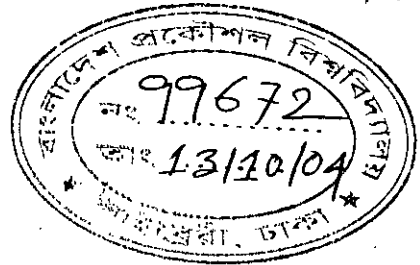


# A HYBRID ADMISSION CONTROL ALGORITHM

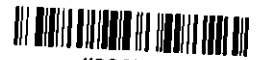
for

## MULTIMEDIA SERVER

**Dewan Tanvir Ahmed**



A Thesis Submitted to the Department of Computer Science and Engineering in the  
Partial Fulfillment of the Requirements for the  
Degree of  
Master of Science in Engineering  
(Computer Science and Engineering)



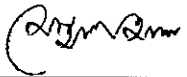
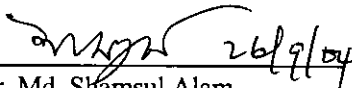
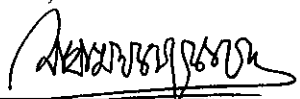

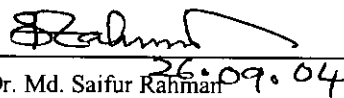
#99672#

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY  
DHAKA, BANGLADESH

SEPTEMBER 2004

The thesis "A Hybrid Admission Control Algorithm for Multimedia Server", submitted by Dewan Tanvir Ahmed, Roll No. 040205056P, Registration No. 95412, Session April 2002, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Master of Science and Engineering (Computer Science and Engineering) and approved as to its style and contents. The examination was held on September 26, 2004.

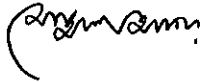
### Board of Examiners

1.   
Dr. Md. Mostofa Akbar  
Assistant Professor  
Department of CSE  
BUET, Dhaka-1000  
Chairman  
(Supervisor)
2.   
Dr. Md. Shamsul Alam  
Professor and Head  
Department of CSE  
BUET, Dhaka-1000  
Member  
(Ex-officio)
3.   
Dr. M. Kaykobad  
Professor  
Department of CSE  
BUET, Dhaka-1000  
Member
4.   
Dr. Md. Saidur Rahman  
Assistant Professor  
Department of CSE  
BUET, Dhaka-1000  
Member
5.   
Dr. Md. Saifur Rahman  
Professor  
Department of EEE  
BUET, Dhaka-1000  
Member  
(External)

## Declaration

I, hereby, declare that the work presented in this thesis is the outcome of the investigation performed by me under the supervision of Dr. Md. Mostofa Akbar, Assistant Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka. I also declare that no part of this thesis and thereof has been or is being submitted elsewhere for the award of any degree or Diploma.

Countersigned



---

(Dr. Md. Mostofa Akbar)

Sign



---

(Dewan Tanvir Ahmed)

## Acknowledgement

Here I would like to take the opportunity to express my greatest gratitude to the patrons of this thesis work, without whom I could never have completed this arduous task.

For me, it has been a big journey from the start to end. Needless to say, that the only thing that kept me going was the support of a number of people. First and foremost, **Dr Md. Mostofa Akbar**, Assistant Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, “Without your unstinting support, faith in my work, there is just no way that I could have completed my thesis. You have been always there whenever I needed your help in any form. I guess no words can adequately describe what you have done for me and for my work as an advisor, and companion. I thank you for everything.”

I would also like to express my heartiest gratitude to **Abu Wasif**, Assistant Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology for his fruitful suggestions whenever I needed.

I must acknowledge with due respect the constant support and patience of my parents for completing the thesis.

## Abstract

A multimedia server has to serve a large number of clients simultaneously. Given the real-time requirements of each client and fixed data transfer bandwidth of disks, a multimedia server must employ admission control algorithms to decide whether a new client can be admitted without violating the service requirements of the clients already being served. The main goal of admission control algorithm is to accept enough traffic to efficiently utilize server resources, while not accepting clients whose admission may lead to the violations of the service requirements of clients. In this thesis we are proposing a hybrid admission control algorithm that can handle a larger number of clients simultaneously. One interesting feature of this algorithm is different admission techniques for different clients based on their service requirements.

The performance of hybrid admission control algorithm is dependent on the disk-scheduling algorithm. Most of the conventional disk-scheduling algorithms have addressed this problem of optimizing total seek time and they completely ignore the rotational latency. In this thesis we demonstrate a new disk scheduling technique named near optimal disk scheduling algorithm that derives a sequence of accessing media blocks from disks so as to minimize both seek time and rotational latency incurred during retrieval. In order to provide continuous retrieval of each media stream, we have to ensure that service time is less than minimum duration of a round. Since the service time is a function of the number of blocks and their relative positions on the disk, it may exceed the minimum duration of a round. We refer to such rounds as overflow rounds. In hybrid admission control algorithm we can restrict overflow rounds within the limit by adjusting some parameters of the algorithm.

The near optimal disk scheduling algorithm as well as the technique for minimizing overflow of rounds presented in this thesis significantly improves the performance of the hybrid admission control algorithm. We have demonstrated the effectiveness of the hybrid admission control algorithm and the near optimal disk-scheduling algorithm through extensive simulation.

# CONTENTS

<b>Acknowledgement.....</b>	<b>i</b>
<b>Abstract.....</b>	<b>ii</b>
<b>List of Figures.....</b>	<b>v</b>
<b>List of Tables.....</b>	<b>vi</b>
<b>CHAPTER 1.....</b>	<b>1</b>
<b>INTRODUCTION.....</b>	<b>1</b>
1.1 MULTIMEDIA.....	1
1.2 MOTIVATION.....	3
1.3 BACKGROUND AND PRESENT STATE OF THE PROBLEM.....	6
1.4 OBJECTIVES AND SCOPE OF THE THESIS.....	7
1.5 ORGANIZATION OF THE THESIS.....	8
<b>CHAPTER 2.....</b>	<b>9</b>
<b>ADMISSION CONTROL ALGORITHMS: A LITERATURE SURVEY.....</b>	<b>9</b>
2.1 ADMISSION CONTROL ALGORITHM.....	9
2.1.1 <i>Deterministic Approach</i> .....	10
2.1.2 <i>Probabilistic Approach</i> .....	10
2.1.3 <i>Observation Approach</i> .....	11
2.2 STATISTICAL ADMISSION CONTROL ALGORITHM.....	11
2.3 OBSERVATION BASED ADMISSION CONTROL ALGORITHM.....	13
2.4 APPLICATION OF ADMISSION CONTROL ALGORITHMS.....	14
2.4.1 <i>Admission Control in Telephone Network</i> .....	14
2.4.2 <i>Resource Reservation in IP Data Networks</i> .....	15
2.4.3 <i>Admission Control in the Internet</i> .....	17
2.5 ADAPTIVE MULTIMEDIA SYSTEM (AMS).....	18
2.6 ADMISSION CONTROL AND QOS ADAPTATION METHODOLOGY FOR AMS.....	19
2.6 MULTIMEDIA SERVERS.....	20
2.7 SUMMARY.....	22
<b>CHAPTER 3.....</b>	<b>24</b>
<b>A NEW DISK SCHEDULING ALGORITHM.....</b>	<b>24</b>
3.1 INTRODUCTION.....	24
3.2 DISK HARDWARE.....	25

3.3 TRADITIONAL DISK SCHEDULING ALGORITHMS-----	26
3.3.1 <i>First Come First Serve</i> -----	27
3.3.2 <i>Shortest Seek Time First</i> -----	27
3.3.3 <i>Scan/Elevator</i> -----	28
3.4 A NEAR OPTIMAL DISK SCHEDULING ALGORITHM BY USING A HEURISTIC APPROACH ----	30
3.5 ANALYSIS OF DISK SCHEDULING ALGORITHMS-----	34
<b>CHAPTER 4-----</b>	<b>36</b>
<b>A HYBRID ADMISSION CONTROL ALGORITHM-----</b>	<b>36</b>
4.1 INTRODUCTION-----	36
4.2 FORMULATING ADMISSION CONTROL PROBLEM-----	36
4.3 HYBRID ADMISSION CONTROL ALGORITHM-----	39
4.4 ADMITTING A NEW CLIENT-----	42
4.4.1 <i>Deterministic Criteria</i> -----	42
4.4.2 <i>Non Deterministic Criteria</i> -----	43
4.5 ENFORCING SERVICE GUARANTEES-----	44
4.5.1 <i>Overflow Rounds</i> -----	44
4.5.2 <i>Policy of Discarding Blocks</i> -----	44
4.5.3 <i>Selection of Affordable Clients to Discard Blocks:</i> -----	45
4.5.4 <i>Selection of Discarding Blocks:</i> -----	46
4.6 ANALYSIS OF HYBRID ADMISSION CONTROL ALGORITHM-----	48
<b>CHAPTER 5-----</b>	<b>51</b>
<b>SIMULATION AND EXPERIMENTAL RESULTS-----</b>	<b>51</b>
5.1 INTRODUCTION-----	51
5.2 SIMULATION PARAMETERS-----	52
5.3 EXPERIMENTAL RESULTS OF NEAR OPTIMAL DISK SCHEDULING ALGORITHM-----	53
5.4 EXPERIMENTAL RESULTS OF HYBRID ADMISSION CONTROL ALGORITHM-----	55
5.5 CONCLUSION-----	61
<b>CHAPTER 6-----</b>	<b>62</b>
<b>CONCLUSION AND RECOMMENDATIONS-----</b>	<b>62</b>
6.1 CONCLUDING WORDS-----	62
6.2 RECOMMENDATIONS FOR FUTURE WORK-----	63
<b>APPENDIX A-----</b>	<b>66</b>
<b>BIBLIOGRAPHY-----</b>	<b>89</b>

## List of Figures

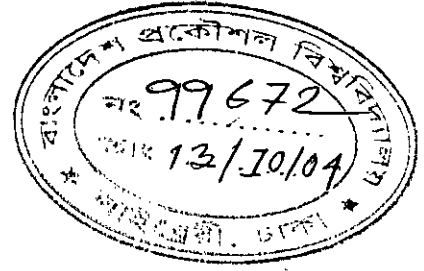
FIG 3.1 TYPICAL STRUCTURE OF A DISK -----	25
FIG 3.2 FIRST-COME, FIRST-SERVED (FCFS) DISK SCHEDULING ALGORITHM -----	27
FIG 3.3 SHORTEST SEEK TIME FIRST (SSTF) DISK SCHEDULING ALGORITHM. -----	28
FIG 3.4 SCAN/ELEVATOR DISK SCHEDULING ALGORITHM-----	29
FIG 3.5 GIVEN A SET OF POINTS, CONSIDERED AS VERTICES IN THE GRID.-----	32
FIG 3.6 MINIMUM SPANNING TREE T OF THESE POINTS. -----	32
FIG 3.7 A WALK OF T, STARTING AT 'A' -----	33
FIG 3.8 NEAR OPTIMAL TOUR -----	33
FIG 3.9 AN OPTIMAL TOUR-----	34
FIG 4.1 DURATION OF A ROUND -----	37
FIG 4.2 FLOW CHART OF HYBRID ADMISSION CONTROL ALGORITHM -----	42
FIG 4.3 DUAL PROCESSORS ARCHITECTURE -----	50
FIG 5.1 SERVICE TIME COMPARISON OF DISK SCHEDULING ALGORITHMS WHEN RETRIEVING 100 BLOCKS-----	54
FIG 5.2 COMPARISON OF SCAN AND NEAR OPTIMAL DISK SCHEDULING ALGORITHMS -----	54
FIG 5.3 TIME ANALYSIS OF VARIOUS DISK SCHEDULING ALGORITHMS -----	55
FIG 5.4 INFLUENCE OF SAFE GUARD ON THE NUMBER OF CLIENTS AND OVERFLOW ROUNDS -----	57
FIG 5.5 INFLUENCE OF EPSILON ON THE NUMBER OF CLIENTS AND OVERFLOW ROUNDS -----	58
FIG 5.6 EVALUATION OF HYBRID ADMISSION CONTROL ALGORITHM -----	59
FIG 5.7 INFLUENCE OF THE SEEK TIME ON CLIENTS AND OVERFLOW ROUNDS -----	59
FIG 5.8 INFLUENCE OF THE MAXIMUM ROTATIONAL LATENCY ON CLIENTS AND OVERFLOW ROUNDS -----	60



## List of Tables

<b>TABLE 5.1</b> DISK PARAMETERS ASSUMED IN THE SIMULATION -----	52
<b>TABLE 5.2</b> SIMULATION PARAMETERS OF ADMISSION CONTROL ALGORITHM -----	53
<b>TABLE 5.3</b> COMPARISON OF DIFFERENT ADMISSION CONTROL ALGORITHMS. -----	56
<b>TABLE 5.4</b> INFLUENCE OF SAFE GUARD ON NUMBER OF CLIENTS AND OVERFLOW ROUNDS FOR $\epsilon=0$ . -----	56
<b>TABLE 5.5</b> INFLUENCE OF SAFE GUARD ON NUMBER OF CLIENTS AND OVERFLOW ROUNDS FOR $\epsilon=1$ . -----	57

# Chapter 1



## Introduction

---

### 1.1 Multimedia

Multimedia means, from the user's perspective, that computer information can be represented through audio and/or video, in addition to text, image, graphics and animation. For example, using audio and video, a variety of dynamic situations in different areas, such as sport or ornithology lexicon, can often be presented well than just using text and image alone.

If we derive a multimedia system from the meaning of the words in the American Heritage Dictionary, then a multimedia system is any system, which supports more than a single kind of media. This characterization is insufficient because it only deals with a quantitative evaluation of the system. We understand a multimedia more in a qualitative rather than a quantitative way.

The integration of these media into the computer provides additional possibilities for the use of computational power currently available. Furthermore, these data can be transmitted through computer and telecommunication networks, which imply applications in the areas of information distribution and cooperative work. Multimedia provides the possibility for spectrum of new applications, many of which are in place today. On the other hand, one also has keep in mind the problems of global communication, with its social and legal implications.

A multimedia system is characterized by computer-controlled, integrated production, manipulation, presentation, storage and communication of independent information, which is encoded at least through a continuous (time-dependent) and a discrete (time-independent) media.

From the networking perspective, all media types can be classified as either Real-time (RT) or Non-real time (NRT) depending on their end-to-end delay requirements. For example, text and image files do not have any delay constraints and hence classified as NRT media types. The RT media types are further classified as Discrete Media (DM) or Continuous Media (CM), depending on whether the RT data is transmitted in discrete quantum (as a file or message) or continuously (as a stream of messages with inter-message dependency). The real time discrete type of media has recently gained high popularity because of ubiquitous applications like MSN/Yahoo messengers (which are error-intolerant) and instant messaging services like stock quotes (which are error tolerant). The RT continuous type of media can further be classified as delay tolerant or delay intolerant. The term 'delay intolerant' only signifies that such media type can tolerate higher amounts of delay than the delay tolerant types, without significant performance degradation. Examples of RT, continuous media delay intolerant media are audio/video media used in audio/video conferencing systems, and remote desktop applications. Streaming audio/video media used in applications like Internet web cast are also the examples of delay-intolerant media types.

Recently there has been an interesting growth in the demand for distributed multimedia applications operating over the Internet. Such applications have shown their value as a powerful technology that can allow people to remotely share resources or work collaboratively, thus saving time and money. Typical applications of distributed multimedia systems include video conferencing, video telephony, collaborative work, multimedia mail, and distance learning. Some recent applications include on-demand multimedia services, such as in entertainment, video news distribution services, and distribution of video rental services, interactive television, and digital multimedia libraries.

These applications demand certain constraints or service guarantees from the communication network, which must be satisfied to deliver an acceptable performance. The performance guarantee that a multimedia network can provide its applications is often referred to as *Quality of Service (QoS)*. This may include guarantees on the throughput, network delays, delay jitter, and error rate. The

current best-effort Internet, however, has been found to be inadequate to satisfy such requirements and enhancements are required to this basic Internet model to overcome this shortcoming.

## 1.2 Motivation

Most traditional applications (e.g. word processors, spreadsheets, graphical editors, file transfer, web servers) often execute, store and retrieve data aperiodically or asynchronously and are not required to have some CPU time or access to disks at regular intervals of time. Additionally, they are loss intolerant but are typically insensitive to variance in delay (jitter). As a result, general-purpose operating systems were geared to provide best-effort (throughput-oriented) service to this random-access, often I/O intense applications. In the context of operating systems, best-effort service means that there is no guarantee about the timing of execution or delivery of data to the applications, but simply that the underlying framework (particularly consisting of the process scheduler, disk scheduler, disk layout and recovery mechanism etc.) tries to do the best it can to meet the service requirements of the applications. While a considerable delay in meeting their requirements can cause severe degradation in performance, relatively minor variations in service do not visibly affect performance. Throughput and response times are the primary measures of performance for these applications.

With the recent advances in computer capabilities, compression technologies and broadband networking audio and video applications have become an integral part of our everyday computational life. It has become necessary for us to provide an integrated environment for the execution of these multimedia applications. It becomes especially relevant in content servers that can serve different kinds of data to various clients. Multimedia applications have quite different resource and performance constraints than do traditional applications [1]. They require time-constrained, fair execution environments and periodic access to disks. Execution and retrieval of data for these applications have deadlines by which the application must get all the resources (data, CPU time etc.) to render video or audio. Most

multimedia applications are soft real time applications, which mean that some loss is tolerable while delay and jitter can greatly reduce performance [2][3]. Although it is important that a certain amount of data be supplied to the application within a given period of time, it is not necessary that all the requests be satisfied in order to provide reasonable application performance. Omission of a few disk requests does not proportionally translate into degradation of quality. This is especially important in order to cater to the information-access needs of a large number of users.

With the explosive growth of the Internet, there has been a huge increase in the amount of content accessed from other computer systems (content providing servers). In order to meet their service requirements of local and remote applications efficiently, the operating system needs to be aware of many different kinds of service patterns that the applications might have. For example, an FTP application requires that data get transferred as fast as possible. So throughput would be used to measure the performance of such an application. Interactive applications require a portion of the CPU or fetch data every now and then but do not necessarily complete their task quickly. Continuous media applications, on the other hand, require a guaranteed rate of delivery of data from the disk when playing back files. Service patterns for these applications are different and the operating system needs to have this knowledge incorporated in it to efficiently serve requests from these applications.

Another bottleneck that can be experienced in multimedia applications is from the network. The network delay (propagation, serialization, switch, and packetization) play a significant role in the quality of service of the multimedia applications. The technological development for hard disk seems to be slower than networking components. The invention of fiber optic networks provides higher transfer rate (terabit per second instead of megabit per second). Although the latency of network can not be overcome fully, recent technologies provide higher transmission rate through the network, but it can not be ignored totally while we consider multimedia transmission over the network. In this thesis, we considered

I/O bandwidth of the disk only for the admission control algorithm. Consideration of the network bandwidth is out of scope of this thesis.

One of the most ubiquitous components of a computer system is the hard disk, which is used for storage and retrieval of programs and data (both application and system dependent) etc, as well as for essential operating system functions like virtual memory management and more. Therefore, one of major factors impacting application performance is how fast data can be stored and retrieved from the hard disk. More importantly, disk access is orders of magnitude slower than memory access, and it is a bottleneck to overcome in order to provide good application performance by retrieving data quickly and efficiently. In order to speed up disk accesses efficient algorithms sequence disk requests in a way that would minimize time spent in retrieving data. The gap in the operating speeds of hard disks and memory has only widened with increase in computing power and faster memory access, making the problem of optimizing disk access even more critical. Compounding this problem is the fact that the applications developed for present day systems have significantly different service requirements [1].

Since a media stream consists of a sequence of media quanta, such as video frames or audio samples, which convey meaning only when presented continuously in time, a multimedia server must ensure that recording and retrieval of media stream to and from disks proceed at their real-time rate. Whereas designing a dedicated, single-client multimedia server does not offer many design choices and is relatively straightforward, the design of a multimedia sever that is capable of serving multiple clients simultaneously poses interesting research challenges. This is because, given the maximum rate of data transfer from disks, a multimedia server can only serve a limited number of clients. Hence, before admitting a new client, a multimedia server must employ admission control algorithms to decide whether a new client can be admitted without violating the continuity requirements of any of the clients already being served. Development of such algorithms together with a new disk-scheduling algorithm is the subject matter of this thesis.

