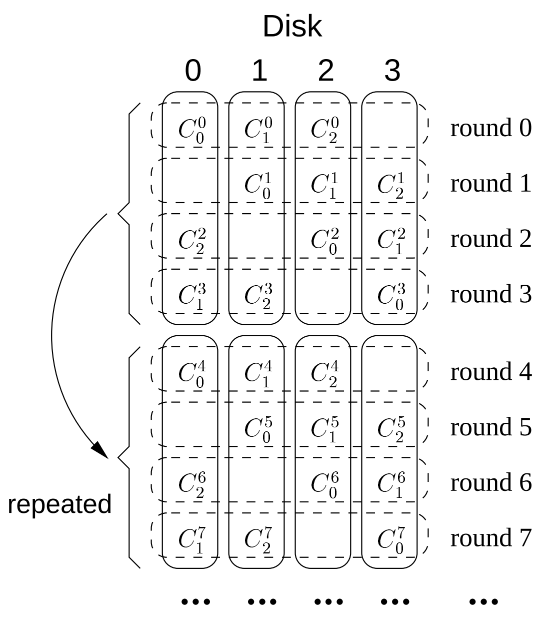


Figure 5.9: An example of the 3rd level resolution service

the distribution function of n as a normal distribution with $\mu_n = \sum_{j=1}^{N} \mu_{n_j}$ and $\sigma_n^2 = \sum_{j=1}^{N} \sigma_{n_j}^2$, where μ_n and σ_n^2 denote the mean and the variation respectively. Chang and Zakhor [Chan97] compute the pdf $f_n(x)$ by convolving $f_{n_j}(x)$ for $1 \leq j \leq N$. When all the blocks are serviced in a single service point (i.e. disk), both of the two approaches give the exact estimation as shown in Figure 5.8(a).

In a disk array environment where data blocks are serviced across multiple disks, however, they may not produce the proper estimation. In Figure 5.8(b), the estimation of two approaches shows a large difference with the result of the simulation; the approaches should be updated in a disk array environment. Observe that the disk request pattern for a video stream is repeated periodically with the period of d rounds. In Figure 5.9, for example, the first four rounds are repeated while the service proceeds. Furthermore, from the view point of each disk, the components retrieved during d rounds contain each resolution level, although they are not in the same segment, for instance in Figure 5.9, $\{C_0^0, C_1^3, C_2^2\}$ in disk 0 and $\{C_0^1, C_1^0, C_2^3\}$ in disk 1. This indicates that disk blocks for d rounds are perfectly distributed across all the disks during d rounds; the variation of the number of blocks to be accessed at a disk during d rounds is much smaller than that of the number of blocks in each round. In addition, the request scheduling evenly distributes the disk workloads for concurrent clients. Eventually, it is statistically true that n_j blocks are evenly distributed across d disks in each round. Thus, given a video stream, we can compute the mean and the variance for the number of blocks ($n_j^{'}$) to be accessed at a disk in each round as follows:

$$\mu_{n_j^{'}} = \mu_{n_j}/d, \ \ \sigma^2_{n_j^{'}} = \sigma^2_{n_j}/d. \tag{5.9}$$

Using the central limit theorem, $n(\mathcal{D}_i) = \sum_{j=1}^{N} n_j^{'}$ approaches a normal distribution $N(\mu, \sigma^2)$ where $\mu = \sum_{j=1}^{N} \mu_{n_j^{'}}$ and $\sigma^2 = \sum_{j=1}^{N} \sigma^2_{n_j^{'}}$. From Eq. (5.7), then, n_{upper} is computed by

$$\int_{n_{upper}}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \ < \ \varepsilon. \tag{5.10}$$

Finally, we can admit a new client N if n_{upper} satisfies Eq. (5.8). The statistics of the random variable n_j which denotes the number of blocks to be serviced in a round for client j can be known *a priori* from the traces for the service of V with k-resolution at the time the video stream is stored as follows:

$$\mu_{V_k} = \frac{1}{s} \sum_{s=0}^{l-1} \sum_{c=0}^{k-1} b_c^s, \ \ \sigma^2_{V_k} = \frac{1}{s} \sum_{s=0}^{l-1} \left(\left(\sum_{c=0}^{k-1} b_c^s \right) - \mu_{V_k} \right)^2 \tag{5.11}$$

Figure 5.10 exhibits that the proposed scheme gives an accurate estimation to the actual number of blocks required for the service. We compare n_{upper} calculated from Eq. (5.10) with the value measured from the simulation. As shown in Figure 5.10, regardless of the number of clients and the number of disks, the