### 1.3 Background and Present State of the Problem

Recent advances in computing and communication technologies have made it feasible as well as economically viable to provide on-line access to a wide variety of information sources such as reference books, journals, newspapers, images, video clips, scientific data, etc, over high speed networks. The realization of such information management systems of the future, however, will require the development of high performance, scalable multimedia servers which can provide a wide range of services to a large number of clients [4]. The fundamental problem in developing such multimedia servers is that images, audio, video and other similar forms of data differ from numeric data and text in their characteristics, and hence requires totally different techniques for their organization and management.

A large-scale multimedia server has to serve a large number of clients simultaneously. Given the real-time requirements of each client, a multimedia server must employ admission control algorithms to decide whether a new client can be admitted for service without violating the requirements of the clients already being served. Two types of new services have been proposed in the literature to support real-time multimedia applications: guaranteed service [5] and predicted service [6]. In a guaranteed service model, client-specified a priori performance bounds are guaranteed to each connection regardless of the behaviors of other connections. In a predicted service model, a network dictated post facto delay bound and the service may be disrupted due to the network load fluctuation.

In [7], two types of guaranteed services are proposed: deterministic service and statistical service. In deterministic service, performance bounds are guaranteed for all packets on a connection even in the worst case. In statistical service, probabilistic performance bounds are guaranteed [8]. Although the quality of deterministic service is better, statistical service allows the network to achieve a higher utilization by exploiting statistical multiplexing. In the predictive service model the admission control criterion is defined using the measured characteristics of the current load on the server, rather than theoretical worst-case bounds. Most

of the existing work on admission control algorithms for multimedia servers has been focused on developing techniques for providing deterministic service guarantees to clients (i.e., playback requirements are strictly met for the entire service duration) [9][10][11][12][13][14]. However, since human perception is tolerant to brief distortions in audio and video, providing deterministic guarantees to each client is superfluous. Furthermore, the worst-case assumptions that characterize most of these techniques may needlessly constrain the number of clients that are served simultaneously, and hence, may lead to severe under-utilization of server resources. This is because, the average time spent in accessing a media block from disk, in practice, and is significantly smaller than the corresponding worst-case times. Hence, in order to improve the utilization of server resources, a multimedia server must employ an admission control algorithm, which exploits the statistical variation in the access times of media blocks from disk.

## 1.4 Objectives and Focus of the Thesis

Typically, disk-scheduling mechanisms for multimedia applications reduce disk access times by only trying to minimize movement to subsequent blocks after sequencing based on Earliest Deadline First. We propose to implement a disk-scheduling algorithm that minimizes not only seek time but also rotational latency. We compared the results with various well-known disk-scheduling algorithms and found that near optimal approach is better than its counter parts in terms of service time when retrieving a large number of blocks from the disk. We find that our approach results in improved performance for multimedia and non-multimedia applications. The main focus of this thesis is a hybrid admission control algorithm that will increase number of clients for multimedia server as well as provide guaranteed service for the clients who have made request for guaranteed service. The thesis will focus on the following objectives:

- Define a new admission control algorithm using the previously mentioned hybrid approach that will decide whether a new client can be admitted for service without violating the requirements of the clients already being served.



- Formulation and implementation a disk-scheduling algorithm that will consider both seek and rotational latency.
- Analysis of the disk-scheduling algorithm.
- Comparison of performance with purely deterministic and purely statistical admission control algorithm.

## 1.5 Organization of the Thesis

The thesis has been organized in different chapters, with each chapter discussing different aspects of the study. The areas covered by different chapters are briefly as follows:

- Chapter 2:** Provides a literature review of the different strategies that have been proposed in the context of admission control algorithm for multimedia server.
- Chapter 3:** A new disk scheduling algorithm considering both the seek time and rotational latency.
- Chapter 4:** A detail discussion and formulation of a new admission control algorithm for multimedia server.
- Chapter 5:** Simulation outcomes and analysis of new disk scheduling algorithm and admission control algorithm for multimedia server.
- Chapter 6:** Concluding remarks and suggestion for future research work.

# Chapter 2

## Admission Control Algorithms: A Literature Survey

---

### 2.1 Admission Control Algorithm

A multimedia server has to serve a large number of clients simultaneously. Given the real-time requirements of each client and fixed data transfer bandwidth of disks, a multimedia server must employ admission control algorithm to decide whether a new client can be admitted for service without violating the requirements of the clients already being served. Admission control is a mechanism that multimedia servers use to restrict service to limited number of clients while either having a mechanism to allow re-negotiation with session requesting clients or deny service to the other clients till such time that there is enough bandwidth available to serve them.

From the client's perspective, it is important that the server guarantee a certain rate of delivery for multimedia content before starting the transmission. Multimedia clients typically negotiate using what are called Quality of Service (QoS) parameters to obtain a certain amount of service in terms of periodicity of data delivery. These QoS parameters are commonly expressed as bit rate, block rate and frame rate [15]. The server uses these performance parameters supplied by the clients for admission control and subsequent service. The server processes a client request based on QoS parameters and decides whether or not the performance guarantees for the client request can be met. This is essential to ensure acceptable deterioration in performance perceived by clients already being served [6][8][16][17]. There are three major approaches to carrying out admission control.

### **2.1.1 Deterministic Approach**

The first approach is to provide deterministic guarantees to the clients [9], the strictest form of admission control, since it uses worst-case values for retrieving media blocks from the disk. The advantage of this approach is that all admitted clients receive all the blocks and no service agreements are violated. The obvious disadvantage is that it is overkill for soft real-time applications like continuous media applications. This approach admits far fewer clients than can be served by the disk resulting in the under-utilization of the disk bandwidth. This is because algorithms like SCAN, C-SCAN (widely used scheduling algorithms) tend to re-sequence requests so as to serve all of those which are in the direction of the arm movement, which results in seek and rotational latency times being far less than the worst case ones (maximum possible values for seek times and rotational latencies) (disk scheduling algorithms are explain in chapter 3). Additionally, owing to the sequential nature of access of multimedia files, a number of serially laid out blocks are cached by the disk controller and operating system for future service. When the actual disk requests arrive for those blocks, they are served from the cache and hardly incur any service time in comparison to a disk access.

### **2.1.2 Probabilistic Approach**

The second approach is probabilistic in nature [8]. This only provides a statistical guarantee that the deadlines for all the admitted clients will be met. This means that at least a fixed percentage of the blocks are retrieved for each client but not necessarily all of them. One of the biggest advantages of this approach is that it results in an increase in the number of clients that are admitted compared to the deterministic approach, since it is not important that all the blocks are retrieved for all the clients. The obvious problem with this approach is that applications that have very strict deadline and loss requirements cannot be accommodated. In the event that deadlines are violated, there must be a mechanism to determine the blocks that can be dropped and to distribute the violation of service guarantees among as many of the admitted clients as possible. Therefore, this approach works very closely with the disk-scheduling algorithm.

### 2.1.3 Observation Approach

The third approach is one based on observation [16]. In this approach, the times taken for retrieving various media blocks are recorded. When there is a request for admission, an extrapolation is made from current values for access times to obtain the time taken for the new client. This estimated time is used to either accept or decline a client's request for admission. Although this does not provide very strict service guarantees like the deterministic approach, it still provides a fair amount of improvement over the deterministic approach. This algorithm also works very closely with the disk-scheduling algorithm in order to spread the effects of deadline violations among as many clients as possible. As we are proposing hybrid admission control algorithm, in the following sections we will describe different admission control algorithms in detail.

## 2.2 Statistical Admission Control Algorithm

Consider a multimedia server that is servicing  $n$  clients, each retrieving a different media strand (say,  $S_1, S_2, \dots, S_n$ , respectively). Let service requirements of Client  $i$  be specified as percentage  $p_i$  of the total number of frames that must be retrieved on time. A multimedia server can serve these clients by proceeding in periodic rounds, retrieving a fixed number of frames for each client during each round. Let  $f_1, f_2, \dots, f_n$  denote the number of frames of strands  $S_1, S_2, \dots, S_n$ , respectively during each round. Then assuming that  $R_{pl}^i$  denotes the playback rate (expressed in terms of frames/sec) of Strand  $S_i$ , the duration of a round, defined as the minimum of the playback durations of the frames accessed during a round, is given by,

$$R = \min_{i \in \{1, n\}} \left( \frac{f_i}{R_{pl}^i} \right) \quad (2.1)$$

In such a scenario, ensuring continuous playback of each media strand requires that the total time spent in retrieving media blocks from the disk during each round (referred as service time  $\tau$ ) should not exceed  $R$ . That is:

$$\tau \leq R \quad (2.2)$$

The service time, however, is dependent on the number of media blocks being accessed as well as their relative placement on the disk. Since each media strand may be encoded using variable bit rate compression technique, the number of media blocks that contain  $f_i$  frames of  $S_i$  strands may vary from one round to another. This difference when coupled with the variation in the relative separation between blocks yields different service times across rounds. In fact, while serving a large number of clients, the service time may occasionally exceed the round duration (i.e.,  $\tau > R$ ). We refer to such rounds as overflow rounds. Given that each client may have requested a different quality of service (i.e., different values of  $p_i$ ), meeting all of their service requirements will require the server to delay the retrieval of or discard media blocks of some of the more tolerant clients during overflow rounds. Consequently, to ensure that the statistical quality of service requirements of clients are not violated, multimedia server must employ admission control algorithms that restrict the occurrence of such overflow rounds by limiting the number of clients admitted for service.

To precisely derive an admission control criterion that meets the above requirement, observe that for rounds in which  $\tau \leq R$ , none of the media blocks need to be discarded. Therefore, the total number of frames retrieved during such rounds is given by  $\sum_{i=1}^n f_i$ . During overflow rounds, however, since a few media blocks may have to be discarded or delayed to yield  $\tau \leq R$ , the total number of frames retrieve will be smaller than  $\sum_{i=1}^n f_i$ . Given  $p_i$ , denotes the percentage of frames of strand  $S_i$  that must be retrieve on time to satisfy the service requirements of Client  $i$ , the average number of frames that must be retrieve during each round is given by  $p_i * f_i$ . Hence assuming  $q$  denotes the overflow probability (i.e.,  $P(\tau > R) = q$ ), the service requirements of the clients will be satisfied if:

$$q * F_0 + (1 - q) \sum_{i=1}^n f_i \geq \sum_{i=1}^n p_i * f_i \quad (2.3)$$

Here,  $F_0$  denotes the number of frames that are guaranteed to be retrieved during a overflow round. The left hand side of the equation 2.3 represents the lower bound on the expected number of frames retrieved during a round and the right hand side

denotes the average number of frames that must be accessed during each round so as to meet the service requirements of all clients. Clearly, the effectiveness of this admission control criteria, measured in terms of the number of clients that can be admitted, is dependent on the values of  $q$  and  $F_0$  [8].

Statistical admission control algorithm improves the utilization of server resources by exploiting the variation in the access times of media blocks from the disk. The main goals of this admission control algorithm are as follows:

1. Allowing enough traffic to efficiently utilize server resources, while not accepting clients whose admission may lead to the violations of the service requirements of the clients.
2. Providing statistical service guarantees to each client.

## 2.3 Observation Based Admission Control Algorithm

In observation based admission control algorithm a client is admitted for service only if the predicted extrapolation from the status quo measurements of the storage server utilization indicates that the service requirements of all the clients can be met satisfactorily. An observation-based admission control algorithm is based on the assumption that the amount of time spent in serving each of the clients already being served will continue to exhibit the same behavior, even after a new client is added into the system. Therefore, a new client will be admitted for service only if the prediction from the status quo measurements of the server performance characteristics indicates that the service requirements of all the clients can be met satisfactorily. A multimedia server that employs such an observation-based approach is referred to as providing predictive service guarantees to clients (A similar technique was presented by Clark et al. [6][18] for optimizing the utilization of network resource).

Notice that the observation-based admission control algorithm offers fairly reliable service, but no absolute guarantees. The admission control decisions are based on the measured characteristics of the current load on the server, rather than

theoretical worst-case behavior. Hence, the key function of the admission control algorithm is to accept enough traffic to efficiently utilize the server resources, while not accepting clients whose admission may lead to the violation of the service requirements. The performance of the admission control algorithm, and hence, the number of clients admitted and served simultaneously are maximized by employing a disk scheduling algorithm that minimizes both seek and rotational latency incurred while accessing a sequence of media blocks from disk.

Finally, since the observation-based admission control algorithm provides fairly reliable service but no absolute guarantees, simultaneous serving of multiple clients may lead to occasional violation of the continuity requirements (i.e., media unit losses) of some of the clients. In order to enable a multimedia server to meet the requirements of as many clients as possible, observation based admission control algorithm proposed a technique for minimizing as well as distributing the media unit losses among multiple clients [16].

## 2.4 Application of Admission Control Algorithms

Applications of admission control algorithm in some fields are presented in this section.

### 2.4.1 Admission Control in Telephone Network

The telephone network, carries audio between a source (the caller terminal) and a destination (the called terminal), is the best example of a circuit switched network. When a user dials a number, the telephone switch finds a free end-to-end voice circuit to carry the audio transmission. If such a path is available then the call is set up, otherwise the user gets the busy tone. All users are guaranteed to receive uninterrupted service if a call is set up. Throughput maximization in a telephone network depends on the algorithm used for selecting particular paths for new circuits. Plotkin [19] presented online competitive routing and admission control strategies for both throughput maximization and congestion minimization models, motivated by competitive analysis [20]. In both models each link in the network is defined by a *length*  $c_e$ , which is defined exponentially with regard to the current

congestion of the link. Let there be a request for bandwidth between nodes  $s$  and  $t$ . In the throughput maximization model, the flow is routed if there exists a path  $P(s,t)$  such that the flow is profitable with respect to this length  $c_e$ . If it is not, the flow is rejected. In the congestion minimization model, a rejection edge  $rej$  is created and a flow is routed along the shortest path with respect to the length  $c_e$ . The flow is then considered rejected if it is routed along  $rej$ , or if the path selected has insufficient capacity. Otherwise, it is considered accepted.

Because of only one QoS level in telecommunication networks, there is no opportunity for downgrading or upgrading the QoS level by changing the allocated bandwidth or the already selected path. Therefore, there is no way to optimize operating conditions dynamically. This is true for the sessions with almost invariable length or for the permanent calls. When the session lengths are unknown, it is important to admit a set of sessions that give maximum throughput for the system. This would require future prediction of the length of the sessions. Statistical distributions can be used to estimate the duration of a particular call. Admission decision with a *risk factor* [20] is also a good technique for prediction.

#### **2.4.2 Resource Reservation in IP Data Networks**

Currently, the Internet is based on a best-effort datagram service model: this model does not require (and generally does not permit) resource reservation prior to data transmission. When a packet arrives at a router, and sufficient resources (such as time and buffer-space on the outgoing link) are available, the packet is forwarded to the next router. However, if the necessary resources are not available, the incoming packet may be delayed, or even dropped. It is therefore difficult to predict, let alone guarantee, the bandwidth or latency experienced by a stream of packets under best-effort datagram services. And since each packet of a session is forwarded through the network independently, packets may experience variable and unpredictable delays, and may arrive at the destination out of order. This service has many advantages, but it is unworkable for real-time multimedia applications requiring absolute standards of performance such as continuous bandwidth during a Video on Demand session with maximum delay and jitter constraints. Hence a best-effort datagram service model is not considered suitable



for the Internet2 [21], which propose to offer the end-to-end quality-of-service guarantees similar to that of the telephone network.

Class-based forwarding proposals such as DiffServ [22], where the packets of applications requiring guaranteed QoS are assigned higher priority classes than the best-effort traffic, are at best partial solutions: they provide superior service to the QoS-sensitive application in the *relative* sense, i.e., relative to the best-effort traffic, but they cannot guarantee *absolute* standards of QoS to applications, including telephony and interactive video communication, which require such standards.

RSVP [23] is a protocol used to reserve resources i.e., link bandwidth over the Internet. It requires reservation in each switch from the source to the destination, which clearly requires the determination of a fixed path on which all datagram of the flow are carried. This protocol works for real-time audio and video transmission, and in that sense could provide a basis for guaranteed QoS, but scalability is a problem.

MPLS [24] provides a mechanism for sending data independent of the IP routing tables in the routers. In this mechanism each packet is routed through a predefined path, which is determined before data transmission. A label is added to the packet, and this label is used for table look up in the router for forwarding packets to the next router with another label. The label-forwarding table is created at the time of fixing the path and it contains additional information such as *Class-of-Service* (CoS) values that can be used to prioritize packet forwarding. As MPLS is a lower layer protocol than IP and UDP, real-time multimedia transmission using IP or UDP over MPLS is considered plausible.

Gerla [25] proposed DiffServ architecture for the support of real-time traffic (e.g., video) with QoS constraints without requiring per flow processing in the core routers and which are thus scalable. Paths are computed by means of a QoS routing algorithm, Q-OSPF, and MPLS is used to handle explicit routing and class separation.

### **2.4.3 Admission Control in the Internet**

There has been considerable work on endpoint admission control in the Internet. Gibbens [26] demonstrates an algorithm embedded in the end system to block an IP- telephony call when the network load is high. Kelly [27] describes a framework for admission control for a packet-based network such as the Internet where decisions are taken by edge devices based on the results of probe packets, rather than by entities within the network. Techniques to estimate network state are also presented. Kelly [28] proposed an ECN (Explicit Congestion Notification) probe based admission control that is fast, scalable and robust. This protocol is viable for both partitioned and integrated networks. Breslau [29] presents a study of endpoint admission control, discussing some architectural issues of endpoint system. The modest performance degradation between traditional router-based admission control and endpoint admission control suggests that a real-time service based on endpoint probing may be viable.

Feedback control theory as developed in control engineering provides another approach to control the Internet data traffic. Imer [30] presented a control theoretic approach to designing Available Bit Rate (ABR) congestion control algorithms. Only ABR traffic such as email and web browsing is regulated. This traffic gets feedback from the end user and the feedback is used to control the admission rate of the service. This is extremely useful for volatile networks, where the number of sessions and the delay of metrics are not exactly known to the switches.

Greenberg [31] proposed an admission control strategy where book-ahead calls and instantaneous calls are admitted when the probability of interrupting a call in progress is less than a threshold. Calls are downgraded by bit dropping in real time data transmission, or by coarser video encoding to maximize the sharing of resources among the calls. Srikant [32] proposed the central limit theorem for the approximation of probability of service interruption.

Scheduling algorithms are used to do admission control in optical and wireless networks where the users of the network share bandwidth. Dasylva [33] presents a

polynomial algorithm to produce optimal schedules under certain conditions of traffic metrics. Shakkottai discusses the problem of real-time streams with deadlines over a shared channel using different scheduling algorithms.

Asynchronous Transfer Mode (ATM) is a packet switch technique where the data is divided into 53 bytes cells and multiplexed on time slotted channels [34]. ATM switches use Virtual Circuits (VCs) and all the packets of a call follow the same route. When a cell arrives at a switch, the switch determines an outgoing link looking at the VC number in the header of the ATM cell. Courcoubetis [35] designed an admission control algorithm for call acceptance that guarantees a bound of cell loss because of buffer overflow.

In the smart market scheme introduced by Varian [36], the actual price for each packet is determined based on the current state of network congestion. Users offer a bid for their packets. The packets whose bids are more than a threshold will be admitted and the rest are dropped or buffered. Singh [37] proposed a dynamic capacity-contracting model, which is implementable in the differentiated services architecture of the Internet.

## 2.5 Adaptive Multimedia System (AMS)

We define a multimedia service provider, which provides multimedia service to users with guaranteed QoS levels and supports QoS adaptation an Adaptive Multimedia System (AMS) [38]. Each user submits its session request together with a set of QoS levels such as Gold (colour video and CD quality audio), Silver (B/W Video with telephone quality audio), or Bronze (phone quality audio). Depending on the availability of the resources of the AMS such as CPU cycles, I/O bandwidth and memory, a session can be admitted to enjoy a particular QoS level, or rejected. There must be an engine working as an admission and QoS adaptation controller in the AMS. For brevity we call it an *Admission Controller*. It keeps track of all the allocated resources of the system. When a new user places a request for a multimedia service, or when a session leaves, the controller can dynamically adapt the QoS level of any running session to allocate some resource

or to re-allocate newly- released resources. Considering multiple QoS level makes the problem of admission control and QoS adaptation more complex. But this also gives the opportunity to upgrade to a higher level and earn more revenue when some resources are released from the system.

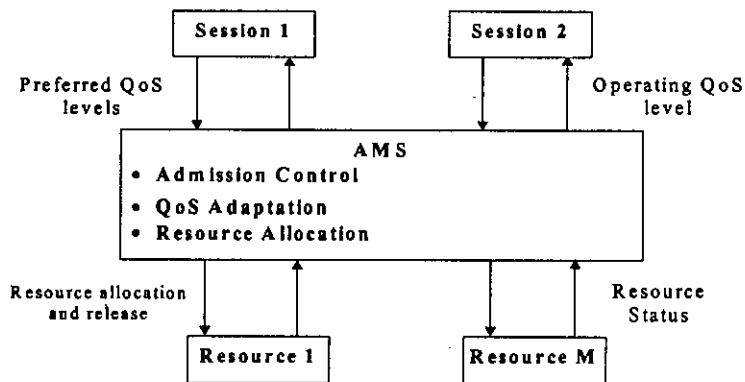


Fig 2.1 Adaptive Multimedia System

## 2.6 Admission Control and QoS Adaptation

### Methodology for AMS

There has been lots of interesting work in recent years on reservation-based management of resources, like CPU cycles and network bandwidth for multimedia service providers [39][40][41][42][43].

The *Benefit Model* for adaptation of quality attributes of a single user multimedia application has been proposed by Schreier and Davis [44]. The quality of the service is expressed by the video frame rate, audio/video quality and audio/video synchronization. Each of these quality parameters has an associated benefit function. The objective is to maximize the benefit of the service by adjusting the quality parameters. Chatterjee [45] proposed a *Logical Application Stream Model* (LASM) to capture the structure of a distributed multimedia application, with relevant resource requirement as well as end-to-end QoS parameters. Moser [46] presented an *Optimally Graceful QoS Degradation Model* (OGQD) where a single session's quality is gracefully degraded to meet resource constraints. For a

multimedia session the system calculates the set of services for maximum utility subject to resource constraints using a heuristic for solving the MMKP [47].

However, the Benefit Model, LASM and OGQD discuss the adaptation of QoS for a single multimedia session. They do not address the problem of adaptation in a multi session environment with a predefined objective like revenue or utility maximization.

Venkatasubramanian [48] proposed an economic framework for a multi-user multimedia service provider with different objectives for the users and for the service provider. In this framework a user's objective is to maximize QoS with respect to paid price but the service provider's objective is to maximize revenue with respect to resource usage of the system. This principle does not ensure the maximum utilization of resources of the service provider.

Lee [49] applied the concept of convex hull to solve the QoS management of a MRMD (Multiple Resource Multiple QoS Dimension) system. Each QoS of a multimedia session request is transformed to a point in a two dimensional space. A convex hull frontier is constructed with the points representing the QoS levels of each requested session. Admission or rejection, as well as QoS adaptation of a session, is based on the list of all the segments of the convex hull frontiers sorted in ascending order according to their slopes.

## 2.7 Multimedia Servers

Advances in storage technology have made it possible for a single commodity server to supply soft real-time multimedia streams to clients across a network. Such servers, however, exhibit poor scalability and availability [50]. One solution is to "clone" the servers, mirroring available data on each node. This approach increases the bandwidth capacity and availability of the service and is common in web server clusters, where the volume of data stored on the server is small. Cloned servers can be grouped to form a network load-balancing cluster and client requests are distributed among cluster nodes according to their capabilities and

current workload. However, the volume of data that is typically stored on a multimedia server usually prohibits this form of complete server replication. In addition, since clients rarely request the majority of multimedia files, replication of low-demand files is wasteful.

Server striping has been used in the past to share workload between multimedia servers. The concept is similar to RAID-0 [51] – multimedia files are divided into equal size blocks, which are distributed among server nodes in a predefined order [50]. Implicit load balancing across server nodes is achieved, while only storing a single copy of each file. The degree of node interdependence caused by server striping is high, however, because each server node is used in parallel to supply each individual multimedia stream. Node interdependence has several disadvantages. First, the process of stream reconstruction is expensive. Secondly, as nodes are added to a server, existing content must be redistributed. Many architectures also require that each sever node has an identical hardware configuration [52]. Finally, the failure of any single node will lead to the loss of all streams, unless redundant data is stored [53].

In a clustered multimedia server, by periodically evaluating client demand for each file and performing selective replication of those files with the highest demand, the files can be distributed among server nodes to achieve load balancing. This technique is referred to as dynamic replication [54].

GPFS (General Parallel File System) is a parallel file system for cluster computers that provides, as closely as possible, the behavior of a general-purpose POSIX (Portable Operating System Interface) file system running on a single machine. GPFS evolved from the Tiger Shark multimedia file system [55]. GPFS successfully satisfies the needs for throughput, storage capacity, and reliability of the largest and most demanding problems.

Traditional supercomputing applications, when run on a cluster, require parallel access from multiple nodes within a file shared across the cluster. Other applications, including scalable file and Web servers and large digital libraries, are

characterized by interfile parallel access. In the latter class of applications, data in individual files is not necessarily accessed in parallel, but since the files reside in common directories and allocate space on the same disks, file system data structures (metadata) are still accessed in parallel. GPFS supports fully parallel access both to file data and metadata. In truly large systems, even administrative actions such as adding or removing disks from a file system or rebalancing files across disks, involve a great amount of work. GPFS performs its administrative functions in parallel as well. GPFS achieves its extreme scalability through its shared-disk architecture [56]. A GPFS system consists of the cluster nodes, on which the GPFS file system and the applications that use it run, connected to the disks or disk subsystems over a switching fabric. All nodes in the cluster have equal access to all disks. Files are striped across all disks in the file system – several thousand disks in the largest GPFS installations. In addition to balancing load on the disks, striping achieves the full throughput of which the disk subsystem is capable.

C. Wu and R. Burns [57] developed a system for load management in shared-disk file systems built on clusters of heterogeneous computers. The system generalizes load balancing and server provisioning. It balances file metadata workload by moving file sets among cluster server nodes. It also responds to changing server resources that arise from failure and recovery and dynamically adding or removing servers. The system is adaptive and self-managing. It operates without any a-priori knowledge of workload properties or the capabilities of the servers.

## 2.8 Summary

The objective of admission control algorithm is to control the usage and allocation of disk resources for various applications requiring service. Admission control is a key component in multimedia servers, which need to allow the resources to be used by the clients only when they are available. This assumes great significance when the server needs to maintain a certain promised level of service for all the clients being served. If the admission control admits too few clients, it results in wastage of system resources. On the other hand, if too many clients are allowed to

contend for resources, then the performance of clients already degrades rapidly in the presence of new clients. Therefore, judicious decision making mechanisms for allocating resources disk bandwidth to clients are needed.



# Chapter 3

## A New Disk Scheduling Algorithm

---

### 3.1 Introduction

The file system is said to be the most sequential part of an operating system. Most programs write or read files. Their program codes, as well as user data, are stored in files. The organization of the file system is an important factor for the usability and convenience of the operating system. A file is a sequence of information held as a unit for storage and use in a computer system.

Files are stored in secondary storage, so they can be used by different applications. The life span of files is usually longer than the execution of a program. In traditional file systems, the information types stored in files are sources, objects, libraries and executable programs, numeric data, text, payroll records, etc. In multimedia systems, the stored information also covers digitized video and audio with their real time “read” and “write” demands. Therefore, additional requirements in the design and implementation of file systems must be considered.

The file system provides access and control functions for the storage and retrieval of files. From the user’s point of view, it is important how the file system allows file organization and structure. The internals, which are more important in our context, i.e., the organization of the file system, deal with the representation of information in files, their structure and organization in secondary storage. Thus disk scheduling is also important in this context.

The next section starts with a brief characterization of traditional file systems and disk scheduling algorithms. Subsequently, different approaches to organize

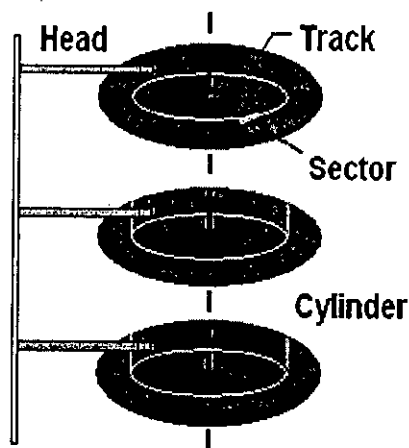
multimedia files and disk scheduling algorithms for multimedia systems are discussed. Finally, a new disk scheduling algorithm will be presented.

## 3.2 Disk Hardware

Nearly all computers have disks for storing information. Disks have three major advantages over using main memory for storage:

1. The storage capacity available is much larger.
2. The price per bit is much lower.
3. Information is not lost when the power is turned off.

All real disks are organized into cylinders, each one containing as many as tracks as there are heads stacked vertically (Fig 3.1). The tracks are divided into sectors, with the number of sectors around the circumference being 8 to 32. All sectors contain the same number of bytes, although a little thought will make it clear that sectors close to the outer rim of the disk will be physically longer than those close to the hub. The extra space is not used [58].



*Fig 3.1 Typical structure of a disk*

Consider the process of accessing one byte of information. A disk may be able to transfer information at a very large rate but before commencing this transfer, two processes are necessary. The head has first to be positioned over the required track. This is called seeking and the time it takes is the seek time. Although there

has been lots of improvement in the seek time, it is still large compared to the transfer rate. The disk unit must then wait for the required sector of that track to appear under the head. On the average half of the revolution of the surface will occur here. The time lag it creates is called latency. The combined time of seeking and latency is several orders of magnitude larger than the time to transfer one byte and dominates the actual transfer rate achievable. If more than one byte is transferred than the effective transfer rate is increased in proportion. That is, given that the dominant time involved is for seeking and latency, a unit of 512 bytes can be effectively read/write in the same time as the a unit of one byte.

This unit of transfer – one sector of one track – is often called a block although it is also often confusingly referred to as a sector. A large block size increases the effective efficiency of the surface. However, given that it is the minimum sized unit, space will be wasted when a smaller size is actually required. A file on the disk will occupy a whole number of blocks and the last block of a file, in general, be only partially used. On average, therefore, the last block will be half used. To summarize, a compromise is made when choosing a block size. A large size makes disk access more efficient, but a small size wastes less space. The equation is not quite simple.

### 3.3 Traditional Disk Scheduling Algorithms

The main goals of traditional disk scheduling algorithms are to reduce the cost of seek operations, to achieve a high throughput and to provide fair disk access for every process. The additional real-time requirements introduced by multimedia systems make traditional disk scheduling algorithms inconvenient for multimedia systems. Systems without any optimized disk layout for storage of continuous media depend far more on reliable and efficient disk scheduling than others. In the case of contiguous storage, scheduling is only needed to serve requests from multiple streams concurrently. In [59], a round-robin scheduler is employed that is able to serve hard real-time tasks. Here, additional optimization is provided through the close physical placement of streams that are likely to be accessed together. The time to read or write a disk block is determined by three factors: the

seek time, the rotational delay, and the actual transfer time. For most disks, the seek time dominates, so reducing the mean seek time can improve system performance substantially.

### 3.3.1 First Come First Serve

If the disk driver accepts requests one at a time and carries them out in that order, that is, First-Come, First-Served (FCFS), little can be done to optimize seek time. However, another strategy is possible when the disk is heavily loaded. It is likely that while the arm is seeking on behalf of one request, other processes may generate other disk requests. Many disk drivers maintain a table, indexed by cylinder number, with all the pending requests for each cylinder chained together in a linked list headed by the table entries. Fig. 3.2 depicts the policy of First-Come, First-Served (FCFS).

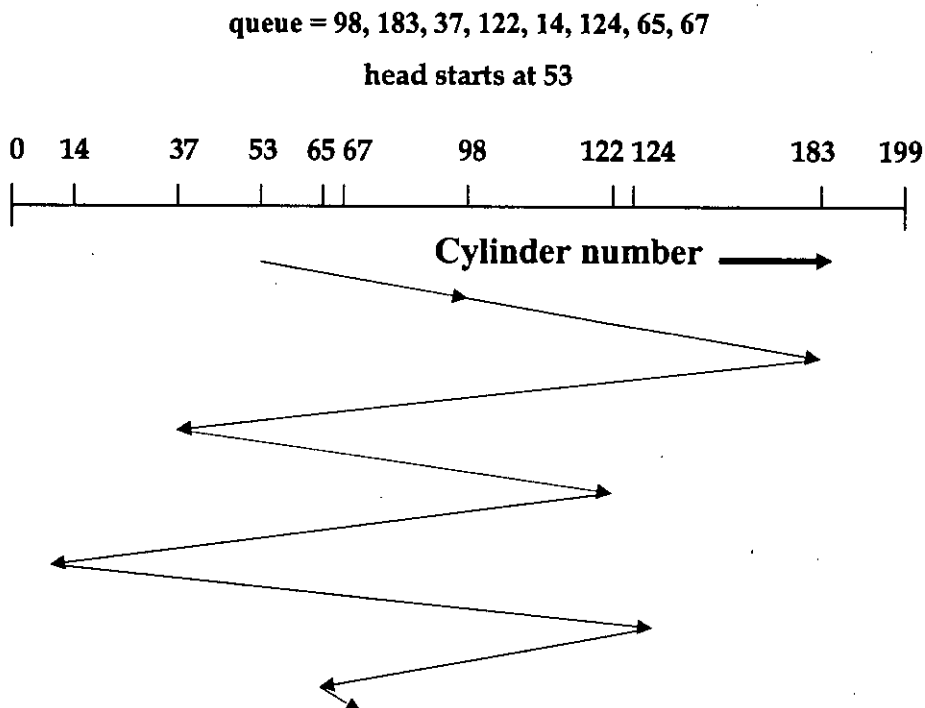
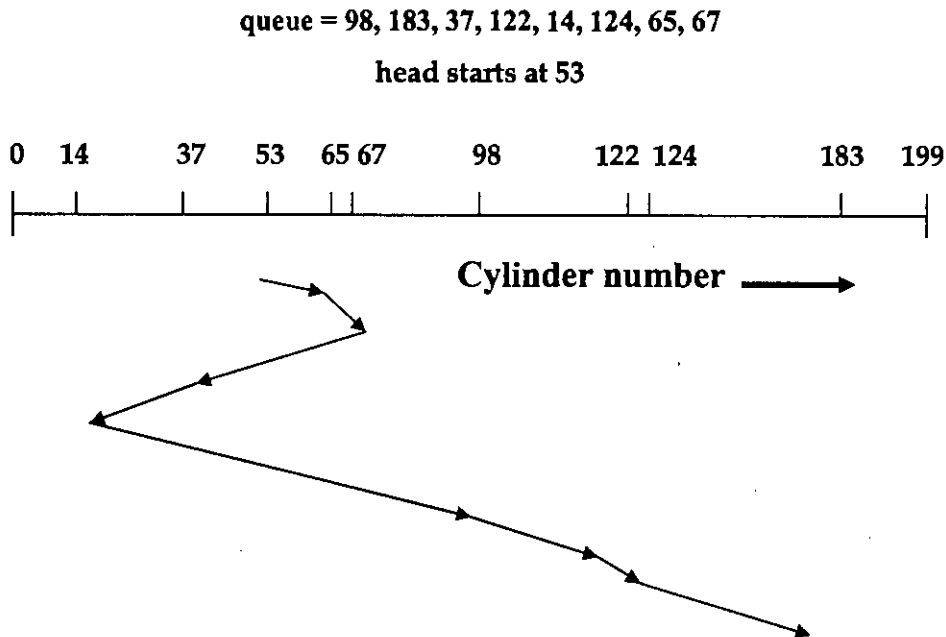


Fig 3.2 First-Come, First-Served (FCFS) disk scheduling algorithm

### 3.3.2 Shortest Seek Time First

Shortest seek time first always handles the closest request next, to minimize seek time. Given the requests of Fig. 3.3, the sequence is 65, 67, 37, 14, 98, 122, 124, and 183, as shown with jagged line in the Fig. 3.3.

Unfortunately, SSTF has a problem. With a heavily loaded disk, the arm will tend to stay in the middle of the disk most of the time, so requests at the either extreme will have to wait until a statistical fluctuation in the load causes there to be no requests near the middle. Requests far from the middle may get poor service. The goals of minimum response time and fairness are in conflict here.



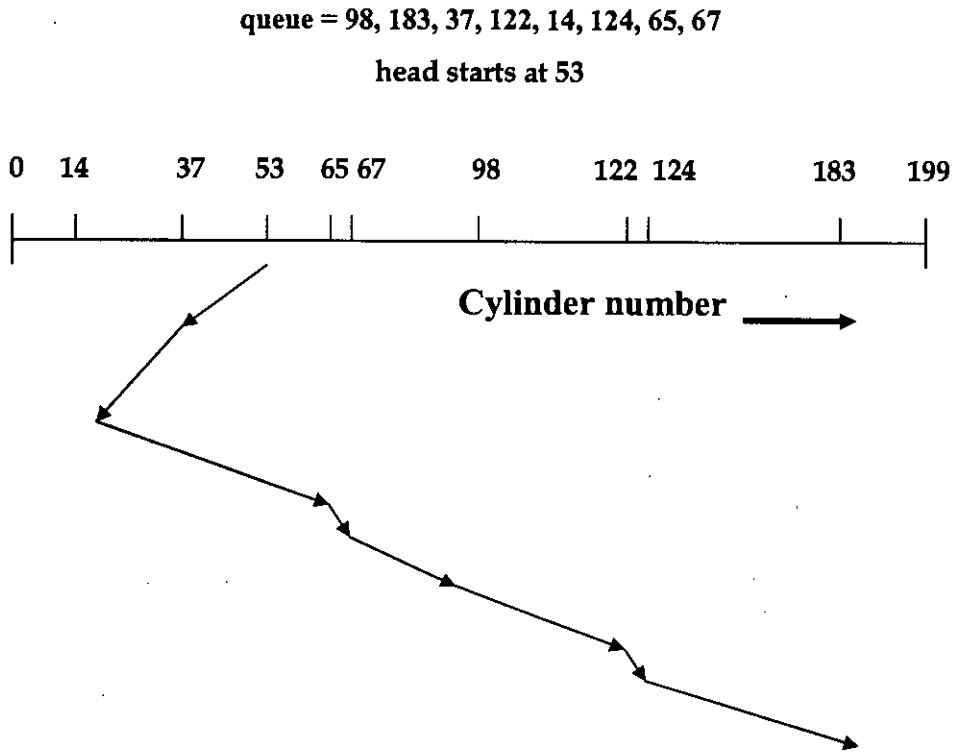
*Fig 3.3 Shortest seek time first (SSTF) disk scheduling algorithm.*

### 3.3.3 Scan/Elevator

The problem of scheduling an elevator in a tall building is similar to that of a disk arm. Requests come in continuously calling the elevator to floors (cylinder) at random. The microprocessor running the elevator algorithm could easily keep the track of the sequence in which customers pushed the call button, and service them using FCFS. It could also use SSTF.

However, most elevators use different algorithm to reconcile the conflicting goals of efficiency and fairness. They keep moving in the same direction until there are no more outstanding requests in that direction, and then they switch directions. This algorithm, known both in the disk world and the elevator world, as the elevator algorithm requires the software to maintain 1 bit: the current direction bit,

UP or DOWN. When a request finishes, the disk or elevator driver checks the bit. If it is UP, the arm is moved to the next highest pending request, if any. If no requests are pending at higher positions, the direction bit is reversed. When the bit is set DOWN, the move is to the next lowest requested position, if any.



*Fig 3.4 SCAN/Elevator disk scheduling algorithm*

Fig. 3.4 shows the elevator algorithm using the same requests as Fig. 3.2, assuming the direction bit was initially DOWN. The order in which the cylinders are served is 53, 37, 14, 65, 67, 98, 122, 124, and 183. In this case elevator is slightly better than SSTF with respect to fairness, although it is usually worse. One nice property that the elevator algorithm has is that given any collection of requests, the upper bound on the total motion is fixed: it is just twice the number of cylinders.

### 3.4 A Near Optimal Disk Scheduling Algorithm by using a Heuristic Approach

Consider a multimedia server that is expected to retrieve  $N$  blocks during each round. Let the position of each media block on disk be denoted as  $z_i = (x_i, y_i)$  where  $x_i$  and  $y_i$  denote the track number and the block number on that track, respectively. The goal of the disk-scheduling algorithm is to find the optimal access sequence for blocks  $z_1, z_2, \dots, z_N$ , assuming that the head is initially positioned at location  $z_0 = (x_0, y_0)$ . Formally, if  $f(z_i, z_j)$  denotes the cost function characterizing the overhead in positioning the disk head at location  $z_j$  starting from location  $z_i$ , then a sequence for retrieving  $N$  media blocks can be considered optimal if it minimizes the sum:

$$F = f(z_0, w_1) + f(w_1, w_2) + \dots + f(w_{N-1}, w_N) \quad (3.1)$$

Here sequence  $w_1, w_2, \dots, w_N$  represents sequence order of retrieval. Most of the disk scheduling algorithms consider only seek time but we are considering both seek and rotational latency incurred during retrieval of disk blocks. So our cost function be:

$$f(z_i, z_j) = l_{seek}(z_i, z_j) + l_{rot}(z_i, z_j) \quad (3.2)$$

Consider a fully connected directed graph  $G = (V, E)$ , where  $V = \{z_1, z_2, \dots, z_N\}$ . Let each edge  $(z_i, z_j)$ ,  $i \neq j$  in  $G$  be labeled with weight  $(z_i, z_j)$ . Notice that since the rotational latency incurred while moving the disk head from node  $z_i$  to  $z_j$ , in general, is not equal to that incurred while moving the disk head from  $z_j$  to  $z_i$ , the graph  $G$  contains both the edges  $(z_i, z_j)$  and  $(z_j, z_i)$ , each with a different weight.