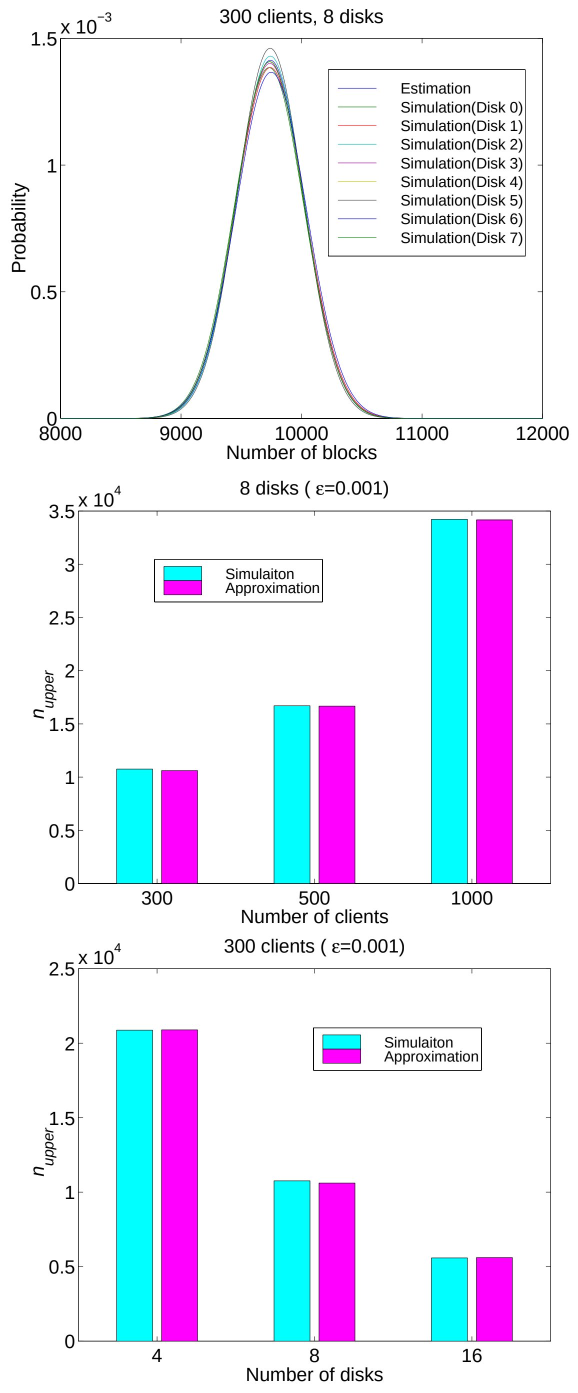


Figure 5.10: Estimation with the proposed scheme

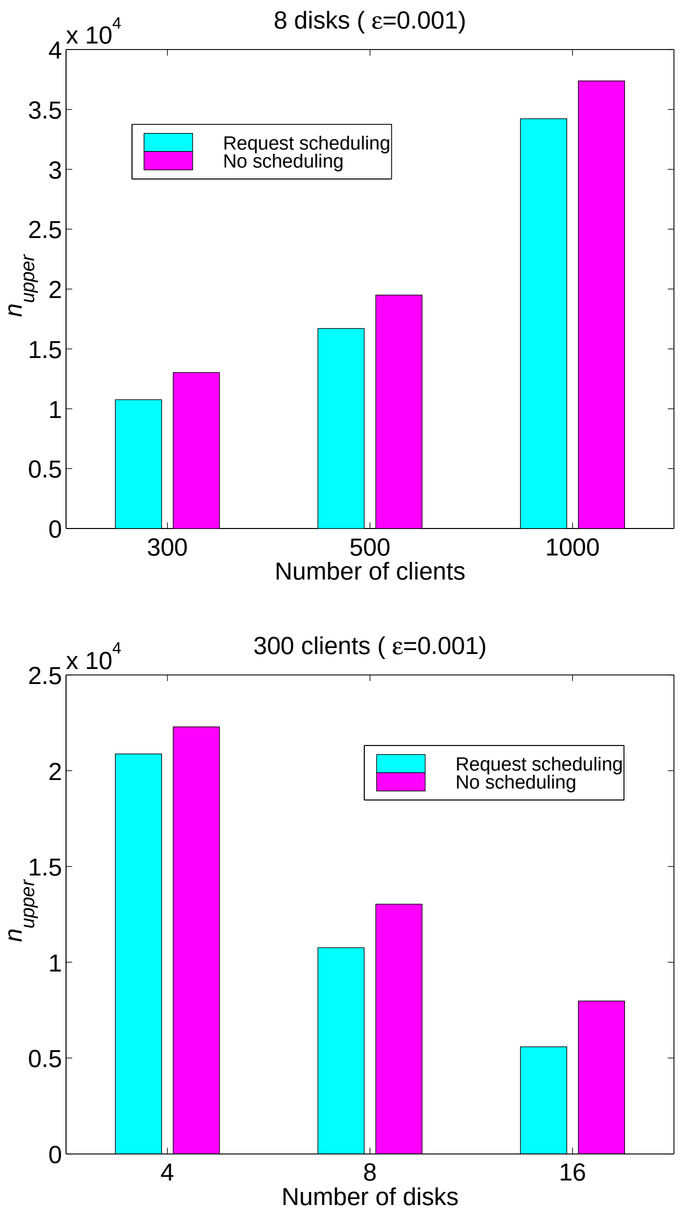


Figure 5.11: Effect of the request scheduling

admission control strategy precisely estimate the actual number of blocks to be accessed at a disk. This indicates that the server resources such as disk bandwidth and buffer memory can be fully utilized. In Figure 5.11, the effect of the request scheduling is presented. By reducing n_{upper}, the request scheduling enables the server to efficiently provide services for more clients. We can also find from Figure 5.11 that the effect of the request scheduling becomes larger as the number of disks increases, since the disk load balancing is more significant on a large number of disks. In the experiment, the request scheduling reduces the required bandwidth by about 9.8 Mbps per disk, but the average service latency increases by 2.16 seconds. We believe that it is worthwhile to increase the number of concurrent clients at the expense of acceptable service latency.

5.5 Implementation of a multi-resolution video manager: MRVman

In this section, we implement a multi-resolution video manager, or MRVman[5], to realize the proposed techniques in Chapter 5. For the quick implementation, we update DAman described in Chapter 3 to provide multi-resolution video services. Since DAman consists of several functional managers, it is easy to add or update each function. MRVman is implemented by updating the file system manager and the striping manager of DAman.

[5]Source codes for MRVman can be accessed at `http://cselab.snu.ac.kr/~cjs/research/MRVman.html`.

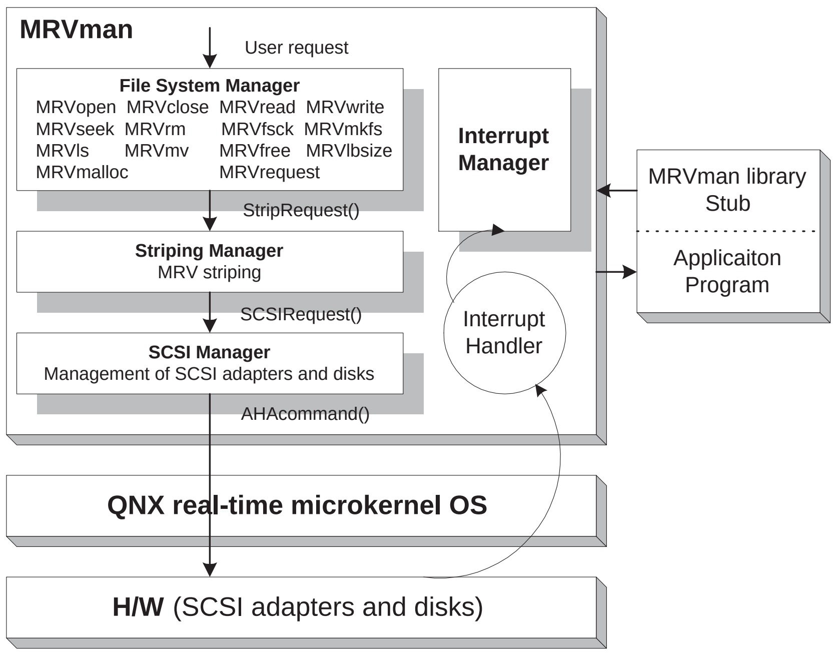


Figure 5.12: Overall architecture of MRVman

5.5.1 Overall architecture

First of all, we employ MPEG-1 streams with hardware decoder for multi-resolution video streams. As mentioned in Section 5.1.1, MPEG-1 video streams are reconstructed into the multi-resolution video model in temporal dimension. In the first prototype of the server, the resolution level is provided with high, medium, or low. MPEG-1 video streams are parsed and separated by the frame type, and then, a segment is made up of a GOP. A set of the same type frames in the GOP constructs each component in a segment, for example, (I_1), (P_1, P_2, P_3, P_4), $(B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9, B_{10})$.