Having constructed this graph, the problem of minimizing the total cost of retrieving media blocks during a round starting from node  $z_0$  can be reduced to the traveling salesman problem, a classical graph theory problem known to be NP-complete [60]. In the constructed graph,  $G$ , for all pair of vertices there are two edges between any two vertices with different weights. As the exact solution requires lots of time we are looking for a heuristic. By using triangle inequality method we approximate the order of blocks to be retrieved.

The following algorithm computes a near-optimal tour of Graph  $G$ , using the minimum spanning tree algorithm MST-PRIM. Graph  $G$  is constructed by using the vertices where each vertex is a block request. The cost function  $c$  is constructed from the weights among the blocks. Here cost function is considered in terms time. Minimum spanning tree is discovered by MST-PRIM algorithm. A preorder tree walk recursively visits every vertex in the tree, listing a vertex when it is first encountered, before any of its children are visited. The operations of near optimal disk scheduling algorithm are illustrated by using figures in the following paragraphs. Fig. 3.5 shows the given set of vertices where the ordinary Euclidean distance is used as the cost function between two points, and Fig. 3.6 shows the minimum spanning tree  $T$  grown from root  $a$  by MST-PRIM. A greedy method to obtain a minimum cost spanning tree would build this tree edge by edge. The next edge to include is chosen according to some optimization criterion. The simplest such criterion would be to choose an edge that results in a minimum increase in the sum of the costs of the edges so far included. In one approach, the set of edges so far selected form a tree. Thus, if  $A$  is the set of edges selected so far, then  $A$  forms a tree. The next edge  $(u, v)$  to be included in  $A$  is a minimum cost edge not in  $A$  with the property that  $A \cup \{(u, v)\}$  is also a tree. This selection criterion results in a minimum cost spanning tree. The corresponding algorithm is known as Prim's algorithm.

**Near\_Optimal\_Disk\_Scheduling\_Algorithm ( $G$ )**

[Say  $c$  is a cost function]

**begin**

1. Construct a graph by considering each block as a vertex and add edges among them by using cost function.
2. Select a vertex  $r \in V[G]$  to be a "root" vertex
3. Grow a minimum spanning tree  $T$  for  $G$  from  $r$  using MST-PRIM ( $G, c, r$ )  
Let  $L$  be the list of vertices visited in a preorder tree walk of  $T$ .
4. Return the Hamiltonian cycle  $H$  that visits the vertices in the order of  $L$  [it is order of blocks to be retrieved].

**end**



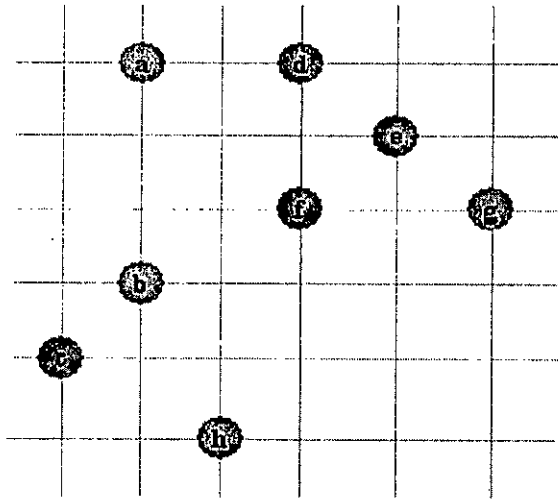


Fig 3.5 Given a set of points, considered as vertices in the grid.

The vertex 'a' is the root vertex. A full walk of the tree visits the vertices in the order  $a, b, c, b, h, b, a, d, e, f, e, g, e, d$ , and  $a$  (Fig. 3.7). A preorder walk of  $T$  lists a vertex just when it is first encountered, yielding the ordering  $a, b, c, h, d, e, f, g$  (Fig 3.8).

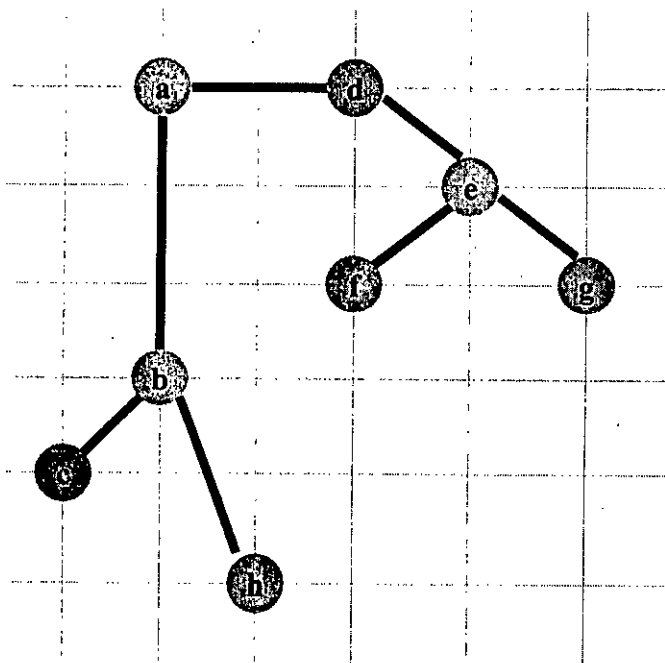


Fig 3.6 Minimum spanning tree  $T$  of these points.

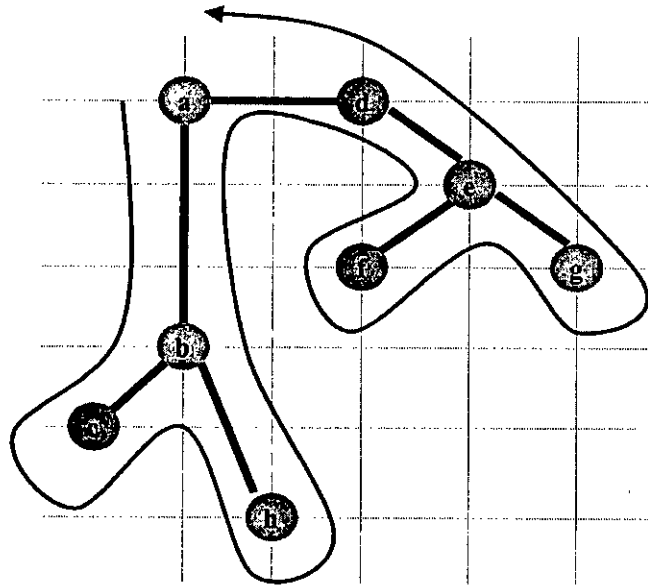


Fig 3.7 A walk of T, starting at 'a'

This is the order  $H$  returned by near optimal disk scheduling algorithm. Its total cost is approximately 19.074. An optimal tour  $H^*$  (Fig 3.9) costs approximately 14.715 (By observation).

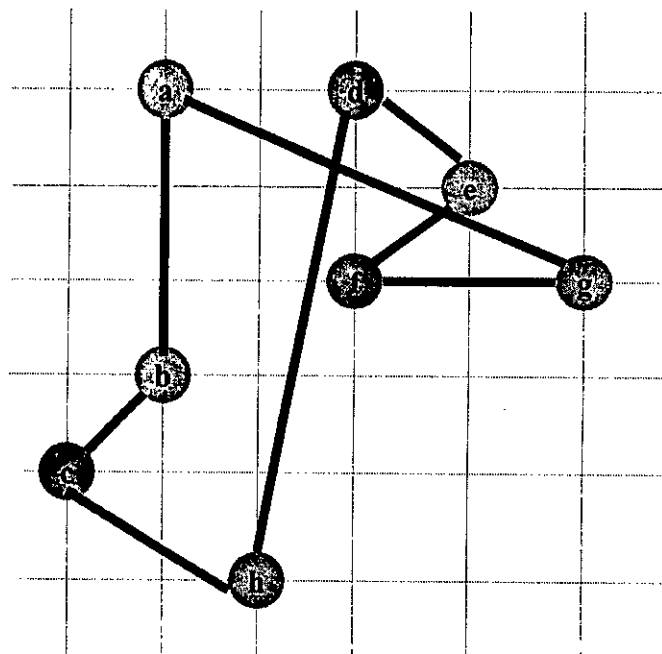


Fig 3.8 Near Optimal tour

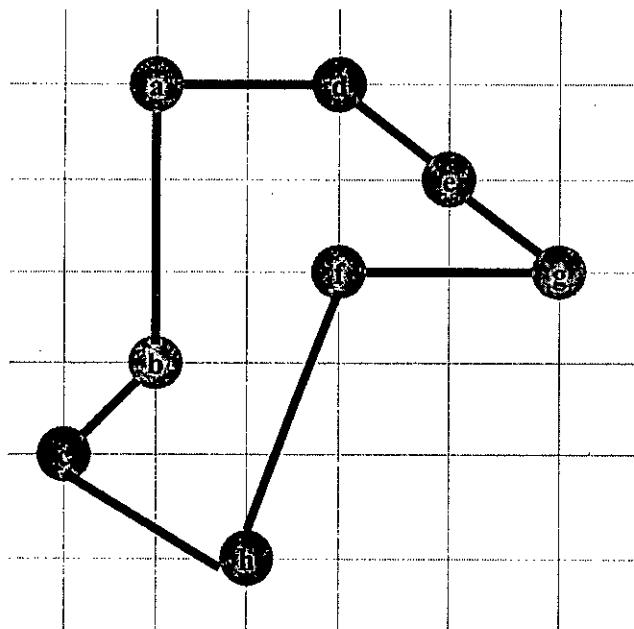


Fig 3.9 An optimal tour

In this case optimal tour is 23% shorter. The Near-Optimal-Disk-Scheduling algorithm is an approximation algorithm with a ratio bound of 2 with triangular inequality [60].

### 3.5 Analysis of Disk Scheduling Algorithms

The computational complexity of different disk scheduling algorithms is analyzed in this section.

- First Come First Server (FCFS) does its operation in constant time, as there is nothing to be done for preprocessing.
- The computational time of both Shortest Seek Time First (SSTF) and SCAN is  $O(n \lg n)$ , where  $n$  is the number of blocks in the queue to be retrieved.
- But the computational complexity of near optimal disk scheduling algorithm is  $\theta(n^2)$ , as this method uses a complete graph and construct minimum spanning tree by using MST\_PRIM algorithm (section 3.4).

The computational complexity of near optimal disk scheduling is higher than the other disk scheduling algorithms, but the order of blocks to be retrieved it

generates makes it lucrative in terms of effectiveness, i.e. lower service time. As this algorithm has a higher complexity, we can do the processing in advance by a parallel processor. Thus one processor is involved in generating the blocks order to be retrieved and other would satisfy clients demand. Threading would be another option to do the processing.

# Chapter 4

## A Hybrid Admission Control Algorithm

---

### 4.1 Introduction

Digitization of audio yields a sequence of samples, and that of video yields a sequence of frames. A continuously recorded sequence of audio samples or video frames is called a strand. A multimedia server must organize the storage of such media strands on disk (in terms of media blocks). Due to the periodic nature of media playback, a multimedia server can serve multiple clients simultaneously by proceeding in rounds. During each round, the multimedia server retrieves a sequence of media blocks for each strand. The number of blocks of each media strand retrieved during a round is dependent on its playback rate requirement, as well as the available buffer space at the client. Consequently, ensuring continuous retrieval of each strand requires that the service time (i.e., the total time spent in retrieving media blocks during a round) does not exceed the minimum of the playback durations of the sequences of blocks for each strand retrieved during the round. Since service time is a function of the number of blocks retrieved during a round, a server can serve only a limited number of clients simultaneously. Hence, before admitting a new client, a multimedia server must employ admission control algorithms to decide whether a new client can be admitted without violating the continuity requirements of any of the clients already being served [8].

### 4.2 Formulating Admission Control Problem

Consider a multimedia server that is serving  $n$  clients, each retrieving a different media strand (let,  $S_1, S_2, \dots, S_n$ , respectively). We refer to a continuously recorded sequence of audio samples or video frames as a strand. Let  $R_1, R_2, \dots, R_n$ , denote the playback rates (in terms of bytes/sec) of strands  $S_1, S_2, \dots, S_n$ , respectively.

Furthermore, let  $k_1, k_2, \dots, k_n$ , denote the number of blocks of strands  $S_1, S_2, \dots, S_n$ , retrieved during each round. The total service time in serving  $n$  requests during a round is dependent on the total seek time and rotational latencies incurred while accessing  $(k_1 + k_2 + \dots + k_n)$  media blocks from the disk. In the worst case, all the media blocks may be stored on separate tracks/cylinders; the disk head may have to be repositioned onto new track at most  $(k_1 + k_2 + \dots + k_n)$  times during each round.

Suppose,

$T$  = Number of tracks/cylinders on disk

$M$  = Disk block size, in bytes

$a, b$  = Seek time parameters, they are constants

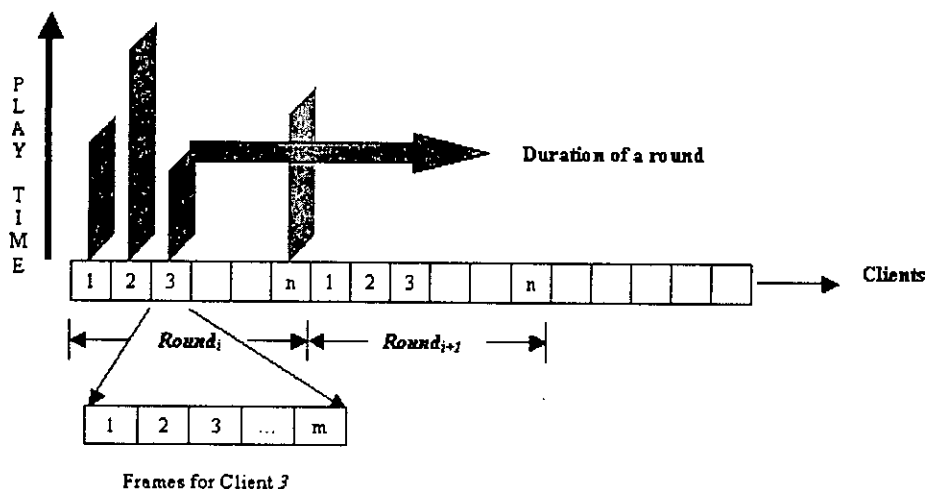
$l_{seek}(t_1, t_2)$  = Seek time  $(a + b * |t_1 - t_2|)$ , in sec

$l_{rot}^{max}$  = Maximum rotational latency, in sec

$R$  = Minimum playback rate

$\tau$  = Service time

It is quite clear in fig.4.1 how to determine the minimum duration of a round. Say, multimedia server is serving  $n$  clients. Server provides a number frames to each client and client consumes these frames. In order to ensure continuous playback condition server must provide a new set of frames before the client consumes earlier set of frames.



*Fig 4.1 Duration of a Round*

Duration of a round is the minimum time to consume a set of frames among the all clients. To ensure continuous playback of a media stream for each client, multimedia server must serve new set of frames before the minimum duration of a round. The equation  $\tau \leq R$  ensures that the playback requirements of all the clients are strictly met for the entire service duration.

So, total seek time incurred during a round be,  $a * \sum_{i=1}^n k_i + b * T$ . Again, maximum rotational latency incurred during a round is,  $I_{rot}^{max} * \sum_{i=1}^n k_i$ . Hence total service time for each round is bounded by:

$$\begin{aligned} \tau &= a * \sum_{i=1}^n k_i + b * T + I_{rot}^{max} * \sum_{i=1}^n k_i \\ &= (a + I_{rot}^{max}) * \sum_{i=1}^n k_i + b * T \end{aligned} \quad (4.1)$$

Consequently, ensuring continuous retrieval of each strand requires that the total service time per round do not exceed the minimum of the playback durations of  $k_1, k_2, \dots, k_n$ , blocks. So the admission control criteria can be formally stated as:

$$\tau = (a + I_{rot}^{max}) * \sum_{i=1}^n k_i + b * T \leq \min_{i \in \{1, n\}} \left( \frac{k_i * M}{R_i} \right) \quad (4.2)$$

Equation 4.2 indicates that service time should be less than or equal to the minimum duration of a round and ensures that the playback requirements of all the clients are strictly met for the entire service duration. Hence, a multimedia server that employs such an admission control criteria is said to provide *deterministic service guarantees* to each client.

The worst-case assumptions that characterize most of the deterministic techniques may needlessly constrain the number of clients that are serviced simultaneously, and hence, may lead to severe under-utilization of server resources. This is because, the average time spent in accessing a media block from disk, in practice, and is significantly smaller than the corresponding worst-case times. Hence, in order to improve the utilization of server resources, a multimedia server must employ an admission control algorithm, which exploits the statistical variation in the access times of media blocks from disk. Our hybrid admission control

algorithm is a combination of deterministic and observation based admission control technique.

### 4.3 Hybrid Admission Control Algorithm

Consider a multimedia server that is serving  $n$  clients, each retrieving a media strand (say  $S_1, S_2, \dots, S_n$ , respectively). Let  $n_s$  and  $n_n$  denote the number of clients that require deterministic and non-deterministic service, we call them super user and normal user respectively (i.e.,  $n = n_s + n_n$ ). Without loss of generality, let us assume that clients retrieving strands  $S_1, S_2, \dots, S_{n_s}$  require deterministic service guarantees, and those retrieving  $S_{n_s+1}, S_{n_s+2}, \dots, S_n$  are tolerant to brief distortions or loss of information.

Let, minimum duration of a round,  $R = \min_{i \in \{1, n\}} \left( \frac{k_i * M}{R_i} \right)$  sec. General rule of admission control is  $\tau \leq R$ . We are defining a new term *safe guard*; it is the reduced percentage of minimum duration of a round. Say, if the admission control technique uses 90% of  $R$  in the admission control equation we called it 10% safe guard. On the other hand, the total time spent in retrieving the media blocks from disk during each round (referred to as service time  $\tau$ ) is dependent on their relative placement on disk as well as the disk-scheduling algorithm. Since the relative placement of blocks can vary from one round to another, the service times may also vary across rounds. Consequently, in some rounds,  $\tau$  could be greater than  $R$ . Such rounds are called *overflow rounds*. Since maintaining continuity of playback for intolerant clients requires the service time to be smaller than the duration of a round, blocks of some of the tolerant clients may have to be discarded (i.e., not retrieved) during such overflow rounds.

Let,  $K$  denotes number of media blocks accessed during a round. So,  $K \leq \sum_{i=1}^n k_i$ .