Figure 5.12 depicts the overall architecture of MRVman which is similar to that of DAman. The major differences between DAman and MRVman are file

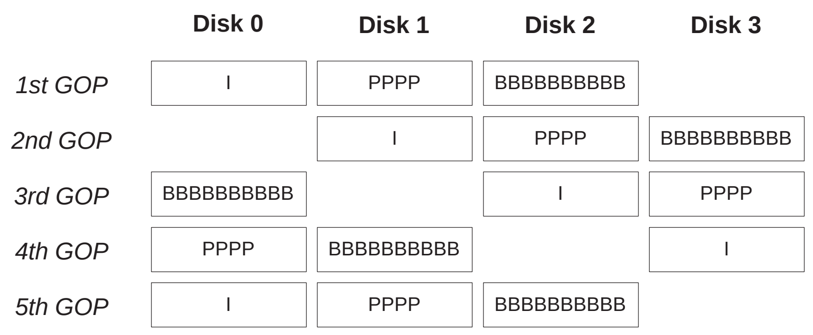


Figure 5.13: Data placement in MRVman

systems and striping policy. System calls are updated for multi-resolution video services as shown in Figure 5.12 and the striping manager places video streams according to the multi-resolution video data placement scheme proposed in Section 5.2. Further description proceeds in the next subsection.

5.5.2 Multi-resolution video file system

We begin by introducing data placement of a video file. As mentioned earlier, MPEG-1 video streams are reconstructed into the multi-resolution video stream model in temporal dimension and the resolution level is provided with high, medium, or low ($z = 3$). Since the prototype has four disks ($d = 4$), a reconstructed MPEG-1 video file is placed on disks according to the placement scheme in Figure 5.4(c) $z < d$ (see Figure 5.13). While placing video files, MRVman should maintain some meta information as followings:

```
struct meta_info {
    int picture_type;
    int size;
```

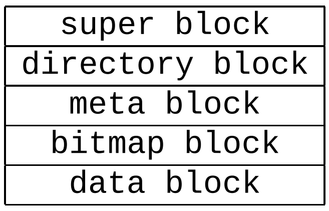


Figure 5.14: MRVman file system structure

```
int disk;
int sector_number;
int sector_count;   };
```

The striping manager of MRVman maintains such meta information. MRVman has a similar file system structure to DAman as shown in Figure 5.14. Each block has the same function with that of DAman in Section 3.4 except that the meta block contains meta information described above.

Similarly to DAman, MRVman supports 16 run-time libraries in Figure 5.15 and 7 file system utilities in Table 5.3. In what follows, we briefly describe storage and retrieval of multi-resolution video streams in MRVman.

- **Storage**: The multi-resolution video file system parses MPEG-1 video files and distributes them by the picture type as shown in Figure 5.13. Hence, storage of multi-resolution video files on MRVman can be accomplished only by `cp2mrv` command in Table 5.3. The `cp2mrv` command parses MPEG-1 video files [ISOa] and store each picture on disks in compliance with the proposed multi-resolution video data placement scheme through `mrv_write` function. At the same time, MRVman stores the related meta information on meta block.

```
int           mrv_open(char *file_name, int flag);
int           mrv_close(int handle);
int           mrv_read(int handle, char __far *buf, int max_size,
                       int resolution_level);
int           mrv_write(int handle, char __far *buf, int size,
                       int picture_type);
int           mrv_request(int n, Req_Blk_t *req_blk);
int           mrv_rewind(int handle);
int           mrv_mkfs(int logical_block_size);
int           mrv_fsck(void);
int           mrv_lbsize(void);
int           mrv_ls(DirEnt_t *dir, char *name);
int           mrv_rm(char *file_name);
int           mrv_mv(char *source, char *dest);
char __far *  mrv_malloc(int size);
int           mrv_free(char __far *pointer);
char *        far2near(char __far *pointer, int size);
int           mrv_errmsg(void);
```

Figure 5.15: Run-time libraries for MRVman

- **Retrieval**: Application programs such as `cpmrv2` and video servers retrieve multi-resolution video data only by `mrv_read` function in Figure 5.15. MRVman retrieves the given resolution of video data up to amount of buffer size. The retrieval unit is the picture. In other words, for example, when the given buffer size accommodates 10 pictures and some part of a picture, only 10 pictures are retrieved concurrently across disks in the array. In the retrieval, the picture sequence is reorganized into the original, so that the existing MPEG hardware/software decoder can work.

On the other hand, MRVman has the following features: First, it supports multi-resolution video services by reconstructing MPEG-1 video files without any special decoder. Second, it supports interactive operations without any additional cost such as disk and network bandwidth. Third, the multi-resolution video stream

Table 5.3: Utilities for MRVman

Command	Description
`cp2mrv`	copy a file to MRVman
`cpmrv2`	copy a file from MRVman
`mkmrvfs`	make a MRVman file system
`mrvls`	list directory entries
`mrvrm`	remove a file
`mrvmv`	move (rename) a file
`mrvfsck`	check MRVman file system

model allows to use only the necessary portion of data; MRVman effectively manages the server resources. Finally, by the multi-resolution video data placement scheme it can achieve load balancing among disks and thus effectively manage the aggregate disk bandwidth.

5.5.3 Multi-resolution video on-demand system

In order to verify the multi-resolution video playback of MRVman, we develop a small-scale prototype of multi-resolution video-on-demand system which consists of a server and a client. Client and server are connected through Ethernet and TCP/UDP protocols are used. On top of MRVman, a VOD server is implemented and client programs run on Windows 95 with RealMagic hardware MPEG decoder[6]. With the help of MRVman, the client program can be implemented with existing libraries supported by RealMagic MPEG decoder[7] while providing high, medium, and low resolution video services. Figure 5.16 shows the window of

[6]Sigma Designs Inc.

[7]The libraries can be accessed at `http://www.sigmadesigns.com`.

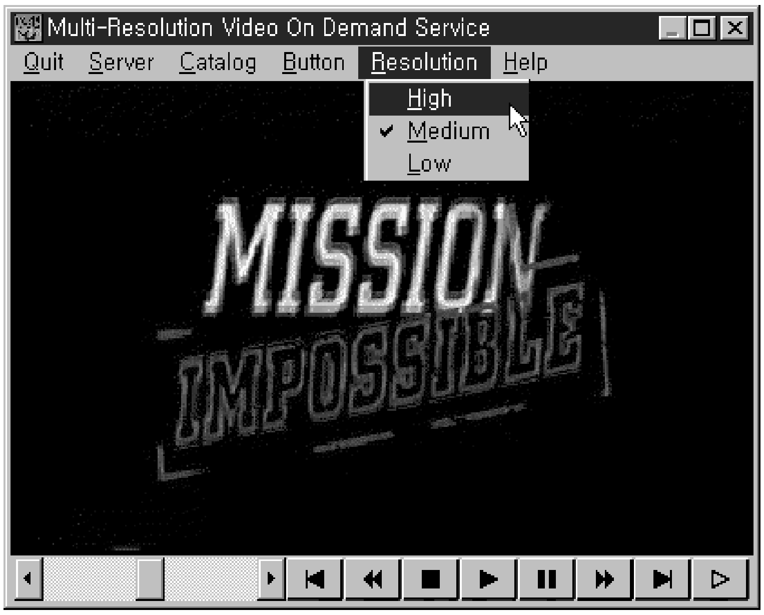


Figure 5.16: Client window in the prototype

client program in the prototype. The prototype exhibits that the visual quality of the multi-resolution playback and fastforward playback is acceptable. This gives us more insights into the proposed techniques when extended to a practical environment.