Hence, average time of retrieving a media block is:

$$\eta = \frac{\tau}{K} \quad (4.3)$$



The admission control algorithm that we are presenting is based on the assumption that the average amount of time spent for the retrieval of each media block (i.e., the value of  $\eta$ ) does not change significantly even after the server admits a new client. In fact, to enable the multimedia server to accurately predict the amount of time expected to be spent while retrieving media blocks during a future round, we will maintain a history of the values of  $\eta$  observed during the most recent  $W$  rounds. If  $\eta_{avg}$  and  $\sigma$ , respectively, denote the average and the standard deviation of  $\eta$  over the last  $W$  rounds, then the time required to retrieve a block in future rounds can be estimated as:

$$\hat{\eta} = \eta_{avg} + \epsilon * \sigma \quad (4.4)$$

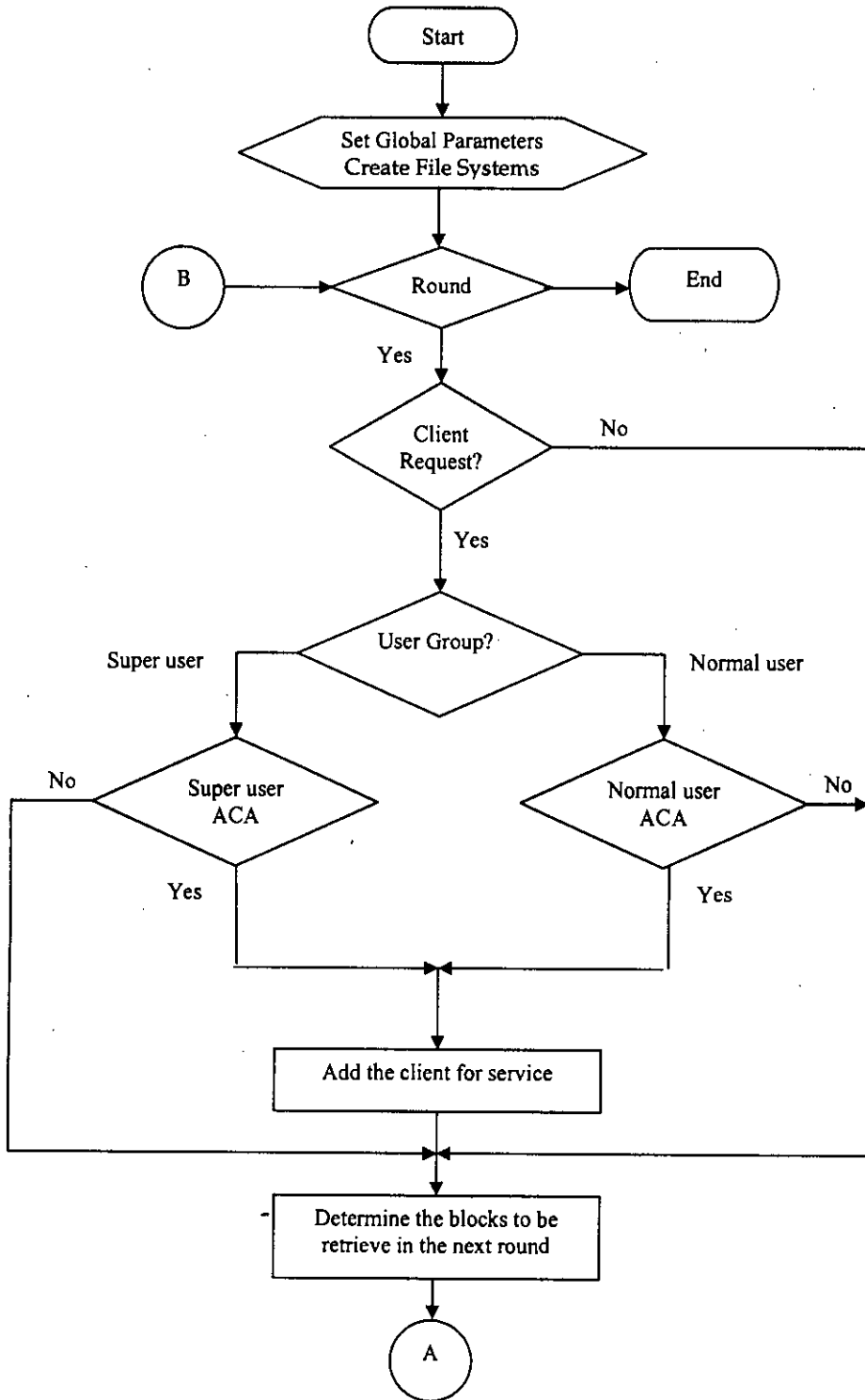
Here  $\epsilon$  is a constant. We will show, how this constant play an important role in admission policy in the next chapter.

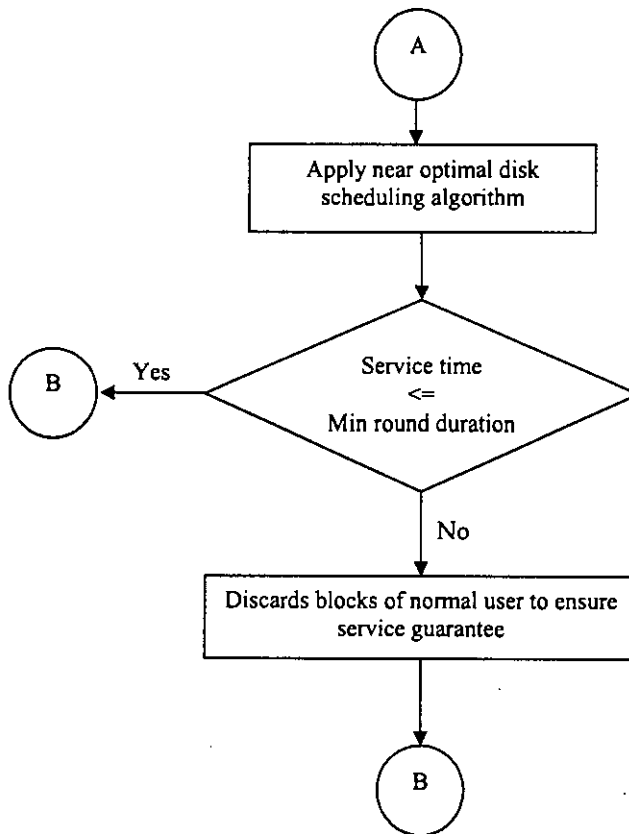
In hybrid admission control technique, we form two groups according to the service requirement. The clients who demand guaranteed service is fall in super user group and other clients are in normal user group. We apply different admission polices based on the service requirement of the clients. Say

$\tau_s$  = Service time of the super user group

$\tau_n$  = Service time of the normal user group

When a new client requests for accessing the resources of a multimedia server, according to the group we may need either to calculate the new service time  $\tau_s$ , or to estimate the new service time  $\tau_n$ . If the summation of these two service times is less than or equal to the minimum duration of a round, request is accepted otherwise it is declined. A flow chart of the hybrid admission control algorithm is given below. In the following paragraphs, hybrid admission policy explained in detail.





*Fig 4.2 Flow chart of hybrid admission control algorithm*

## 4.4 Admitting a New Client

In order to precisely formulate the admission control criteria, consider a scenario in which a multimedia server receives a new client request for the retrieval of strand  $S_{n+1}$ . Let,  $R_{n+1}$  denotes the playback rate requirement for the new client, and  $k_{n+1}$  denotes the number of blocks of strand  $S_{n+1}$  that need to be retrieved during each round.

### 4.4.1 Deterministic Criteria

If the new client desires deterministic service guarantees, then before admitting the client, the multimedia server must ensure that neither super user nor normal users are being suffered after the new client is admitted. The admission control

criterion has the following structure. Here by considering the worst-case assumption, we are trying to satisfy the following admission control criterion in order to admit a new client for getting multimedia service.

Say,  $n_s$  = Number of super users

$n_n$  = Number of normal users

$p_i$  = Percentage of blocks that must be retrieved on time for Client  $i$ .

New service time of super users,  $\tau_s^{new} = (a + l_{rot}^{max}) * \sum_{i=1}^{n_s+1} k_i + b * T$

Service time of normal users,  $\tau_n = \hat{\eta} * \sum_{i=1}^{n_n} k_i * p_i$

Thus admission control criterion be,

$$\tau_s^{new} + \tau_n \leq \min_{i \in [1, n+1]} \left( \frac{k_i * M}{R_i} \right) \quad (4.5)$$

#### 4.4.2 Non Deterministic Criteria

Consider, on the other hand, the scenario in which the new client requires non-deterministic service. Let  $0 < P_{n+1} \leq 1$  denotes the percentage of media blocks of strand  $S_{n+1}$  that must be retrieved on time from the multimedia server so as to satisfy the service requirements of the new client. Since admitting a normal client cannot violate the requirements of super users (since in the case of an overflow, media blocks of only normal users are discarded), the multimedia server can admit the new client if its admission will not violate the service being provided to any client. Hence, the multimedia server will admit the new client requiring this type of service only if equation 4.6 is satisfied.

Service time of super users,  $\tau_s = (a + l_{rot}^{max}) * \sum_{i=1}^{n_s} k_i + b * T$

New service time of normal users,  $\tau_n^{new} = \hat{\eta} * \sum_{i=1}^{n_n+1} k_i * p_i$

Thus admission control criterion be,

$$\tau_s + \tau_n^{new} \leq \min_{i \in [1, n+1]} \left( \frac{k_i * M}{R_i} \right) \quad (4.6)$$

The admission control algorithm, presented for the normal user, will increase the number of clients for multimedia server that can be served simultaneously. The performance of such an approach is dependent on the following parameters.

1. The values of  $\eta_{avg}$  and  $\sigma$ : smaller the values of  $\eta_{avg}$  and  $\sigma$ , greater is the number of clients that can be served simultaneously by the server. Hence, the multimedia server must employ disk-scheduling algorithms that minimize the total time spent in retrieving media blocks during each round.
2. The ability of the hybrid algorithm to meet the requirements of as many clients as possible. This requires that the multimedia server employ policies for determining the minimum number of media blocks, which when discarded will yield sufficient reduction in service time so as to ensure that service time is less than or equal to the duration of a round (i.e.  $\tau \leq R$ ).

## 4.5 Enforcing Service Guarantees

In this section we describe some policies to ensure service guarantee for the multimedia clients. This technique of ensuring service guarantees was first used by A. Goyal [8][18].

### 4.5.1 Overflow Rounds

Recall that the admission control algorithm presented in Section 4.4 admits a new client if the client is able to meet the admission control criteria in which it belongs. Due to the aggressive nature of this admission control criteria in case of normal user, the total time spent in retrieving media blocks during a round (i.e., service time  $\tau$ ) may occasionally exceed the duration of a round  $R$ , yielding an overflow.

### 4.5.2 Policy of Discarding Blocks

Observe, however, that since the retrieval sequence for Round  $i$  is pre-computed during Round  $(i-1)$  an overflow can be detected before actually initiating

Round  $i$ . Hence, in order to ensure that the deterministic service guarantees provided to clients are not violated, a multimedia server must discard (i.e., not retrieve) sufficient number of media blocks of normal clients so as to maintain the service time within the duration of the round (i.e.,  $\tau \leq R$ ). However, in doing so, the multimedia server must minimize the number of blocks discarded, as well as distribute the set of discarded blocks among the normal clients so as not to violate any of their requirements. In what follows, we present a technique for addressing this problem [8].

### 4.5.3 Selection of Affordable Clients to Discard Blocks

Consider a multimedia server that is serving  $n$  clients simultaneously. Let  $\forall_i \in [1, n]: P_i$  denotes the percentage of media blocks of strand  $S_i$  that must be retrieved on time from the multimedia server so as to meet the service requirements of the clients. Assuming that  $n_i$  clients require deterministic service guarantees, we get  $\forall_i \in [1, n]: P_i = 1$ .

Let,

$I$  = Average length of service of normal clients, in terms of time unit,  
 $R$  = Duration of a round,

Thus,

$$r = \text{Number of rounds in this service interval,} \\ = \frac{I}{R} \text{ rounds.}$$

Suppose, for each Client  $i$ ,

$k_i$  = Number of blocks of strand  $S_i$  to be retrieved during each round

$l_i$  = Number of media blocks of strand  $S_i$  that have already been discarded.

$\chi_i$  = Loss affordability

Say,  $L_i$  is the number of media blocks of strand  $S_i$  which can be discarded without violating the service requirements of Client  $i$ . So,  $L_i$  is bounded by:

$$L_i = \lfloor (1 - P_i) * k_i * r \rfloor \quad (4.7)$$

Consider, now, an overflow round  $j$ ,  $0 \leq j \leq r$ . Thus, the number of media blocks of strand  $S_i$  that can be discarded during Round  $j$  is bounded by  $(L_i - l_i)$ , based on which we define the loss affordability for Client  $i$  as:

$$\chi_i = \frac{L_i - l_i}{L_i} = 1 - \frac{l_i}{L_i} \quad (4.8)$$

Clearly, as the number of blocks of strand  $S_i$  discarded during an interval  $I$  increase, the loss affordability of Client  $i$  decreases. In the limit, if  $l_i = L_i$  then  $\chi_i = 0$ . That is, if the number of media blocks of strand  $S_i$  discarded during interval  $I$  have already reached its upper limit; no more blocks of  $S_i$  should be discarded during Round  $j$ . A multimedia server will discard blocks of only those clients with  $\chi_i > 0$ .

#### **4.5.4 Selection of Discarding Blocks**

Given the set of clients with  $\chi_i > 0$ , a multimedia server may employ various techniques to select the precise set of media blocks, which can be discarded.

##### **4.5.4.1 Technique Based simply on the Loss Affordability**

In the simplest case, media blocks of a strand with the highest loss affordability can be discarded until no more blocks of that strand can be discarded during the round. If the resulting round continues to yield an overflow (i.e.,  $\tau > R$ ), then discard blocks of the strand with the second largest loss affordability, and so on. A multimedia server can address this problem by distributing the number of media blocks that need to be discarded during Round  $j$  among all the clients with  $\chi_i > 0$ .

##### **4.5.4.2 Effectiveness**

Once a media block to be discarded is determined, the multimedia server can recompute the service time  $\tau_{new}$  (by determining a new retrieval sequence using the disk scheduling algorithm). If  $\tau_{new} > R$ , then the number of media blocks discarded can be progressively increased until the new retrieval sequence yields  $\tau_{new} \leq R$ . However, the high computational complexity of the disk-

scheduling algorithm renders this straightforward approach prohibitively expensive to implement.

To avert the re-computations, a multimedia server may just approximate the reduction in service time yielded by discarding a media block located at  $w_i$  as  $f(w_{i-1}, w_i) + f(w_i, w_{i+1}) - f(w_{i-1}, w_{i+1})$ , where  $w_{i-1}$  and  $w_{i+1}$  denote the predecessor and successor nodes of  $w_i$  in the original retrieval sequence. Conceptually, this is equivalent to replacing edges  $(w_{i-1}, w_i)$  and  $(w_i, w_{i+1})$  from the graph by edge  $(w_{i-1}, w_{i+1})$

Whereas such an approximation technique significantly reduces the computational complexity of the algorithm, it is not suitable for either one of the schemes (namely, discarding as much for one client before switching to the next client, or distributing the discarded blocks among all the clients with non-zero loss affordability). This is because, both of these schemes determine the set of blocks to be discarded based on the loss affordability of the strands, and not on the relative placement of the chosen blocks on disk. Since the near optimal disk schedule algorithm derives a retrieval sequence which minimizes the seek time as well as the rotational latency, discarding an isolated block from the sequence may not yield any significant reduction in the service time (i.e., the difference  $f(w_{i-1}, w_i) + f(w_i, w_{i+1}) - f(w_{i-1}, w_{i+1})$ ) may not be significant. For instance, if blocks  $w_{i-1}$ ,  $w_i$  and  $w_{i+1}$  happen to be located on the same track/cylinder, then discarding block  $w_i$  does not yield any reduction in the service time at all. In fact, the average reduction in service time yielded for each discarded media block is higher if a sequence of  $n$  media blocks, rather than  $n$  isolated media blocks, are discarded during a round. Consequently, any scheme that is based solely on the loss affordability of clients may discard a larger number of media blocks, than are necessary, during an overflow round.

#### **4.5.4.3 Technique Based on Loss Affordability and Block Position**

To address this limitation, we present an algorithm that selects a set of media blocks to be discarded during a round based on both the loss affordability of the



clients as well as the relative placement of the selected blocks on the disk [16]. To clarify the exposition of the algorithm, let us assume that  $w_0, w_1, \dots, w_N$  denotes the retrieval sequence derived by the near optimal disk-scheduling algorithm. Based on the loss affordability of each strand, the algorithm first labels each media block (i.e.,  $w_i$ 's) to be retrieved during the round as either can-be-discarded or can-not-be-discarded, and then determines maximal length subsequences of can-be-discarded blocks. For each such subsequence, the server computes the average reduction in service time per node yielded by discarding the entire subsequence. The algorithm then discards as many complete subsequences as needed, in the decreasing order of the average reduction, so as to make  $\tau \leq R$ , with possibly the last subsequence discarded partially.

If, even after discarding all the subsequences, the service time continues to be greater than the duration of the round, then the subsequence determination algorithm is repeated with all the blocks of normal clients marked as can-be-discarded. Whereas this would violate the requirements of some of the normal clients, proper choice of  $\epsilon$  in the admission control criteria (Equation 4.4) virtually eliminates the possibility of such an event.

The labeling operations as well as the processing of deriving subsequences of can-be-discarded blocks are relatively straightforward. Hence, the algorithm not only reduces the number of media blocks that are discarded during Round  $j$ , but also does so efficiently.

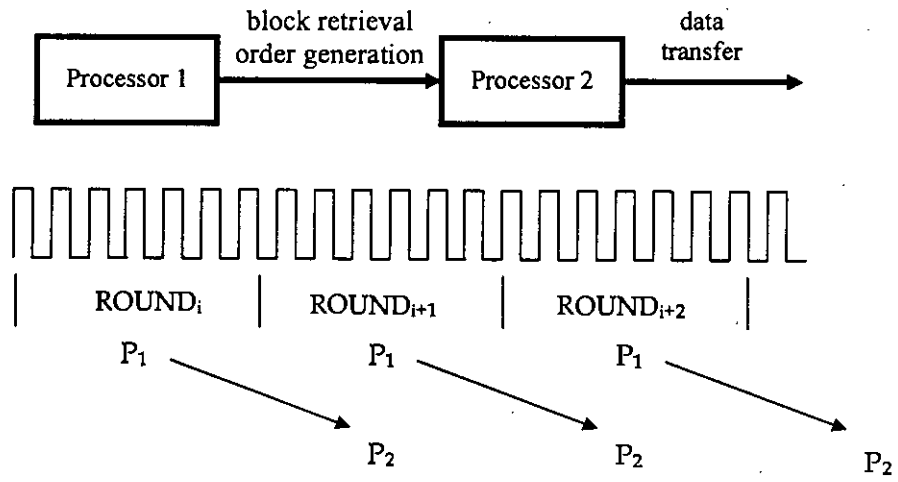
## 4.6 Analysis of Hybrid Admission Control Algorithm

In hybrid admission control algorithm there are two types of user. We classify them into two groups; super user group and normal user group. Each group has an average rate of inter-arrival times. In simulation, client requests were generated using poisson process.

- After classifying client request, multimedia server tests this request to admit it for accessing server resources by using hybrid admission control

algorithm. The computational time to test admission condition, i.e. whether the request is accepted or declined, is linear. So computational complexity of hybrid admission control algorithm is  $O(n)$ , where  $n$ , is the number of clients that are currently accessing server resources.

- Next step of the multimedia server is to determine the blocks that are needed to retrieve in the next round. After determining the blocks, disk-scheduling algorithm is responsible to generate the order of blocks to be retrieved. Our near optimal disk-scheduling algorithm does a good job and it takes a running time of  $\Theta(n^2)$ , where  $n$  is the number of blocks to be retrieved in a round.
- The computation complexity of deterministic admission control algorithm is constant. But block retrieval time depends on the disk-scheduling algorithm it uses. In contrast, the computation complexity of statistical admission control algorithm is  $O(n)$ , where,  $n$  is the number of clients that are currently accessing server resources.
- Hybrid admission control algorithm uses near optimal disk scheduling algorithm and this algorithm has a higher running time than other conventional disk scheduling algorithms, but it benefits lies in the generation of the near optimal block order sequence.
- Now, we propose system architecture (fig 4.2) to get better performance from the hybrid admission control algorithm. This architecture has two processors. One processor (P1) is used to do all the preprocessing works such as to determine the blocks to be retrieved in the next round as well as actual retrieval sequence of blocks by using near optimal disk scheduling algorithm. Other processor (P2) does the actual data transfer. This architecture also indicates that processor P2 is one round behind the processor P1. The extra overhead of near optimal disk scheduling algorithm is pulverized by this architecture.



*Fig 4.3 Dual processors architecture*

# Chapter 5

## Simulation and Experimental Results

---

### 5.1 Introduction

So far, we have presented an admission control algorithm and a disk scheduling technique for multimedia server. In this chapter, we demonstrate their viability by analyzing the performance of both disk scheduling scheme and admission control algorithm through simulations. The performance measurement of disk scheduling algorithm is achieved by comparing the results with various well-known disk-scheduling techniques. The performance measures of different admission control schemes may help us to understand how various systems can be better utilized to serve multimedia clients. The following metrics were used to evaluate our hybrid admission controllers and compare their performance with deterministic and statistical admission control schemes.

1. **Number of Clients Admitted:** The total number of clients that were admitted for a multimedia session out of a large number of requesting clients.
2. **Number of Overflow Rounds:** In serving the multimedia clients, there could be a number of rounds that may exceed the duration of a round.

To evaluate the performance of the near optimal disk-scheduling algorithm we use following metrics:

1. **Service Time:** How much time it takes when retrieving large number of blocks from the disk.
2. **Rotational Latency:** Influence of rotation latency in different disk scheduling algorithms.