5.5.4 Empirical evaluation

This subsection measures the execution time of MRVman with a timer board which has 0.1μs granularity and analyzes the results. Meanwhile, we employ a MPEG-1 video file `mission.mpg` which is compressed by MPEG-1 encoder[8] using the part of a movie title 'Mission Impossible' about 20 minutes.

[8]OptiVision Inc.

Table 5.4: Execution time of MRVman (μs)

Functional manager	Execution time (read)
Request to MRVman	94.0
File system manager	389.1
Striping manager	4839.2
SCSI manager	320.7
Interrupt manager	16918.4

First of all, the analysis of `mission.mpg` indicates that (1) a GOP consists of 1 I picture, 4 P pictures, and 10 B pictures and (2) the average sizes of I, P, and B pictures are 16657B, 8183B, and 4399B, respectively (3.79:1.86:1). So, the playback rates of high, medium, and low resolution are 1.5Mbps, 0.8Mbps, and 0.18Mbps. On the other hand, the ratio of I, P, B picture sizes shows great differences along the type of video streams [Rose95]. Hence, according to the stream type and encoding parameters in MPEG-1 (N and M) [ISOa], various resolution levels can be provided.

Next, Table 5.4 shows the execution time of each functional manager of MRV-man. The values are the elapsed times for retrieving one segment or a GOP (15 pictures) with `mrv_read` function. As shown in Table 5.4, the relatively large portion of time is spent in the striping manager and the interrupt manager. This is because the striping manager should read meta information from meta block and the interrupt manager should copy data into user address space. For identifying data copying overhead, we compare the elapsed time of `mrv_read` with that of `mrv_write` which exploits DMA mechanism. The execution time of interrupt manager is 1672 μs in `mrv_write`; so, the data copying overhead would be 90% of the execution time of interrupt manager.

Table 5.5: Effect on storage overhead and retrieval time of logical block size

Logical block size	Storage overhead	Retrieval time
0.5 KB	3.8 %	116504.2 μs
1 KB	9.3 %	94869.7 μs
2 KB	19.2 %	104482.4 μs
4 KB	33.9 %	120853.0 μs

Table 5.6: Effect of the number of disks

Number of disks	Retrieval time
1	118159.0 μs
2	109420.7 μs
3	94574.9 μs
4	94869.7 μs

As described in Subsection 5.5.2, MRVman stores video streams by the picture. The storage unit is the logical block of which the size is usually 1$\sim$4 KB in most operating systems. In general, the size of logical block affects the storage efficiency and the read performance of disks. When the logical block size becomes larger, it is expected that the read operation performs better but the storage efficiency becomes worse. The effect on the storage efficiency and the read performance of logical block size is given in Table 5.5. Table 5.5 indicates that the logical block of 1KB reveals the best performance in the retrieval time. In addition, when the logical block size is greater than 1KB, the large storage overhead results in the large retrieval time. We can conclude that the most appropriate value of logical block size in MRVman is 1KB.

Table 5.6 presents the effect of the number of disks. Since the resolution level of a video stream is three (high, medium, and low), striping on 3 and 4 disks

Table 5.7: Effect of video resolution

Resolution	Retrieval time
High	94869.7 μs
Medium	50308.7 μs
Low	25307.7 μs

Table 5.8: Effect of multiple streams

Number of streams	Retrieval time
1	94869.7 μs
2	163205.3 μs
3	244365.6 μs
4	326490.1 μs
5	396571.3 μs

show the similar performance. However, striping on 4 disks has an idle disk in each round; it can service more clients. When the number of disks is 1 or 2, the retrieval time increases because multiple disk requests may arrive at a disk.

So far we experimented with high resolution video service. Since lower resolution service needs smaller amount of data, it is expected that the retrieval time decreases. Table 5.7 presents the result. Hence, it is verified that MRVman can service more clients by degrading the resolution levels of existing clients.

We described the retrieval time of a single video stream until now. MRVman supports multiple read/write operations by `mrv_request`. Table 5.8 presents the retrieval time of multiple streams through `mrv_request` function. It results from disk load balancing that the retrieval time does not show linear increase on the number of disks. In other words, while there is an idle disk on the service of

Table 5.9: Number of disk blocks retrieved in each disk ($l = 1000$)

Resolution	Disk 0	Disk 1	Disk 2	Disk 3
High	25247	25280	25248	25254
Medium	12954	12955	12938	12931
Low	2620	2563	2545	2598

one stream, the service of multiple streams activates all the disks in the array.

Finally, Table 5.9 validates once more the disk load balancing property of MRVman although it is validated through simulation in Subsection 5.4.1.

Chapter 6

Conclusions

In this thesis we have addressed the problems of designing video servers in various environments by providing efficient storage and retrieval of video data. The followings summarize the main results obtained from the thesis:

- A simple performance analysis of disk arrays through simulation studies for a single server architecture recommended that (1) the number of disks should be less than four, (2) SCAN is a competitive disk scheduling algorithm, (3) the striping policy should be AID5, (4) 1$\sim$2 tracks are appropriate for the striping unit size, and (5) each video stream should be placed contiguously.

- Implementation of a disk array manager (DAman) validated the above results. Although the absolute values in the graphs obtained from simulation studies and performance measure of DAman are different from each other, the shapes of the graphs (*i.e.* the tendency to the effects of parameters on performance) are similar.

- On top of DAman, a video server has been developed. By integrating the server with a VOD system, we figured out the behavior of video servers and obtained some feedbacks.
- For a large-scale server, storage and retrieval in a parallel server have been proposed including data placement, retrieval scheduling, and communication scheduling. By the proposed scheduling algorithms, disk bandwidth in a parallel server can be fully utilized and communication between nodes in a parallel server is guaranteed conflict-free.
- Given a large number of nodes, the configuration of a large-scale server has been described. In other words, we addressed how to cluster such nodes into server clusters (parallel servers). The analysis indicated that clients experience relatively large service latency when the number of server clusters is small, that is, the size of a server cluster is large. On the other hand, when the number of server clusters is large, client requests are not balanced among server clusters, *i.e.* there exist hot spots although popular videos are replicated. The tradeoff of large versus small clusters provides a basis for the design of the most effective server configuration.
- A queueing analysis for the large-scale video server has been conducted with a server cluster being an independent service entity. An open queueing network model has been developed which consists of M/M/1 queues and Poisson input processes. From the model, we derived the packet loss probability that a packet request is not serviced within its deadline. The queueing

analysis revealed, as can be expected, that the parameters which greatly affect the performance of large-scale video servers are the disk bandwidth and the access network bandwidth. The proper combination of them should be derived.

- The benefits of employing multi-resolution video have been identified: heterogeneous client support, storage efficiency, adaptive service, and interactive operations support.

- For the purpose of modeling multi-resolution video, we proposed a z-level multi-resolution video stream model. In the model, each video stream can be provided with z levels of quality and the QoS parameter is represented by the number of components in a segment. We also described how to build the proposed multi-resolution video stream model using the current scalable video compression techniques including DCT-based scheme, subband schemes, fractal-based schemes, and object-based schemes.

- We addressed the issues on storage and retrieval of multi-resolution video. The placement scheme exploits both concurrency and parallelism offered by striping data across disks and achieves the disk load balancing during any resolution video service. The deterministic access property of the placement scheme also permits the retrieval scheduling to be performed on each disk independently and to support interactive operations simply by reconstructing the input parameters of the scheduler. In addition, we developed an efficient admission control algorithm which precisely estimates the actual disk workload for the given resolution services. The proposed schemes have been validated through simulation studies with trace data generated from actual scalable video streams.

- Based on storage and retrieval schemes of multi-resolution video, a multi-resolution video manager (MRVman) has been developed. A prototype of the multi-resolution VOD system exhibited that the visual quality of multi-resolution playback and fastforward playback is acceptable.

Throughout this thesis, we have assumed that disk storage and retrieval of video data are the major bottleneck on the performance of video servers. Another system resource, or buffer memory should be effectively managed and the hierarchical storage management of memory, disk, and tape storage should be also treated carefully. The effective management of them remains to be solved in the future.

Appendix A

Proof of Theorem 5.1

Proof: i) $z = d$: Without loss of generality, we assume that $StartDisk_V = 0$. Then, for $V = \{C_c^s \mid 0 \le s < l,\ 0 \le c < z\}$, $\mathcal{V}_{i,k}$ is given from Eq. (5.1) and (5.2) as follows:

$$\mathcal{V}_{i,k} = \{C_c^s \mid 0 \le s < l,\ 0 \le c < k,\ c = [i-s]_d\}. \tag{A.1}$$

Therefore, $|\mathcal{V}_{i,k}|$ is the number of s's, $0 \le s < l$, which satisfies

$$[i-s]_d \ < \ k. \tag{A.2}$$

Let $s = x \cdot d + y$, $(0 \le x < \left\lfloor \frac{l}{d} \right\rfloor$, $0 \le y < d)$ and apply it to Eq. (A.2).

$$[i - x \cdot d - y]_d < k \tag{A.3}$$

If $l = m \cdot d$, for each x, $0 \le x < \left\lfloor \frac{l}{d} \right\rfloor$, the number of y's, $0 \le y < d$, which satisfies Eq. (A.3) is k. If $l \ne m \cdot d$, for each x, $0 \le x < \left\lfloor \frac{l}{d} \right\rfloor$, the number of y's is k and for $x = \left\lfloor \frac{l}{d} \right\rfloor$, the number of y's is α, $0 < \alpha < d$. Hence, $|\mathcal{V}_{i,k}| = \left\lfloor \frac{l}{d} \right\rfloor \times k + \alpha$

$(0 \le \alpha < d)$.

ii) $z < d$: This is equivalent to the case where $z' = d$ and $k \le z$. From the result of case i), $|\mathcal{V}_{i,k}| = \left\lfloor \frac{l}{d} \right\rfloor \times k + \alpha$ $(0 \le \alpha < d)$.

iii) $z > d$: From Eq. (5.1) and (5.2),

$$\mathcal{V}_{i,k} = \{C_c^s \mid 0 \le s < l,\ 0 \le c < k,\ c = [i-s]_d + a \cdot d,\ 0 \le a < \left\lceil \frac{z}{d} \right\rceil\}. \quad \text{(A.4)}$$

If $k \le d$, this case is equivalent to case i) because each disk retrieves one component in a segment. If $k > d$, each disk retrieves one or more components in a segment. From Eq. (A.4), $l \cdot \left\lfloor \frac{k}{d} \right\rfloor$ components $\{C_c^s \mid 0 \le s < l,\ c = [i-s]_d + a \cdot d, 0 \le a < \left\lfloor \frac{k}{d} \right\rfloor\}$ are retrieved and additional components $\{C_c^s \mid 0 \le s < l,\ c = [i-s]_d + \left\lfloor \frac{k}{d} \right\rfloor \cdot d < k\}$ are also retrieved. According to the result of case i), the number of the additional components is $\left\lfloor \frac{l}{d} \right\rfloor \times (k - \left\lfloor \frac{k}{d} \right\rfloor \cdot d) + \alpha'$ $(0 \le \alpha' < d)$. By integrating two terms, we can obtain the total number of components, $\left\lfloor \frac{l}{d} \right\rfloor \times k + \alpha$ $(0 \le \alpha < d)$.
$\square$

Bibliography

[Abra98] E. L. Abram-Profeta and K. G. Shin, "Providing unrestricted VCR functions in multicast video-on-demand servers," In *Proc. of International Conference on Multimedia Computing and Systems*, pages 66–75, 1998.

[Agne96] P. W. Agnew and A. S. Kellerman, *Distributed Multimedia: Technologies, Applications, and Opportunities in the Digital Information Industry*, Addison-Wesley Publishing Company, 1996.

[Ahn95] S. Ahn, Y. Lee, J. Cho, T. Kim, and H. Shin, "Design and implementation of a real-time storage server for digital audio/video using disk array technology," *Journal of KISS(C)*, 1(1):35–45, 1995.

[Alle90] A. O. Allen, *Probability, Statistics, and Queueing Theory with Computer Science Applications*, 2nd Ed., Academic Press, Inc., 1990.

[Ande92] D. P. Anderson, Y. Osawa, and R. Govindan, "A file system for continuous media," *ACM Transactions on Computer Systems*, 10(4):311–337, 1992.

[Andl96] P. K. Andleigh and K. Thakrar, *Multimedia Systems Design*, Prentice Hall, 1996.

[Beck98] C. J. Beckmann, A. A. Moin, and S. Nog, “Bandwidth reservation with selectable bit-rate streams,” *Multimedia Systems*, 6(4):219–231, 1998.

[Bern93] P. J. Bernhard, “Bounds on the performance of message routing heuristics,” *IEEE Transactions on Computers*, 42(10):1253–1256, 1993.

[Bers94] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju, “Staggered striping in multimedia information systems,” In *Proc. of ACM SIG-MOD ’94*, 1994.

[Bers95] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju, “Staggered striping: A flexible technique to display continuous media,” *Multimedia Tools and Applications*, 1(2):127–148, 1995.

[Bogd94] A. Bogdan, “Multiscale (intrer/intra-frame) fractal video coding,” In *Proc. of IEEE International Conference on Image Processing*, 1994.

[Bolo96] W. J. Bolosky and et. al, “The Tiger video fileserver,” In *Proc. of International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 97–104, 1996.

[Bufo94] J. F. K. Buford, *Multimedia Systems*, Addison-Wesley Publishing Company, 1994.

[Cata95] V. Catania, A. Puliafito, S. Riccobene, and L. Vita, "Design and performance analysis of a disk array system," *IEEE Transactions on Computers*, 44(10):1236–1247, 1995.

[Chan94] E. Chang and A. Zakhor, "Scalable video data placement on parallel disk arrays," In *Proc. of IS&T/SPIE International Symposium on Electronic Imaging: Science and Technology*, pages 208–221, 1994.

[Chan96a] E. Chang and A. Zakhor, "Cost analyses for VBR video servers," *IEEE Multimedia Magazine*, 4(3):56–71, 1996.