## 5.2 Simulation Parameters

The simulations were carried out in an environment consisting of a synchronous disk array (access arms/heads move in unison) with 16 disks. The characteristics of disk are shown in Table 5.1. The unit of each parameter is also attached in the table.

Parameter	Relevant Information	Unit
Disk Capacity	4	GB
Number of disks in the array	16	
Number of tracks per disk	1024	
Disk block size	32	KB
Rate of disk rotation	3600	RPM
Seek formula	$4+0.02* C_1 - C_2 $	ms.
Max seek time	24.48	ms
Max rotational latency	16.66	ms
Number of blocks per track	128	

*Table 5.1 Disk parameters assumed in the simulation*

For our simulations, we assume client requests received by multimedia server are categorized into two groups: Super user and Normal user. Normal user has a tolerance level. Client requests of each group were generated using a poisson process, with average inter-arrival times of 3 seconds and 1 second, respectively. During each round, exactly one block from each of the disks was retrieved for each client. Since the block size is assumed to be 32 KB, the total amount of media information retrieved during each round is 512 KB. Assuming that the playback rate of each strand is 512 KB/second yields  $R = 1$  second ( $R$  is the minimum duration of a round). The simulation parameters for admission control algorithm are given in Table 5.2.

### 5.3 Experimental Results of Near Optimal Disk Scheduling Algorithm

The near optimal disk scheduling algorithm derives a sequence of accessing blocks from the disk so as to simultaneously minimize seek time and rotational latency. In contrast, conventional disk scheduling algorithms such as SCAN optimizes the total seek time only. Fig. 5.1 shows time requirements of various algorithms when retrieving 100 blocks. Here it is quite clear that near optimal disk scheduling algorithm is better than the other disk scheduling algorithms when retrieving a large number of blocks.

Parameter	Values
Playback rate	30 frames/sec
Frame Size	16.25 KB
Average inter-arrival times of Super user	3
Average inter-arrival times of Normal user	1
Average no of frames retrieve on time, P	95%
No of previous rounds used to determine average block retrieval time	20
Epsilon/ Overflow control parameter	0.0 – 1.0
Safe Guard	0 %– 10%

*Table 5.2 Simulation parameters of admission control algorithm*

Fig. 5.2 shows the variation in the ratio of service times derived for SCAN and the near optimal algorithms with the increase in the number of blocks,  $n$ , retrieved during each round. Observed that the performance of the near optimal scheduling technique improves with increase in  $n$ . This is because, at smaller values of  $n$ , the total seek time incurred during their retrieval dominates the performance and hence, SCAN performs as well as the near optimal. However, as the value of  $n$  increases, the cumulative rotational latency starts dominating the service time, thereby enabling the near optimal to outperform SCAN. Fig. 5.2 also indicates

that even at higher values of rotational rate, the gain in performance yielded by the near optimal algorithm is significant.

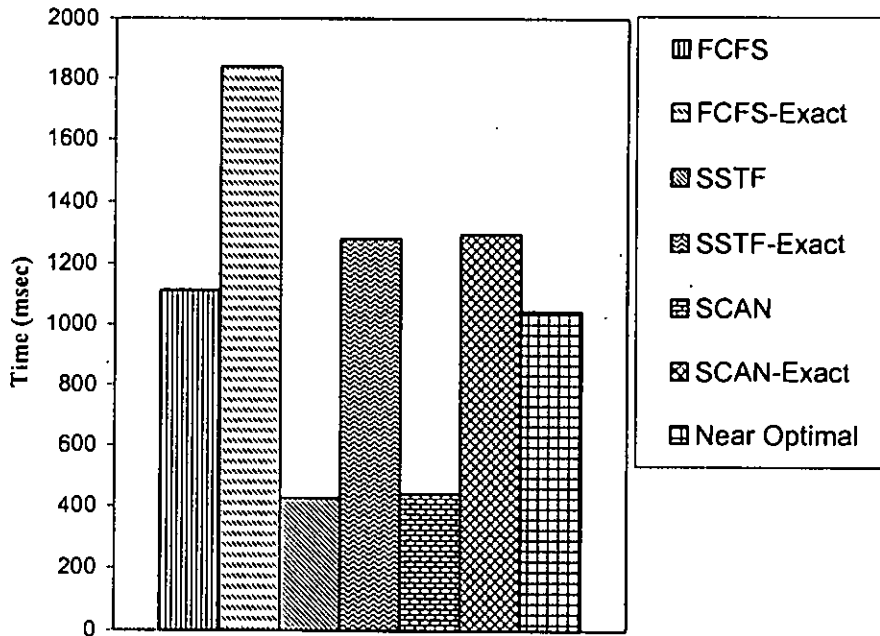


Fig 5.1 Service time comparison of disk scheduling algorithms when retrieving 100 blocks

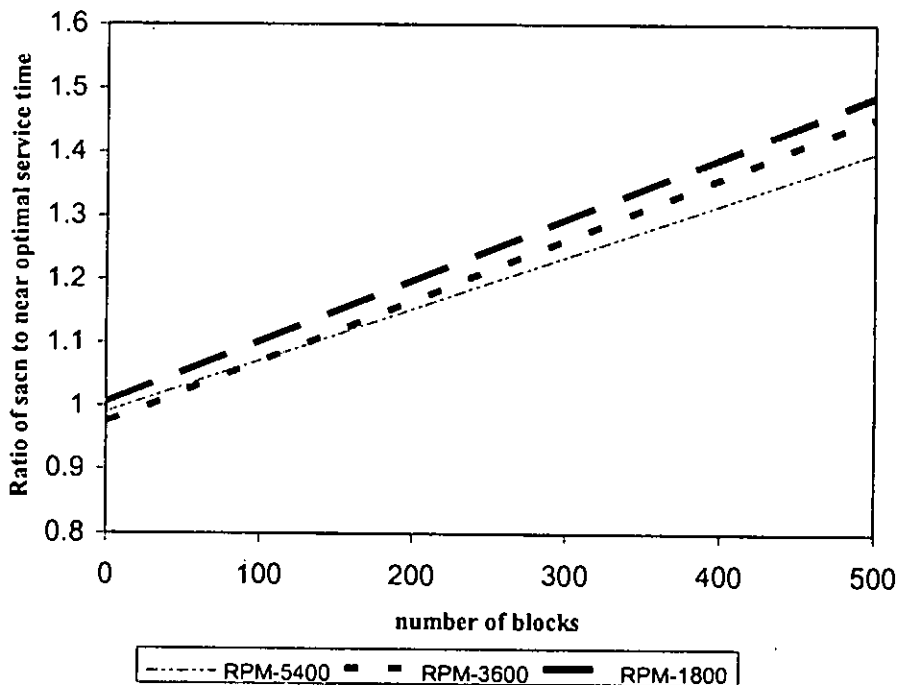


Fig 5.2 Comparison of SCAN and Near Optimal disk scheduling algorithms

Finally in fig. 5.3, we compare various disk-scheduling algorithms, where x-axis represents number of blocks and y-axis represents service time in millisecond. It is quite clear that near optimal disk scheduling algorithm is better than its counter parts for higher number of blocks.

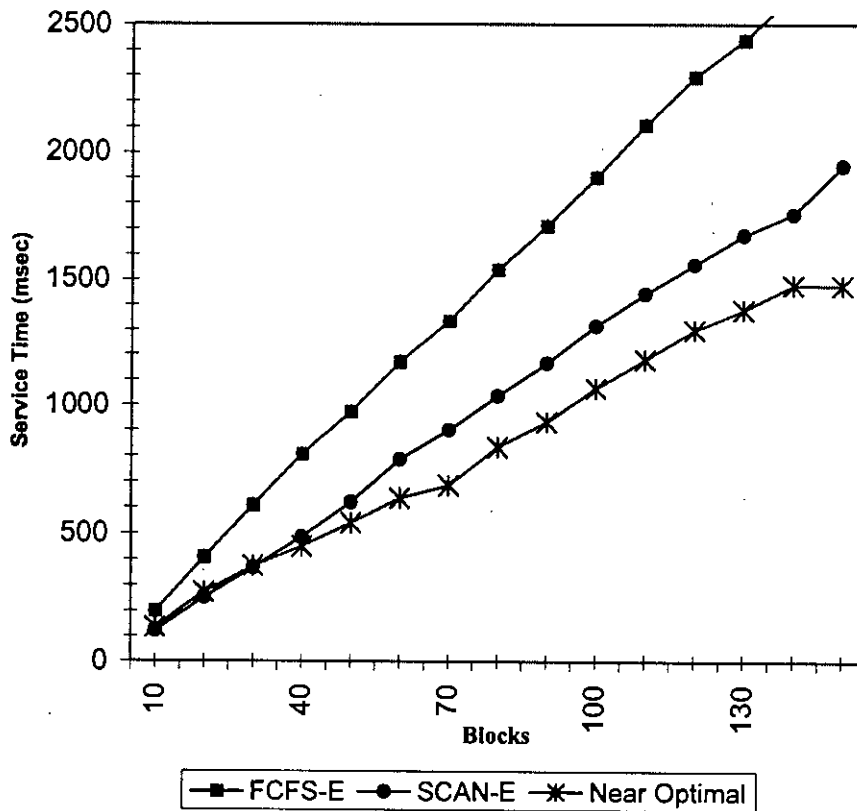


Fig 5.3 Time Analysis of Various Disk Scheduling Algorithms

## 5.4 Experimental Results of Hybrid Admission Control Algorithm

We have compared the performance of our hybrid admission control algorithm with conventional deterministic and statistical admission control algorithm. The results of this experiment show that hybrid admission control mechanism admits quite a reasonable number of clients. This number is higher than deterministic admission control algorithm but lower than statistical admission control mechanism. But average overflow round is lower in hybrid admission control



algorithm than that of statistical admission control technique. Table 5.3 summarizes the information in this regard.

	<b>Deterministic Admission Control Algorithm</b>	<b>Statistical Admission Control Algorithm</b>	<b>Hybrid Admission Control Algorithm</b>
No of clients served	47	135	97
No of clients increased	-	187%	106%
Overflow rounds	No	Very high	Very low

*Table 5.3 Comparison of different admission control algorithms.*

In our algorithm, we are also proposing two approaches to control overflow round. These are safe guard (Section 4.3) and epsilon (Section 4.3). We varied safe guard from 0% to 10% and epsilon from 0.0 to 1.0. The percentages of overflow rounds were determined and some of these results are shown in the table 5.4 and table 5.5.

<b>Safe Guard</b>	<b>Max Clients</b>	<b>% Of Overflow</b>
0%	96	63
2.5%	96	43
5%	94	17
7.5%	92	4
10%	89	1

*Table 5.4 Influence of safe guard on number of clients and overflow rounds for  $\epsilon = 0$ .*

Safe Guard	Max Clients	% Of Overflow
0%	97	53
2.5%	96	41
5%	94	14
7.5%	92	3
10%	89	0.15

Table 5.5 Influence of safe guard on number of clients and overflow rounds for  $\epsilon = 1$ .

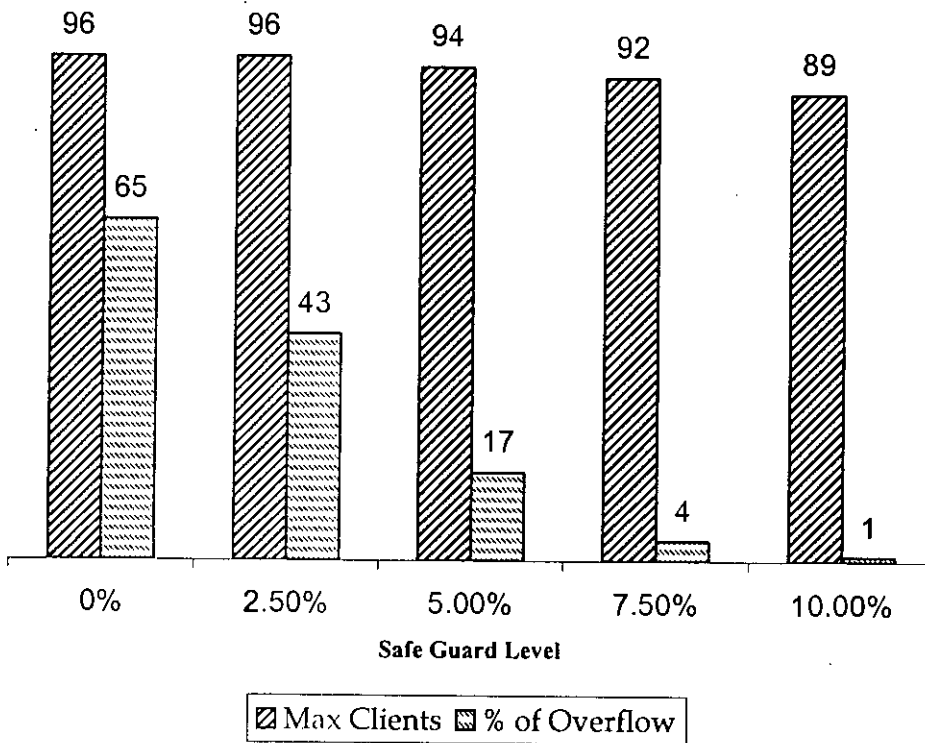
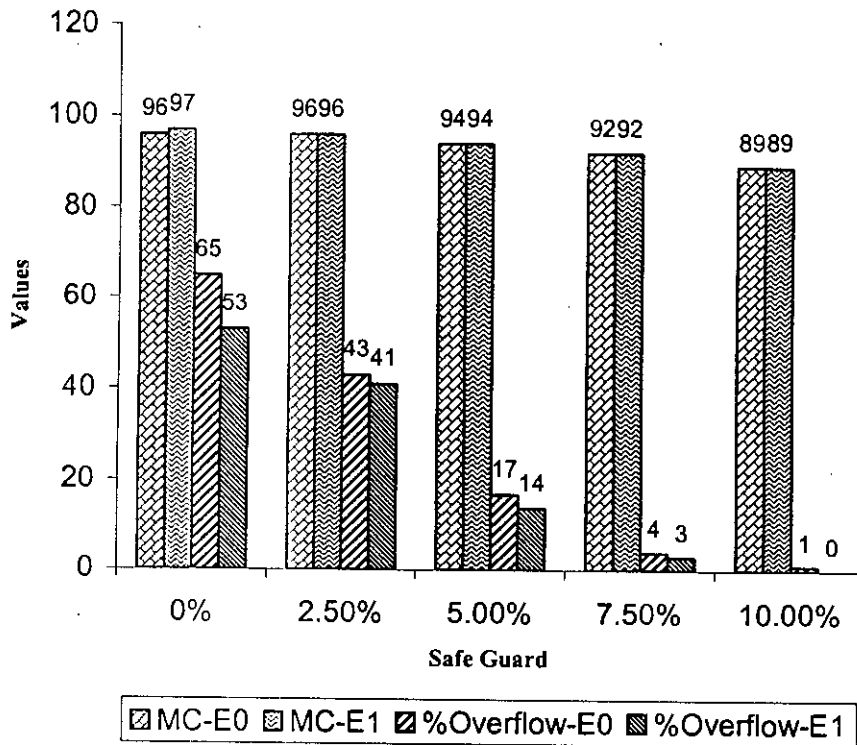


Fig 5.4 Influence of Safe Guard on the Number of Clients and Overflow Rounds

From the fig 5.4 it is clear that if we increase the value of safe guard, overflow rounds decrease with the slight expense of clients. This nature of decrease is almost exponential.

By controlling safe guard we can restrict the overflow of rounds but it reduces the number of clients that may be served by the multimedia server. But when safe guard is fixed we can control the overflow of rounds by adjusting the value of epsilon (Fig 5.5). Here we observed an interesting result. Epsilon has no influence on the number of clients but it reduces the overflow of rounds slightly when safe guard is fixed. This is because epsilon has little influence in admission policy; it only adjusts the average block retrieval time for the next round. For example, when safe guard is 5%, total number of clients that may be served by the server is 92. This value is same for any epsilon 0 to 1. But variation of epsilon changes overflow of rounds.



**Fig 5.5 Influence of Epsilon on the Number of Clients and Overflow Rounds**  
 (MC-E0: Maximum number of clients when  $\epsilon=0$ , MC-E1: Maximum number of clients when  $\epsilon=1$ , % Overflow-E0: Percentage of overflow when  $\epsilon=0$ , % Overflow-E1: Percentage of overflow when  $\epsilon=1$ )

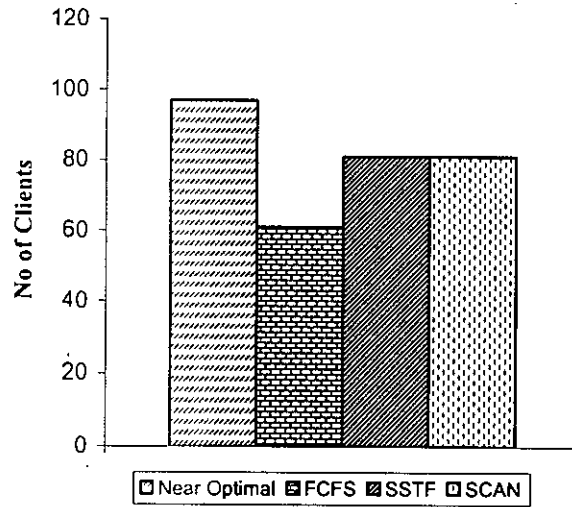


Fig 5.6 Evaluation of Hybrid Admission control algorithm

Fig 5.6 also illustrates that multimedia server employing near optimal disk scheduling algorithm can serve a larger number of clients simultaneously as compared to the other disk scheduling algorithms. In fact, for  $\epsilon=0$ , a multimedia server with hybrid admission control algorithm can serve almost 96 clients simultaneously by using near optimal disk scheduling algorithm whereas SCAN reduces this number to 81.

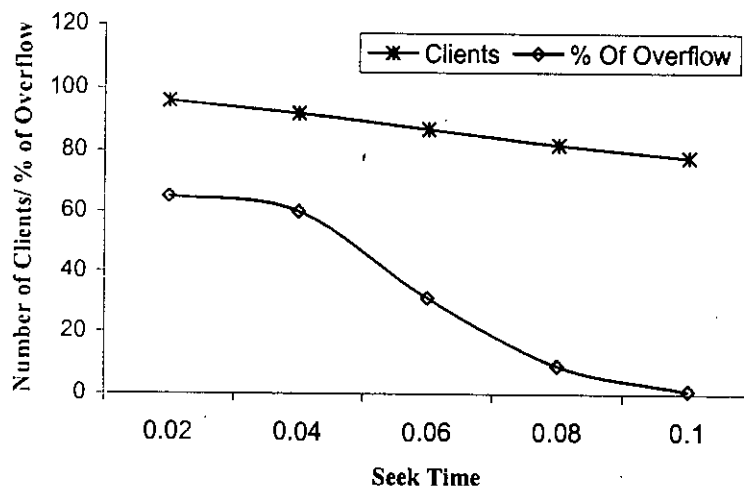
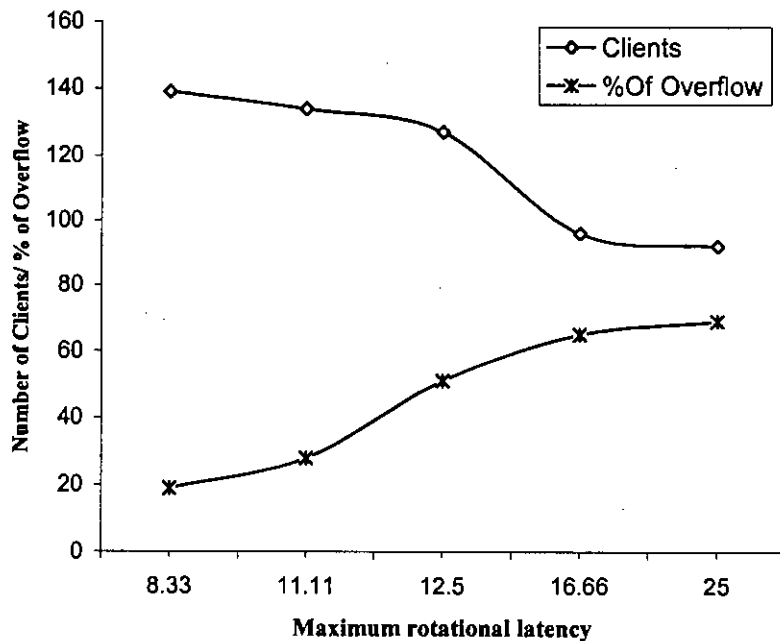


Fig 5.7 Influence of the seek time on clients and overflow rounds

In fig 5.7 we demonstrate the influence of seek time on the maximum number of clients that can be served simultaneously and on the percentage of overflow rounds. Here we vary seek time and found that with the increase in seek time number of admitted clients decrease slightly but percentage of overflow rounds decrease significantly.