[Chan96b] E. Chang, *Storage and Retrieval of Compressed Video*, PhD thesis, University of California at Berkeley, 1996.

[Chan97] E. Chang and A. Zakhor, "Disk-based storage for scalable video," *IEEE Transactions on Circuits and Systems for Video Technology*, 7(5):758–770, 1997.

[Chen93] M.-S. Chen, D. D. Kandlur, and P. S. Yu, "Optimization of the grouped sweeping scheduling (GSS) with heterogeneous multimedia streams," In *Proc. of ACM Multimedia '93*, pages 235–242, 1993.

[Chen94] M.-S. Chen, D. D. Kandlur, and P. S. Yu, "Support for fully interactive playout in a disk-array-based video server," In *Proc. of ACM Multimedia '94*, pages 391–398, 1994.

[Chen95] M.-S. Chen, D. D. Kandlur, and P. S. Yu, "Using rate staggering to store scalable video data in a disk-array-based video server," In *Proc. of IS&T/SPIE Symposium on Electronic Imaging Conference on Multimedia Computing and Networking*, pages 338–345, 1995.

[Chen96] M.-S. Chen and D. D. Kandlur, "Stream conversion to support interactive video playout," *IEEE Multimedia Magazine*, 3(2):51–58, 1996.

[Cher95] A. L. Chervenak, D. A. Patterson, and R. H. Katz, "Choosing the best storage system for video service," In *Proc. of ACM Multimedia '95*, pages 109–119, 1995.

[Chia94] T. Chiang and D. Anastassiou, "Hierarchical coding of digital television," *IEEE Communications Magazine*, 32(5):38–45, 1994.

[Chiu93] T.-C. Chiueh and R. H. Katz, "Multi-resolution video representation for parallel disk arrays," In *Proc. of ACM Multimedia '93*, pages 401–409, 1993.

[Cho94] J. Cho and H. Shin, "A scheduling method for real-time multimedia storage server using disk arrays," *Journal of KISS*, 21(11):1981–1989, 1994.

[Cho95] J. Cho, T. Kim, Y. Kim, M. Sung, and H. Shin, "Performance analysis of disk arrays for storage architecture of multimedia servers," In *Proc. of the 22nd KISS Fall Conference*, pages 823–826, 1995.

[Cho96] J. Cho, T. Kim, Y. Kim, M. Sung, and H. Shin, "A disk array manager on microkernel environment for video servers," In *Proc. of the 23rd KISS Spring Conference*, pages 335–338, 1996.

[Cho97a] J. Cho and H. Shin, "Scheduling algorithms in a large-scale VOD server," In *Procs. of the IPPS'97 Workshop on Parallel Processing and Multimedia*, pages 17–25, 1997.

[Cho97b] J. Cho and H. Shin, "Queueing model of a large-scale VOD server," In *Proc. of the 24th KISS Spring Conference*, pages 383–386, 1997.

[Cho97c] J. Cho and H. Shin, "Design issues for multimedia information servers in mobile computing environment," In *Proc. of the 4th International Workshop on Mobile Multimedia Communications*, pages 336–339, 1997.

[Cho97d] J. Cho and H. Shin, "Heuristic scheduling for multimedia streams with firm deadlines," In *Procs. of the 4th International Workshop on Real-Time Computing Systems and Applications*, pages 67–72, 1997.

[Cho97e] J. Cho and H. Shin, "Scheduling video streams in a large-scale video-on-demand server," *Parallel Computing*, 23(12):1743–1755, 1997.

[Cho98a] J. Cho and H. Shin, "Performance analysis of a large-scale video-on-demand server using queueing model," *Journal of KICS*, 23(1):155–161, 1998.

[Cho98b] J. Cho and H. Shin, "MRVman: A multi-resolution video manager for MPEG-1 streams," In *Proc. of KISS SIGCS Fall Conference*, pages 237–245, 1998.

[Cho98c] J. Cho and H. Shin, "Temporal multi-resolution video playback based on reconstructing MPEG-1 streams," *Journal of KISS(C)*, 4(4):439–448, 1998.

[Cho99a] J. Cho and H. Shin, "A multi-resolution video scheme for multimedia information servers in mobile computing environment," In *Proc.*

of International Conference on Telecommunications, pages 388–392, 1999.

[Cho99b] J. Cho and H. Shin, "A design framework for multi-resolution video servers," *submitted for publication*, 1999.

[Chun96] S. M. Chung, *Multimedia Information Storage and Management*, Kluwer Academic Publishers, 1996.

[Coul94] G. Coulson, G. S. Blair, P. Robin, and D. Shepherd, "Supporting continuous media applications in a micro-kernel environment," Technical report, Lancaster University Computing Dept., Internal Report Number MPG-94-16, 1994.

[Dan95] A. Dan, D. Dias, R. Mukherjee, D. Sitaram, and R. Tewari, "Buffering and caching in large-scale video servers," In *Proc. of IEEE CompCon '95*, pages 217–224, 1995.

[Dan97] A. Dan, E. Eshel, J. Hollan, R. Kenneson, M. Kienzle, J. McAssey, R. Rose, D. Sitaram, and W. Tetzlaff, "The research server complex manager for large-scale multimedia servers," Technical report, IBM Research, No. RC20705, 1997.

[Delg94] L. Delgrossi, C. Halstrick, D. Hehmann, R. G. Herrtwich, O. Krone, J. Sandvoss, and C. Vogt, "Media scaling in a multimedia communication system," *Multimedia Systems*, 2(4):172–180, 1994.

[Dey94] J. K. Dey-Sircar, J. D. Salehi, J. F. Kurose, and D. Towsley, "Providing VCR capabilities in large-scale video servers," In *Proc. of ACM Multimedia '94*, pages 25–32, 1994.

[Doga93] Y. N. Doganata and A. N. Tantawi, “A video server cost/performance estimator tool,” *Multimedia Tools and Applications*, 1(2):127–148, 1993.

[Free95] C. S. Freedman and D. J. DeWitt, “The SPIFFI scalable video-on-demand system,” In *Proc. of 1995 ACM SIGMOD*, pages 352–363, 1995.

[Gall91] D. L. Gall, “MPEG: A video compression standard for multimedia applications,” *Communications of ACM*, 34(4):46–58, 1991.

[Gemm92] D. J. Gemmell and S. Christodoulakis, “Principles of delay-sensitive multimedia data storage and retrieval,” *ACM Transactions on Information Systems*, 10(1):51–90, 1992.

[Gemm95] D. J. Gemmell, H. M. Vin, D. D. Kandlur, P. V. Rangan, and L. A. Rowe, “Multimedia storage servers: A tutorial,” *IEEE Computer Magazine*, 28(5):40–49, 1995.

[Ghan93] S. Ghanderharizadeh and L. Ramos, “Continuous retrieval of multimedia data using parallelism,” *IEEE Transactions on Knowledge and Data Engineering*, 5(4):658–669, 1993.

[Ghan94] S. Ghandeharizadeh and C. Shahabi , “On multimedia repositories, personal computers, and hierarchical storage systems,” In *Proc. of ACM Multimedia ’94*, pages 407–416, 1994.

[Gros97] W. I. Grosky, R. Jain, and R. Mehrotra, *The Handbook of Multimedia Information Management*, Prentice Hall, 1997.

[Han95] C.-C. Han and K. G. Shin, "Scheduling MPEG-compressed video streams with firm deadline constraints," In *Proc. of ACM Multimedia '95*, pages 411–422, 1995.

[Hamd95] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," *IEEE Transactions on Computers*, 44(12):1443–1451, 1995.

[Harr93] P. G. Harrison and N. M. Patel, *Performance Modeling of Communication Networks and Computer Architectures*, Addison-Wesley Publishing Company, 1993.

[Hask93] R. L. Haskin, "The Shark continuous-media file server," In *Proc. of Spring COMPCON '93*, pages 12–15, 1993.

[Heyb96] A. Heybey, M. Sullivan, and P. England, "Calliope: A distributed, scalable multimedia server," In *Proc. of USENIX 1996 Annual Technical Conference*, 1996.

[Hunt] J. Hunter, V. Witana, and M. Antoniades, "A review of video streaming over the internet," White paper, `http://www.dstc.edu.au/RDU/staff/jane-hunter/video-streaming. html`.

[Huyn94] K. D. Huynh and T. M. Khoshgoftaar, "Performance analysis of advanced I/O architecture for PC-based video servers," *Multimedia Systems*, 2(1):36–50, 1994.

[Hwan93] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, Inc., 1993.

[ISOa] ISO/IEC 11172, *Information Technology - Coding of Moving Pictures and Associated Audio for Digital Storage Media at Up to about 1.5 Mbits/s*.

[ISOb] ISO/IEC 13818, *Information Technology - Generic Coding of Moving Pictures and Associated Audio*.

[Kand93] D. D. Kandlur, M.-S. Chen, and Z.-Y. Shae, "Design and a multimedia storage server," IBM Research Report, RC 18158, 1993.

[Kane96] H. Kaneko and J. A. Stankovic, "Integrating scheduling of multimedia and hard real-time tasks," In *Proc. of the 17th Real-Time Systems Symposium*, pages 206–217, 1996.

[Keet93] K. Keeton and R. H. Katz, "The evaluation of video layout strategies on a high-bandwidth file server," In *Proc. of International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 237–248, 1993.

[Kim97a] T. Kim, J. Cho, M. Sung, S. Jung, K. Kim, and H. Shin, "Design and implementation of a scalable multi-purpose multimedia-on-demand system," In *Proc. of the 24nd KISS Fall Conference*, pages 519–522, 1997.

[Kim97b] J.-W. Kim, Y.-U. Lho, and K.-D. Chung, "An efficient video block placement scheme on VOD server based on multi-zone recording disks," In *Proc. of International Conference on Multimedia Computing and Systems*, pages 29–36, 1997.

[Klei75] L. Kleinrock, *Queueing Systems*, Volume I, John Wiley & Sons, Inc., 1975.

[Kwon97] T.-G. Kwon, Y. Choi, and S. Lee, “Disk placement for arbitrary-rate playback in an interactive video server,” *Multimedia Systems*, 5(4):271–281, 1997.

[Lau97] S.-W. Lau and J. C. S. Lui, “Scheduling and data layout policies for a near-line multimedia storage architecture,” *Multimedia Systems*, 5(5):310–323, 1997.

[Laur94] A. Laursen, J. Olkin, and M. Porter, “Oracle media server: Providing consumer based interactive access to multimedia data,” In *Proc. of ACM SIGMOD 94*, pages 194–201, 1994.

[Lawr75] D. H. Lawrie, “Access and alignment of data in an array processor,” *IEEE Transactions on Computers*, 24(12):1145–1155, 1975.

[Laza94] M. S. Lazar and L. T. Bruton, “Fractal block coding of digital video,” *IEEE Transactions on Circuits and Systems for Video Technology*, 4(3):297–308, 1994.

[Lee97] H.-J. Lee and D. H. C. Du, “The effect of disk scheduling scheme on a video server for supproting quality MPEG video accesses,” In *Proc. of International Conference on Multimedia Computing and Systems*, pages 194–201, 1997.

[Lee98] J. Y. B. Lee, “Parallel video server: A tutorial,” *IEEE Multimedia Magazine*, 5(2):20–28, 1998.

[Lian97] J. Liang, "Highly scalable image coding for multimedia applications," In *Proc. of ACM Multimedia '97*, pages 11–19, 1997.

[Litt93] T. D. C. Little and D. Venkatesh, "Probabilistic assignment of movies to storage devices in a video-on-demand system," In *Proc. of the International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 213–224, 1993.

[Loug92] P. Lougher and D. Shepherd, "The design and implementation of a continuous media storage server," In *Proc. of International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 63–74, 1992.

[Loug93] P. K. Lougher, *The Design of a Storage Server for Continuous Media*, PhD thesis, Department of Computing, Lancaster University, 1993.

[Mac87] M. H. MacDougall, *Simulating Computer Systems: Techniques and Tools*, MIT Press, 1987.

[Maka97] D. Makaroff, G. Neufeld, and N. Hutchinson, "An evaluation of VBR disk admission algorithms for continuous media file servers," In *Proc. of ACM Multimedia '97*, pages 143–154, 1997.

[Mc96] S. R. McCanne, *Scalable Compression and Transmission of Internet Multicast Video*, PhD thesis, University of California at Berkeley, 1996.

[Mour96] A. N. Mourad, "Issues in the design of a storage server for video-on-demand," *Multimedia Systems*, 4(2):70–86, 1996.

[Mok96] A. K. Mok and D. Chen, “A multiframe model for real-time tasks,” In *Proc. of the 17th Real-Time Systems Symposium*, pages 22–29, 1996.

[Neuf96] G. Neufeld, D. Makaroff, and N. Hutchinson, “Design of a variable bit rate continuous media file server for an ATM network,” In *Proc. of IS&T/SPIE Multimedia Computing and Networking*, pages 370–380, 1996.

[Ng96] R. T. Ng and J. Yang, “An analysis of buffer sharing and prefetching techniques for multimedia systems,” *Multimedia Systems*, 4(2):55–69, 1996.

[Ozde95] B. Ozden, R. Rastogi, and A. Silberschatz, “Research issues in multimedia storage servers,” *ACM Computing Surveys*, 1995.

[Ozde96] B. Ozden, R. Rastogi, and A. Silberschatz, “Buffer replacement algorithms for multimedia storage systems,” In *Proc. of International Conference on Multimedia Computing and Systems*, pages 172–180, 1996.

[Paek95] S. Paek, P. Bocheck, and S. F. Chang, “Scalable MPEG2 video servers with heterogeneous QoS on parallel disk arrays,” In *Proc. of International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 363–374, 1995.

[Paek96] S. Paek and S.-F. Chang, “Video server retrieval scheduling for variable bit rate scalable video,” In *Proc. of Internaltional Conference on Multimedia Computing and Systems*, pages 108–112, 1996.