*Fig 5.8 Influence of the maximum rotational latency on clients and overflow rounds*

In fig 5.8 we illustrate the influence of maximum rotational latency on the maximum number of clients that can be served simultaneously and on the percentage of overflow rounds. Here we change disk rotation, i.e. rotational latency, and found that with the increased rotational latency percentage of overflow rounds increase and the maximum number of clients that can be served simultaneously decrease. From the above two figures we may conclude that both the seek time and the rotational latency are very important parameters for admission control mechanism. As the performance of hybrid admission control algorithm is dependent on disk scheduling algorithm very closely and so seek time and rotational latency.

Finally in the fig 5.9 we present the performance of hybrid admission control algorithm by varying disk capacity. Here we found that the number of clients admitted by the multimedia server does not change significantly with the disk capacity but it is slightly decreasing. This is because a large disk has more tracks and in large disk, media blocks request are distributed in a wider range of tracks.

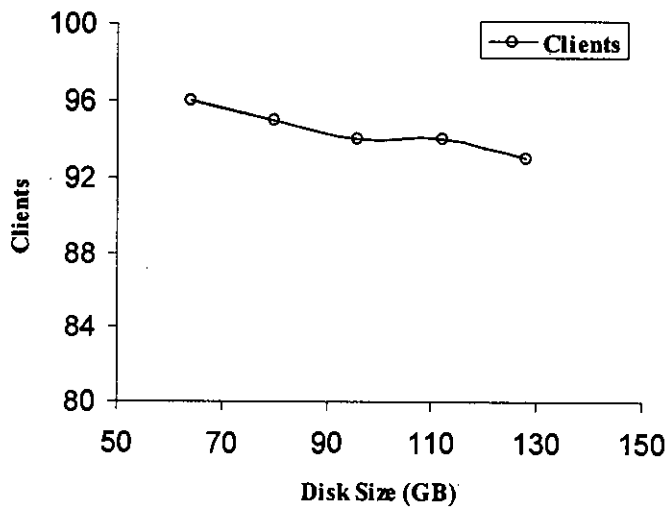


Fig 5.9 Relationship between clients and disk capacity

## 5.5 Conclusion

Experimental results indicate that hybrid admission control algorithm can serve more clients than the deterministic admission control algorithm but less than the statistical admission policy. But, hybrid admission technique has a very good control over the overflow rounds. In this chapter we also illustrated two techniques, sage guard and epsilon ( $\epsilon$ ), to control the overflow of rounds. We presented that disk parameters are very important over the performance of multimedia server. Disk scheduling algorithm that we presented in this thesis is better than the most of the conventional disk scheduling algorithms. Moreover, by using new disk scheduling algorithm we can serve more clients simultaneously than others.

# Chapter 6

## Conclusion and Recommendations

---

### 6.1 Concluding Words

In this thesis we derive a new admission control algorithm for multimedia server to accept enough traffic to efficiently utilize the server resources, while not accepting clients whose admission may lead to the violations of the service requirements of clients. The near optimal disk scheduling algorithm as well as the technique for minimizing overflow of rounds presented in this thesis significantly improve the performance of the admission control algorithm. We have demonstrated the effectiveness of the hybrid admission control algorithm and the near optimal disk-scheduling algorithm through simulations.

To summarize, the main contributions presented in this research are as follows:

1. Providing deterministic service guarantees to each client may needlessly constrain the number of clients that are served by a multimedia server, and hence, may lead to severe under-utilization of server resources. To address this limitation, we have presented a hybrid admission control algorithm, in which a multimedia server admits, a client only if it satisfies admission criteria of the hybrid admission control algorithm and the service requirements of all the clients can be met satisfactorily.
2. The effectiveness of this admission criterion is illustrated through extensive simulations. By using hybrid admission control algorithm multimedia server can serve more clients than deterministic admission control algorithm simultaneously. In comparison, the deterministic admission control algorithm only serves 47 clients, thereby demonstrating that the hybrid admission control algorithm increases the number of clients served simultaneously by about 106%.

- 99672
3. In this thesis we demonstrate two terms, safe guard and epsilon, by adjusting their values we can control the overflow of rounds. Safe guard mechanism reduces overflow of rounds exponentially with the slight expense of the number of clients served simultaneously. When safe guard is fixed, we can limit the overflow of rounds slightly again by adjusting the value of epsilon without reducing the number of clients.
  4. The performance of our hybrid admission control algorithm is critically dependent on the disk-scheduling algorithm. Most of the conventional disk scheduling algorithms (such as, SCAN, Shortest Seek Time First (SSTF), etc.) has addressed the problem of optimizing total seek time incurred while accessing a sequence of blocks from disk. The fundamental limitation of these algorithms, however, is that they optimize only the seek time, and completely ignore the rotational latency. In this thesis, we present a new disk-scheduling algorithm, i.e., near optimal disk scheduling algorithm, that derives a sequence for accessing media blocks from disk so as to simultaneously minimize both seek and rotational latency incurred during retrieval.
  5. The performance of near optimal disk scheduling is better than the other disk scheduling algorithms. The effectiveness of this algorithm is demonstrated through simulations. We have shown that as the number of blocks retrieved in each round increases near optimal disk scheduling starts to dominant other disk scheduling algorithms.

## 6.2 Recommendations for Future Work

In order to understand the requirements for handling various kinds of audio and video applications within the framework of existing operating systems and for building new ones, we believe that there are a number of areas that requires further research. This section essentially provides some pointers to pursue further investigation for building better video/audio servers.



1. One of the most important aspects of data storage that affects disk retrieval speed is the manner in which data has been organized on the disk (in short the file system). Typically, if the data belonging to a file are spread out in the disk, then a typical access to the disk is more likely to look like a random seek rather than a sequential seek. Random seeks not only increase retrieval times but also can result in little or no utilization of any caching policy the operating system might employ to reduce subsequent accesses to the same data. The greater the sequential storage of files, the lower the times for retrieval, since subsequent accesses to the data in a file has a greater chance of being served from the cache. Therefore, a direction for further research could be the design of a file system that is multimedia friendly and works well in conjunction with the way disk and operating system. Such a file system could be used to employ various schemes for storage. For example, since some blocks of data could be more important than others, this information can be stored in the file system and can be used to group more important blocks together. Also, it can help the disk scheduler to make decision easily and quickly about how it can drop disk requests in over-loaded round with minimal impact to the quality of the playback.
2. In this thesis we demonstrated a new disk-scheduling algorithm, i.e. near optimal disk scheduling algorithm that uses a heuristic approach to determine the order of blocks retrieval. This is an approximation algorithm with a ratio bound of 2 for the traveling salesman problem with triangular inequality [60]. The running time of this algorithm is  $\Theta(E) = \Theta(V^2)$ , since the input is a complete graph. Some potential research topics would be the determination of the order of block sequences by using a heuristic with a better running time.
3. Hybrid admission control algorithm that we presented in this thesis can serve more clients than that of deterministic approach. Hybrid technique has a very good control over the overflow of rounds. Some research still

can be done to increase the number of clients that can be served by the multimedia server simultaneously as well as reduce the overflow of rounds. The performance of admission control algorithms is dependent on admission policy and disk scheduling algorithm. If admission policy is strict few clients get chance for accessing server resources but definitely it reduces overflows of rounds. On the other hand, simple admission policy may admit large number of clients but it introduces high overflow of rounds. So, we need to find an optimal situation where overflow of rounds does not hamper the performance of the multimedia server.

4. Network delay can be classified as end-to-end delay and delay at resource. The delay "at the resource" is the maximum time span for the completion of a certain task at the resource. The end-to-end is the total delay for a data unit to be transmitted from the source to its destination. It is quite difficult to determine their percentage. This depends on the multimedia server and the networks in which server and clients are running. As end-to-end delay is reducing due to fast advancing networking technologies, it is now the time to speed up the processing at server as well.
5. One of the most interesting research topics would be to design system architecture for multimedia server with parallel processors and multiple disks with their own file system. How well this hybrid admission policy performs in this environment? What adjustments are needed to achieve high performance?

## Appendix A

---

**File Name:** *Parameter.h*

```
#include <stdio.h>
#include <vector>

// Disk Parameters
#define NoOfDisks 16 // No of Disk array
#define TPD 1024
#define BPT 128
#define BSize 32
#define MST 24.48 // ms
#define MRL 16.66 // ms
#define a 4
#define b 0.02

// Admission Control Parameters

#define PBR 30 // frame per second
#define FrameSize 16.25 // 16.25 KB or 130000
#define RD 1000 // Round duration in msec
#define ClassA 0
#define ClassB 1
#define P 0.95
#define ROUND 20

#define INFINITE 99999
#define NIL -1
#define UP 1
#define DOWN -1
```

```
#define MAX 400
struct BLOCK{
    int track;
    int block;
};
struct Graph{
    BLOCK bl;
    double key;
    int pi;
    int inQueue;
    double wt[MAX];
};

struct Directory{
    int filename;
    int fileSizeInMB;
    BLOCK startBlock;
};

struct DirectoryMap{
    bool notfree;
    int nextTrack;
    int nextBlock;
};

struct Client{
    int reqFile;
    int Class;
    int curT;
    int curB;
    unsigned int blockForFile;
};

using namespace std;
void DiskSim();
void random_block_gen();
double ServiceTime();
double FCFS();
double SSF();
```

```
void SortReq(BLOCK *r);
double ExactTime(BLOCK *r);
double MST_PRIM(int r);
void InitializeGraph(Graph *graph, int r);
int Extract_MIN(Graph *graph);
void NodeOrderGen(int list[MAX][MAX],int r);
double ELEVATOR();
int CreateFile(int fileName,int sizeInMB);
void ShowFileSystem();
void SetGlobalParameters();
void AdjustFat(unsigned int nBlock);
int Class_A();
int Class_B();
void GenerateFiles();
double SrvTimeSuper();
double SrvTimeOther();
int DetAdmissionCA();
int HybridAdmissionCA(int cl);
void UpdateClient();
void CalAvgBTime();
double genrand();
void sgenrand(unsigned long seed);
```

**File Name:** *block request.cpp*

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include "parameter.h"

extern BLOCK bl[MAX];
int cur,constDir=UP;;
int preorder[2*MAX];
extern unsigned int REQUEST;
extern double blockReadTime[ROUND];
extern double avgBTime,flag,epsilon;
extern int en,QosFail;

// Random block generation
void random_block_gen()
{
    int i;

    srand( (unsigned)time( NULL ) );

    for( i = 0; i < REQUEST; i++){
        bl[i].track = rand() % TPD;
        bl[i].block = rand() % BPT;
        // printf("\n<%4d, %3d>",bl[i].track,bl[i].block);
    }
    cur = bl[0].track;
}

// calculating service time for a sequence of block request
double ServiceTime(BLOCK *bl)
{
    int i;
    double t=0;
```

```
    for( i=1; i< REQUEST; i++){
        t += abs(bl[i-1].track - bl[i].track);
    }
    t = a*REQUEST + 0.02 * t;
    return t;
}
```

// First Come First Serve Strategy

```
double FCFS()
{
    double srvTime = ServiceTime(bl);
    // printf(" %6.f", srvTime);
    srvTime = ExactTime(bl);
    // printf("\t%6.f",srvTime);
    return srvTime;
}
```

// Sorting the block request according to track number

```
void SortReq(BLOCK *r)
{
    int i,j;
    BLOCK temp;
    for(i=0; i<REQUEST-1; i++)
        for(j=i+1; j<REQUEST; j++){
            if(r[i].track > r[j].track){
                temp = r[i];
                r[i] = r[j];
                r[j] = temp;
            }
        }
    for( i = 0; i < REQUEST; i++)
        if(r[i].track == cur) cur = i;
}
```

```
// Shortest Seek First Strategy
double SSF()
{
    BLOCK *ssf;
    ssf = new BLOCK[REQUEST];
    int i,x,y,z;
    int left=0,right=0;
    double srvTime,movement=0;

    for(i=0; i<REQUEST; i++)
        ssf[i] = bl[i];
    SortReq(ssf);
    for(i=1; i<REQUEST; i++){
        x = ssf[cur-left-1].track;
        y = ssf[cur+right+1].track;
        z = ssf[cur].track;
        if( abs(z-x) < abs(z-y) ) {
            movement += abs(z-x);
            cur = cur-left-1;
            right = i;
            left = 0;
        }
        else {
            movement += abs(z-y);
            cur = cur+right+1;
            left = i;
            right = 0;
        }
    }
    srvTime = a*REQUEST + b*movement;
    // printf("\t%6.f",srvTime);
    // printf("\t%6.f",ExactTime(ssf));
    srvTime = ExactTime(ssf);
    delete [] ssf;
    return srvTime;
}
```



```
double ExactTime(BLOCK *r)
{
    double perST, perRL, SrvTime;
    int i, curT, curB, waitB;
    curT = r[0].track;
    curB = r[0].block;
    SrvTime = 0;
    for( i=1; i<REQUEST; i++){
        perST = a + b * abs(r[i-1].track - r[i].track);
        curB = ( int ) ceil((BPT/MRL)*perST);
        curB = (curB + r[i-1].block) % BPT;
        if(r[i].block > curB)
            waitB = r[i].block - curB;
        else
            waitB = r[i].block - curB + BPT;
        perRL = (MRL * waitB) / BPT;
        SrvTime += perST + perRL;
    }
    return SrvTime;
}

void InitializeGraph(Graph *graph, int r)
{
    int i, j, curB, waitB;
    double perST, perRL;
    for( i=0; i<REQUEST; i++){
        graph[i].bl = bl[i];
        graph[i].key = INFINITE;
        graph[i].pi = NIL;
        graph[i].inQueue = 1;
    }
    graph[r].key = 0;

    for( i=0; i<REQUEST; i++)
        for( j=0; j<REQUEST; j++){
            if( i==j ) { graph[i].wt[j] = 0; continue; }
```

```
        perST = a + b * abs(graph[i].bl.track - graph[j].bl.track);
        curB = ( int ) ceil((BPT/MRL)*perST);
        curB = (curB + graph[i].bl.block) % BPT;
        if(graph[j].bl.block > curB)
            waitB = graph[j].bl.block - curB;
        else
            waitB = graph[j].bl.block - curB + BPT;
        perRL = (MRL * waitB) / BPT;
        graph[i].wt[j] = perST + perRL;
    }
}
```

```
int Extract_MIN(Graph *graph)
{
    int i,min;
    double minVal;
    for(i=0, minVal = INFINITE+1; i<REQUEST; i++){
        if(graph[i].inQueue == 0) continue;
        if(graph[i].key < minVal){
            minVal = graph[i].key;
            min = i;
        }
    }
    graph[min].inQueue = 0;
    return min;
}
```

```
void NodeOrderGen(int list[MAX][MAX],int r)
{
    int i;
    preorder[en++] = r;
    // printf("%4d",preorder[n-1]);
    for(i=0;i<REQUEST;i++){
        if(list[r][i] == NIL) return;
        NodeOrderGen(list,list[r][i]);
    }
}
```

```
double MST_PRIM(int r)
{
    Graph *graph;
    int i,j,u,v,list[MAX][MAX];
    int parent;
    BLOCK *mst;
    double srvTime;

    mst = new BLOCK[REQUEST];
    graph = new Graph[REQUEST];
    for( i=0;i<REQUEST;i++)
        for(j=0;j<REQUEST;j++)
            list[i][j]= NIL;
    // initializing the graph
    InitializeGraph(graph, r);
    // applying the MST-PRIM algorithm
    for( i=0;i<REQUEST; i++){
        u = Extract_MIN(graph);
        // adjacent nodes
        for( v=0; v<REQUEST; v++){
            if((graph[v].inQueue == 1) && (graph[u].wt[v]<graph[v].key)){
                graph[v].pi = u;
                graph[v].key = graph[u].wt[v];
            }
        }
    }
    // Constructing the tree
    for( i=0;i<REQUEST; i++){
        parent = graph[i].pi;
        if( parent != NIL ){
            for(j=0;j<REQUEST;j++){
                if (list[parent][j] != -1) continue;
                list[parent][j] = i;
                break;
            }
        }
    }
}
```

```
    }
/* printing the child
    for( i=0;i<REQUEST; i++){
        printf("\nParent: %4d\t",i);
        for( j=0;j<REQUEST; j++){
            if (list[i][j] == NIL) break;
            printf("%4d", list[i][j]);
        }
    }
*/

// Node order generation
en = 0;
NodeOrderGen(list,r);
for(i=0; i<REQUEST; i++){
    mst[i] = graph[preorder[i]].bl;
}
srvTime = ExactTime(mst);
// printf("\t\t%6.f",srvTime);
delete [] mst;
delete [] graph;
return srvTime;
}

double ELEVATOR()
{
    BLOCK *elevator,*temp;
    double srvTime;
    int i;
    elevator = new BLOCK[REQUEST];
    temp = new BLOCK[REQUEST];

    for(i=0; i<REQUEST; i++)
        temp[i] = bl[i];
    SortReq(temp);
    for( i=0; i<REQUEST; i++){
        elevator[i] = temp[cur];
    }
}
```

```
        cur = (cur+1)%REQUEST;
    }
    srvTime = ServiceTime(elevator);
//    printf("\t%6.f",srvTime);
    srvTime = ExactTime(elevator);
//    printf("\t%6.f", srvTime);
    delete [] temp;
    delete [] elevator;
    return srvTime;
}

void CalAvgBTime()
{
    int j;
    double curBReadTime,term,stddev;
    static int i=0;
    curBReadTime = MST_PRIM(0);
//    curBReadTime = FCFS();
//    curBReadTime = SSF();
//    curBReadTime = ELEVATOR();
    if(curBReadTime>1000) {
        QosFail++;
        //    printf("\nQoS Fail: %3d",QosFail);
    }

    curBReadTime /= REQUEST;
    if(curBReadTime<=0) { printf("\nNeg\n");return;}
    blockReadTime[i%ROUND] = curBReadTime;
    i = (i+1) % ROUND;
    avgBTime = 0;
    stddev = 0;
    if(flag==0){
        for(j=0;j<i;j++){
            avgBTime += blockReadTime[j];
        }
        avgBTime = avgBTime/i;
        for(j=0;j<i;j++){
```

```
        term = avgBTime - blockReadTime[j];
        stddev += term*term;
    }
    stddev = sqrt(stddev)/i;
}
else {
    for(j=0;j<ROUND;j++)
        avgBTime += blockReadTime[j];
    avgBTime = avgBTime/ROUND;
    for(j=0;j<i;j++){
        term = avgBTime - blockReadTime[j];
        stddev += term*term;
    }
    stddev = sqrt(stddev)/ROUND;
    flag = 1;
}
avgBTime = avgBTime + epsilon*stddev;
}
```

**File Name:** *filesystem.cpp*

```
#include "parameter.h"
#include <math.h>
#include <time.h>
extern DirectoryMap fat[TPD][BPT];
extern vector<Directory> FileSystem;
extern BLOCK nextFreeBlock;
extern unsigned int noOfBlock;
extern int REQUEST;
extern int noOfFiles;

void AdjustFat(unsigned int nBlock)
{
    int i,track,block;
    track = nextFreeBlock.track;
    block = nextFreeBlock.block;
    for(i=0; i<nBlock; i++){
        fat[track][block].notfree = 1;
        block = (block+1) % BPT;
        if (block==0) track++;
    }

    nextFreeBlock.track = track;
    nextFreeBlock.block = block;
}

int CreateFile(int fileName, int sizeInMB)
{
    Directory tempFile;
    unsigned int usedBlock,requiredBlock,availableBlock,sizeInKB;
    sizeInKB = sizeInMB * 1024;
    usedBlock = nextFreeBlock.track*BPT + nextFreeBlock.block;
    requiredBlock = ceil((1.0*sizeInKB)/BSize);
    availableBlock = noOfBlock-usedBlock;
```

```
    if(availableBlock >= requiredBlock){
        tempFile.filename = fileName;
        tempFile.fileSizeInMB = sizeInMB;
        tempFile.startBlock = nextFreeBlock;
        FileSystem.push_back(tempFile);
        AdjustFat(requiredBlock);
        return 1;
    }
    else return 0;
}

void ShowFileSystem()
{
    int i,fileName;
    for( i=0; i<FileSystem.size(); i++){
        fileName = FileSystem.at(i).filename;
        printf("\nFileName: %d",fileName);
        printf("\tSize: %d",FileSystem.at(i).fileSizeInMB);
        printf("\tStart at <Track,Block>: <%4d,%4d>
",FileSystem.at(i).startBlock.track,FileSystem.at(i).startBlock.block);
    }
}

int Class_A()
{
    double term,sum,val;
    int n;
    // sgenrand((unsigned)time( NULL ));
    val = genrand();
    term = -0.33*log(val);
    sum = term;
    n = 1;
    while( sum < 1){
        val = genrand();
        term = -0.33*log(val);
        sum += term;
    }
}
```