[Pan98] H. Pan, L. H. Ngoh, and A. A. Lazar, "A buffer-inventory-based dynamic scheduling algorithm for multimedia-on-demand servers," *Multimedia Systems*, 6(2):125–136, 1998.

[Patt88] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," In *Proc. of ACM SIGMOD*, pages 109–116, 1988.

[Pete85] J. L. Peterson and A. Silberschatz, *Operating System Concepts*, 2nd Ed., Addison-Wesley Publishing Company, 1985.

[QNX93] "QNX System architecture," *QNX Operating Systems Manual*, QNX Software Systems Ltd., 1993.

[Rang91a] P. V. Rangan and H. M. Vin, "Designing file systems for digital video and audio," In *Proc. of ACM Symposium on Operating Systems Principles*, pages 69–79, 1991.

[Rang91b] P. V. Rangan, W. A. Burkhard, W. Bowdidge, and et. al, "A testbed for managing digital video and audio storage," In *Proc. of USENIX Summer*, pages 199–208, 1991.

[Rang92] P. V. Rangan, H. M. Vin, and S. Ramanathan, "Designing an on-demand multimedia service," *IEEE Communications Magazine*, 30(7):56–65, 1992.

[Rang93] P. V. Rangan and H. M. Vin, "Efficient storage techniques for digital continuous multimedia," *IEEE Transactions on Knowledge and Data Engineering*, pages 564–573, 1993.

[Redd93] A. L. N. Reddy and J. Wyllie, “Disk scheduling in a multimedia I/O system,” In *Proc. of ACM Multimedia ’93*, pages 225–233, 1993.

[Redd95] A. L. N. Reddy, “Scheduling and data distribution in a multiprocessor video server,” In *Proc. of International Conference on Multimedia Computing and Systems*, pages 256–263, 1995.

[Redd97] A. L. N. Reddy, “Evaluation of caching strategies for an internet server,” In *Proc. of International Conference on Multimedia Computing and Systems*, pages 118–125, 1997.

[Rosa96] J. M. D. Rosario and G. Fox, “Constant bit rate network transmission of variable bit rate continuous media in video-on-demand servers,” *Multimedia Tools and Applications*, 2(3):215–232, 1996.

[Rose95] O. Rose, “Statistical properties of MPEG video traffic and their impact on traffic modeling in ATM systems,” Technical report, University of Wuerzburg, Institute of Computer Science Research, Report No. 101, 1995.

[Ruem94] C. Ruemmler and J. Wilkes, “An introduction to disk drive modeling,” *IEEE Computer Magazine*, 27(3):17–28, 1994.

[Shen95] P. J. Shenoy and H. M. Vin, “Efficient support for scan operations in video servers,” In *Proc. of ACM Multimedia ’95*, pages 131–140, 1995.

[Shen98] P. J. Shenoy and H. M. Vin, “Efficient support for interactive operations in multi-resolution video servers,” *Multimedia Systems*, accepted for publication, 1998.

[Sriv97] A. Srivastava, A. Kumar, and A. Singru, “Design and analysis of a video-on-demand server,” *Multimedia Systems*, 5(4):238–254, 1997.

[Tan96] W. Tan, E. Chang, and A. Zakhor, “Real-time software implementation of scalable video codec,” In *Proc. of International Conference on Image Processing*, pages 17–20, 1996.

[Taub94] D. Taubman and A. Zakhor, “Multirate 3-D subband coding of video,” *IEEE Transactions on Image Processing*, 3(5):572–588, 1994.

[Tind93] K. Tindell, A. Burns, and R. Davis, “Fixed priority scheduling of hard real-time multimedia disk traffic,” In *Proc. of Workshop on the Role of Real-Time in Multimedia/Interactive Computing Systems*, 1993.

[Tewa96a] R. Tewari, R. Mukherjee, D. M. Dias, and H. M. Vin, “Design and performance tradeoffs in clustered video servers,” In *Proc. of the International Conference on Multimedia Computing and Systems*, pages 144–150, 1996.

[Tewa96b] R. Tewari, R. King, D. Kandlur, and D. M. Dias, “Placement of multimedia blocks on zoned disks,” In *Proc. of IS&T/SPIE Multimedia Computing and Networking*, 1996.

[Toba93] F. A. Tobagi, J. Pang, R. Baird, and M. Gang, “Streaming RAID^{TM} - a disk array management system for video files,” In *Proc. of ACM Multimedia ’93*, pages 393–400, 1993.

[Tong98] S.-R. Tong and Y.-F. Huang, “Study on disk zoning for video servers,” In *Proc. of International Conference on Multimedia Computing and Systems*, pages 86–95, 1998.

[Vin93] H. M. Vin and P. V. Rangan, “Designing a multiuser HDTV storage server,” *IEEE Journal on Selected Areas in Communications*, 11(1):153–164, 1993.

[Vin94] H. M. Vin, P. Goyal, A. Goyal, and A. Goyal, “A statistical admission control algorithm for multimedia servers,” In *Proc. of ACM Multimedia ’94*, pages 33–40, 1994.

[Vin95] H. M. Vin, S. S. Rao, and P. Goyal, “Optimizing the placement of multimedia objects on disk arrays,” In *Proc. of International Conference on Multimedia Computing and Systems*, pages 158–165, 1995.

[Wang96] J. Z. Wang, K. A. Hua, and H. C. Young, “SEP: A space efficient piplelining technique for managing disk buffers in multimedia servers,” In *Proc. of International Conference on Multimedia Computing and Systems*, pages 598–607, 1996.

[Wang97a] Q. Wang and M. Ghanbari, “Scalable coding of very high resolution video using the virtual zerotree,” *IEEE Transactions on Circuits and Systems for Video Technology*, 7(5):719–727, 1997.

[Wang97b] Y. Wang, J. C. L. Liu, D. H. C. Du, and J. Hsieh, “Efficient video file allocation schemes for video-on-demand services,” *Multimedia Systems*, 5(5):283–296, 1997.

[Wang97c] J. Z. Wang and K. A. Hua, “A bandwidth management technique for hierarchical storage in large-scale multimedia servers,” In *Proc. of International Conference on Multimedia Computing and Systems*, pages 261–268, 1997.

[Wu80] C. L. Wu and T. Y. Feng, “On a class of multistage interconnection networks,” *IEEE Transactions on Computers*, 29(8):694–702, 1980.

[Wu96] K.-L. Wu and P. S. Yu, “Consumption-based buffer management for maximizing system throughput of a multimedia system,” In *Proc. of International Conference on Multimedia Computing and Systems*, pages 161–171, 1996.

[Wu97] M.-Y. Wu and W. Shu, “Scheduling for interactive operations in parallel video servers,” In *Proc. of International Conference on Multimedia Computing and Systems*, pages 178–185, 1997.

[Wu98] K.-L. Wu and P. S. Yu, “Increasing multimedia system throughput with consumption-based buffer management,” *Multimedia Systems*, 6(6):421–428, 1998.

[Yu92] P. S. Yu, M.-S. Chen, and D. D. Kandlur, “Design and analysis of a grouped sweeping scheme for multimedia storage management,” In *Proc. of International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 38–49, 1992.