```
        n++;
    }
    return n;
}

int Class_B()
{
    double term,sum,val;
    int n;
    // sgenrand((unsigned)time( NULL ));
    val = genrand();
    term = -log(val);
    sum = term;
    n = 1;
    while( sum < 1){
        val = genrand();
        term = -log(val);
        sum += term;
        n++;
    }
    return n;
}

void GenerateFiles()
{
    int i;
    i = 0;
    while(1){
        if (CreateFile(i++,1+rand()%15)==0) break;
    }
    noOfFiles = FileSystem.size();
    // ShowFileSystem();
}
```

**File Name:** *mt19937.cpp*

```
/* A C-program for MT19937: Real number version */
/*  genrand() generates one pseudorandom real number (double) */
/*  which is uniformly distributed on [0,1]-interval, for each */
/*  call. sgenrand(seed) set initial values to the working area */
/*  of 624 words. Before genrand(), sgenrand(seed) must be */
/*  called once. (seed is any 32-bit integer except for 0). */
/*  Integer generator is obtained by modifying two lines. */
/*  Coded by Takuji Nishimura, considering the suggestions by */
/*  Tophy Cooper and Marc Rieffel in July-Aug. 1997. */

/* This library is free software; you can redistribute it and/or */
/* modify it under the terms of the GNU Library General Public */
/* License as published by the Free Software Foundation; either */
/* version 2 of the License, or (at your option) any later */
/* version. */
/* This library is distributed in the hope that it will be useful, */
/* but WITHOUT ANY WARRANTY; without even the implied warranty of */
/* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. */
/* See the GNU Library General Public License for more details. */
/* You should have received a copy of the GNU Library General */
/* Public License along with this library; if not, write to the */
/* Free Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA */
/* 02111-1307 USA */

/* Copyright (C) 1997 Makoto Matsumoto and Takuji Nishimura. */
/* Any feedback is very welcome. For any question, comments, */
/* see http://www.math.keio.ac.jp/matsumoto/emt.html or email */
/* matumoto@math.keio.ac.jp */

#include "parameter.h"
/* Period parameters */
#define N 624
#define M 397
#define MATRIX_A 0x9908b0df /* constant vector a */
```

```
#define UPPER_MASK 0x80000000 /* most significant w-r bits */
#define LOWER_MASK 0x7fffffff /* least significant r bits */
```

```
/* Tempering parameters */
```

```
#define TEMPERING_MASK_B 0x9d2c5680
```

```
#define TEMPERING_MASK_C 0xefc60000
```

```
#define TEMPERING_SHIFT_U(y) (y >> 11)
```

```
#define TEMPERING_SHIFT_S(y) (y << 7)
```

```
#define TEMPERING_SHIFT_T(y) (y << 15)
```

```
#define TEMPERING_SHIFT_L(y) (y >> 18)
```

```
static unsigned long mt[N]; /* the array for the state vector */
```

```
static int mti=N+1; /* mti==N+1 means mt[N] is not initialized */
```

```
/* initializing the array with a NONZERO seed */
```

```
void sgenrand(unsigned long seed)
```

```
{
```

```
    /* setting initial seeds to mt[N] using */
```

```
    /* the generator Line 25 of Table 1 in */
```

```
    /* [KNUTH 1981, The Art of Computer Programming */
```

```
    /* Vol. 2 (2nd Ed.), pp102] */
```

```
    mt[0]= seed & 0xffffffff;
```

```
    for (mti=1; mti<N; mti++)
```

```
        mt[mti] = (69069 * mt[mti-1]) & 0xffffffff;
```

```
}
```

```
/* generating reals */
```

```
/* unsigned long */ /* for integer generation */
```

```
double genrand()
```

```
{
```

```
    unsigned long y;
```

```
    static unsigned long mag01[2]={0x0, MATRIX_A};
```

```
    /* mag01[x] = x * MATRIX_A for x=0,1 */
```

```
    if (mti >= N) { /* generate N words at one time */
```

```
        int kk;
```

```
if (mti == N+1) /* if sgenrand() has not been called, */
    sgenrand(4357); /* a default initial seed is used */

for (kk=0;kk<N-M;kk++) {
    y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
    mt[kk] = mt[kk+M] ^ (y >> 1) ^ mag01[y & 0x1];
}
for (;kk<N-1;kk++) {
    y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
    mt[kk] = mt[kk+(M-N)] ^ (y >> 1) ^ mag01[y & 0x1];
}
y = (mt[N-1]&UPPER_MASK)|(mt[0]&LOWER_MASK);
mt[N-1] = mt[M-1] ^ (y >> 1) ^ mag01[y & 0x1];

mti = 0;
}

y = mt[mti++];
y ^= TEMPERING_SHIFT_U(y);
y ^= TEMPERING_SHIFT_S(y) & TEMPERING_MASK_B;
y ^= TEMPERING_SHIFT_T(y) & TEMPERING_MASK_C;
y ^= TEMPERING_SHIFT_L(y);

return ((double)y / (unsigned long)0xffffffff); /* reals */
/* return y; */ /* for integer generation */
}
```

**File Name:** *Disk Schedule Simulation.cpp*

```
#include "parameter.h"
void DiskSim()
{
    random_block_gen();
    FCFS();
    SSF();
    ELEVATOR();
    MST_PRIM(0);
    printf("\n");
}
```

**File Name:** *thesis.cpp*

```
#include "parameter.h"
#include <math.h>
#include <time.h>

vector<Directory> FileSystem;
vector<Client> client;
DirectoryMap fat[TPD][BPT];
unsigned int noOfBlock,superUser,norUser;
BLOCK nextFreeBlock;
unsigned int REQUEST;
int noOfFiles,en;
double avgBTime,flag;
BLOCK bl[MAX];
double blockReadTime[ROUND];
double epsilon;
int QosFail;

// Set the global parameters of the program
void SetGlobalParameters()
{
    noOfBlock = TPD * BPT;
    nextFreeBlock.track = 0;
    nextFreeBlock.block = 0;
    superUser = 0;
    norUser = 0;
    noOfFiles = 0;
    avgBTime = a + b + MRL;
    REQUEST = 0;
    flag = 0;
    epsilon = 0;
    QosFail = 0;
}
```

```
int DetAdmissionCA()
{
    double srvTime;
    srvTime = TPD * b + (a + MRL) * (client.size() + 1);
    if ( srvTime <= RD ) return 1;
    return 0;
}

int HybridAdmissionCA(int cl)
{
    double srvTime;
    int i;
    srvTime = 0;
    for(i=0;i<client.size();i++){
        if(cl==ClassA)
            srvTime += a + MRL;
        else
            srvTime += avgBTime*P;
    }
    srvTime += b*TPD;
    if(srvTime <= RD) return 1;
    return 0;
}

void UpdateClient()
{
    int i;
    REQUEST = 0;
    for( i=0; i<client.size(); i++){
        bl[REQUEST].track = client.at(i).curT;
        bl[REQUEST++].block = client.at(i).curB;
        client.at(i).blockForFile--;
        if(client.at(i).blockForFile == 0){
            if(client.at(i).Class == ClassA) superUser--;
            else norUser--;
            // printf("\nReleasing client: %d.",i);
        }
    }
}
```

```
        client.erase(client.begin() + i);
    }
}

// Analysis of Hybrid Admission Control Algorithm
void main()
{
    int fileno,ca,cb,maxClient;
    Client tempC;

//    clock_t start, finish;
//    double duration;

    // Set Global Parameters
    SetGlobalParameters();

    // Generating Files for our file system
    GenerateFiles();
    maxClient = 0;
    ca = cb = 0;
    long int r = 0;
    while(1){
        if(client.size()>maxClient){
            maxClient = client.size();
            //    printf("\n%d Clients",maxClient);
        }
        if (ca==0 && HybridAdmissionCA(ClassA)==1) {
            // serve the client
            fileno = rand() % noOfFiles;
            tempC.reqFile = fileno;
            tempC.Class = ClassA;
            tempC.curT = FileSystem.at(fileno).startBlock.track;
            tempC.curB = FileSystem.at(fileno).startBlock.block;
        }
    }
}
```



```
tempC.blockForFile =
    FileSystem.at(fileno).fileSizeInMB * 1024 / 32;

    client.push_back(tempC);
    superUser++;
    ca = Class_A();
}
if (cb==0 && HybridAdmissionCA(ClassB)==1) {
    // serve the client
    fileno = rand() % noOfFiles;
    tempC.reqFile = fileno;
    tempC.Class = ClassB;
    tempC.curT = FileSystem.at(fileno).startBlock.track;
    tempC.curB = FileSystem.at(fileno).startBlock.block;
    tempC.blockForFile =
        FileSystem.at(fileno).fileSizeInMB * 1024 / 32;

    client.push_back(tempC);
    norUser++;
    cb = Class_B();
}
ca = ((ca-1)<0)?0:(ca-1);
cb = ((cb-1)<0)?0:(cb-1);
UpdateClient();
CalAvgBTime();
r++;
// printf("\n\nSuper: %d\nNormal: %d",superUser,norUser);
}
/* printf("\nRounds: %d\nMax Client: %d \nOverflow: %d\n",
r,maxClient,QosFail); */
}
```

## Bibliography

---

- [1] P. R. Barham, "A Fresh Approach to File System Quality of Service", in Proceedings of NOSSDAV'97, (St. Louis, Missouri), pp. 119–128, May 1997.
- [2] M. L. Claypool and J. Reidl, "End-to-End Quality in Multimedia Applications", in Handbook on Multimedia Computing, ch. 40, Boca Raton, Florida: CRC Press, 1999.
- [3] M. L. Claypool and J. Tanner, "The Effects of Jitter on the Perceptual Quality of Video", ACM Multimedia Conference, October 1999.
- [4] G. Miller, G. Baber, and G. Gilliland, "News on-Demand for Multimedia Networks", In Proceedings of ACM Multimedia '93, Anaheim, CA, pages 383-392, August 1993.
- [5] D. Ferrari and D. Verma, "A scheme for real-time channel establishment in wide-area networks", IEEE Journal on Selected Areas in Communications, 8(3): 368–379, April 1990.
- [6] D. Clark, S. Shenker, and L. Zhang, "Supporting real-time applications in an integrated services packet network: Architecture and mechanism", In Proceedings of ACM SIGCOMM'92, pages 14–26, Baltimore, Maryland, August 1992.
- [7] D. Ferrari, "Client requirements for real-time communication services", IEEE Communications Magazine, 28(11): 65–72, November 1990.
- [8] H. Vin, P. Goyal, A. Goyal, "A statistical admission control algorithm for multimedia servers", Proceedings of the second ACM international conference on

Multimedia, p.33-40, October 15-20, 1994, San Francisco, California, United States.

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

[10] J. Gemmell and S. Christodoulakis, "Principles of Delay Sensitive Multimedia Data Storage and Retrieval", *ACM Transactions on Information Systems*, 10(1): 51-90, 1992.

[11] P. Venkat Rangan and Harrick M. Vin, "Designing File Systems for Digital Video and Audio", In *Proceedings of the 13th Symposium on Operating Systems Principles (SOSP'91)*, *Operating Systems Review*, Vol. 25, No. 5, pages 81-94, October 1991.

[12] F.A. Tobagi, J. Pang, R. Baird, and M. Gang, "Streaming RAID: A Disk Storage System for Video and Audio Files", In *Proceedings of ACM Multimedia'93*, Anaheim, CA, pages 393-400, August 1993.

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

[14] P. Yu, M.S. Chen, and D.D. Kandlur, "Design and Analysis of a Grouped-Sweeping Scheme for Multimedia Storage Management", *Proceedings of Third International Workshop on Network and Operating System Support for Digital Audio and Video*, San Diego, pages 38-49, November 1992.

[15] S. Childs, "Portable and Adaptive Specification of Disk Bandwidth Quality of Service", *Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '99)*, June 1999.

- [16] H. M. Vin, A. Goyal, A. Goyal, and P. Goyal, "An Observation-Based Admission Control Algorithm for Multimedia Servers", in *Proceedings of the First IEEE Inter-135 national Conference on Multimedia Computing and Systems (ICMCS'94)*, (Boston), pp. 234–243, May 1994.
- [17] P. Mohapatra and X. Jiang, "An Aggressive Admission Control Scheme for Multimedia Servers", in *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pp. 620–621, 1997.
- [18] S. Jamin, S. Shenker, L. Zhang, and D.D. Clark, "An Admission Control Algorithm for Predictive Real-Time Service (extended abstract)", In *Proceedings of Third International Workshop on Network and Operating Systems Support for Digital Audio and Video*, San Diego, CA, pages 308–315, November 1992.
- [19] S. Plotkin, B. Awerbuch, Y. Azar, "Throughput Competitive On-line Routing", *34<sup>th</sup> Annual Symposium on Foundations Computer Science*, Los Alamitos, CA, Nov 1993.
- [20] Borodin & R. El-Yaniv, "*Online Computation & Competitive Analysis.*" Cambridge University Press, 1998.
- [21] B. Teitelbaum, "Future Priorities for Internet2 QoS", *Working Group: Papers*, <http://www.internet2.edu/qos/wg/documents.shtml>, October 2, 2001.
- [22] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services" *RFC 2475*, December 1998.
- [23] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP) Version 1 Functional Specification", *RFC 2205*, September 1997, Proposed Standard.

- [24] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus, "Requirements for Traffic Engineering over MPLS", *Draft-ietf-mpls-traffic-eng-01.txt*, June 1999.
- [25] M. Gerla, C. Casetti, S. S. Lee, and G. Reali, "Resource Allocation and Admission Control Styles in QoS DiffServ Networks", *QoS-IP 2001*, Rome, Italy, Jan 2001.
- [26] R. J. Gibbens and F. P. Kelly, "Distributed Connection Acceptance Control for a Connectionless Network", *ITC16*, Edinburgh, 1999.
- [27] F. Kelly, P. Key and S. Zachary, "Distributed Admission Control", *IEEE Journal on Selected Areas in Communications*, pp. 2617-2628, Volume 18, 2000.
- [28] T. Kelly, "An ECN Probe-Based Connection Acceptance Control", *Computer Communication Review*, Volume 31(3), July 2001.
- [29] L. Breslau, E. Knightly, S. Shenker, I. Stoica, H. Zhang, "Endpoint Admission Control: Architectural Issues and Performance", *SIGCOMM 2000*.
- [30] O. C. Imer, S. Compans, T. Basar and R. Srikant, "ABR Congestion Control in ATM Networks", *IEEE Control Systems Magazine*, Feb 2001.
- [31] A. Greenberg, R. Srikant and W. Whitt, "Resource Sharing for Book-ahead and Instantaneous-request Calls", *IEEE/ACM Transactions on Networking*, pp 10-22, Volume 7(1), Feb 1999.
- [32] R. Srikant and W. Whitt, "Resource Sharing with Book-Ahead and Instantaneous-Request Calls Using a CLT Approximation", *Telecommunication Systems*. pp. 235-255, Volume 16(3), March/April 2001.
- [33] A. Dasylva and R. Srikant, "Optimal WDM Schedules for Optical Star Networks", *IEEE/ACM Transactions on Networking*, pp 446-456, June 1999.

- [34] A. S. Tanenbaum, "*Computer Networks*", pages 147-155, Prentice Hall, 1996.
- [35] C. Courcoubetis, G. Kesidis, A. Ridder, J. Wairand and R. Weber, "Admission Control and Routing in ATM Networks using Inferences from Measured Buffer Occupancy", *IEEE Transaction on Communication*, Feb./March/April 1995.
- [36] J. K. MacKie-Mason, H. R. Varian. Pricing, "Congestible Network Resources", *IEEE J. Selected Areas Comm*. Vol 13, pp 1141-1149, 1995.
- [37] R. Singh, M. Yuksel, S. Kalyanaraman and T. Ravichandran, "A comparative evaluation of Internet Pricing models: Smart market and Dynamic Capacity Contracting", *Workshop on Information Technologies and Systems (WITS)*. Queensland, Australia, 2000.
- [38] S. Khan, "Quality Adaptation in a Multi-Session Adaptive Multimedia System: Model and Architecture", PhD Dissertation, Department of Electrical and Computer Engineering, University of Victoria, 1998.
- [39] G. Campbell, D. Coulson, "A Quality of Service Architecture", *ACM Operating Systems Review*, Volume 24. April 1994.
- [40] L. Chen, S. Khan, K. F. Li and E. Manning, "Building an Adaptive Multimedia System using the Utility Model", *International Workshop on Parallel and Distributed Realtime Systems*, San Juan, Puerto Rico, April, 1999.
- [41] K. Kawachiya and H. Tokuda, "QOS-Ticket: A New Resource-Management Mechanism for Dynamic QOS Control of Multimedia," *Proceedings of Multimedia Japan 1996*, pp 14-21, April 1996.

- [42] R. K. Watson, "Applying the Utility Model to IP Networks: Optimal Admission & Upgrade of Service Level Agreements", MASC Thesis, Dept of ECE, University of Victoria, April 2001.
- [43] N. Nishio and H. Tokuda, "QoS Translation and Session Coordination Techniques for Multimedia Systems", *Sixth International Workshop on Network and Operating System Support for Digital Audio and Video*, Jushi, Japan, April 23—26, 1996.
- [44] L. C. Schreier, B. Davis, "System-Level Resource Management for Network-based Multimedia Applications", *Fifth International Workshop on Network and Operating System Support for Digital Audio and Video*, NOSSDAV 95, Durham, NH, 1995.
- [45] S. Chatterjee, J. Sydir and B. Sabata and T Lawrence, "Modeling Applications for Adaptive QoS – based Resource Management", *2nd IEEE High Assurance Systems Engineering Workshop*, August, 1997.
- [46] M. Moser, "Declarative Scheduling for Optimally Graceful QoS Degradation", *IEEE Multimedia Systems*, June 1996, Hiroshima, Japan.
- [47] M. Moser, D. P. Jokanovic and N. Shiratori, "An Algorithm for the Multidimensional Multiple-Choice Knapsack Problem", *IEICE Transactions on Fundamentals of Electronics*, pp 582-589, Volume 80(3), 1997.
- [48] N. Venkatasubramanian, K. Nahrstedt, "An Integrated Metric for Video QoS", *Fifth ACM International Multimedia Conference*, Seattle, USA.
- [49] C. Lee, "On QoS Management", PhD Dissertation. School of Computer Science, Carnegie Mellon University, August 1999.
- [50] Lee J.Y.B. "Parallel Vidco Servers: A Tutorial", *IEEE Multimedia*, 5, no. 2, 20–28, 1998.

- [51] Patterson D.A., Gibson G. and Katz R.H., "A Case for Redundant Arrays of Inexpensive Disks (RAID)", In Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, 109-116, 1988.
- [52] Bolosky W.J., Fitzgerald R.P. and Douceur J.R., "Distributed Schedule Management in the Tiger Video Fileserver", In Proceedings of the Sixteenth ACM Symposium on Operating System Principles, Saint-Malo, France, 212-223, 1997.
- [53] Wong P.C. and Lee Y.B., "Redundant Arrays of Inexpensive Servers (RAIS) for On-Demand Multimedia Services", In Proceedings ICC 97, Monr'eal, Qu'ebec, Canada, 787-792, 1997.
- [54] Jonathan Dukes and Jeremy Jones, "Dynamic Replication of Content in the HammerHead Multimedia Server", In Proceedings of EUROMEDIA 2003, Plymouth, UK.
- [55] Roger L. Haskin, "Tiger Shark - a scalable file system for multimedia", IBM Journal of Research and Development, Volume 42, Number 2, March 1998, pp. 185-197.
- [56] C. Mohan, Inderpal Narang, "Recovery and Coherency-Control Protocols for Fast Intersystem Page Transfer and Fine-Granularity Locking in a Shared Disks Transaction Environment", VLDB 1991: 193-207.
- [57] C. Wu and R. Burns, "Handling Heterogeneity in Shared-Disk File Systems", Proceedings of the ACM/IEEE SC2003 Conference, November 15 - 21, 2003, Phoenix, Arizona.
- [58] A. S. Tanenbaum, "Modern Operating Systems", pages 217-220, Prentice Hall, 2000.



Bibliography

[59] P. Lougher and D. Shepherd, "The Design of a Storage Service for Continuous Media", the Computer Journal, 36(1): 32-42, 1993.

[60] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, "Introduction to Algorithms", pages 969-973, Prentice Hall, 1998.

