Universitat Autònoma de Barcelona

Admission Control and Media Delivery Subsystems for Video on Demand Proxy Server

Computer Science Department
Unit of Computer Architecture and Operating Systems

A thesis submitted by Bahjat Mohammad Khaleel Qazzaz In fulfilment of the requirements for the degree of Doctor by the University Autónoma of Barcelona.

Barcelona (Spain), April 2004

Admission Control and Media Delivery Subsystems for Video on Demand Proxy Server

Thesis submitted by Bahjat Mohammad Khaleel Qazzaz In fulfilment of the requirements for the degree of Doctor by the University Autonomy of Barcelona. This work has been developed in the Computer Science Department of the University Autónoma of Barcelona and was supervised by Dr. Remo Suppi Boldrito.

Barcelona (Spain), April 2004

Thesis Advisor

Remo Suppi Boldrito

اليهم........

وقل ربي زدني علما *

*
"… and say, O my Lord! Advance me in knowledge…"
Quran, Sura Ta,Ha 20:114

Acknowledgements

In my case, I have so many people to thank for the completion of this dissertation. Some of them are in the occupied Palestine and many of them are in Spain.

I am grateful to my advisor, Dr. Remo Suppi for motivating and guiding my research and especially for his rescue ideas each time I was close to failure. I am thankful to the time he devoted to my work and to his insistence and willingness to make this work fruitful for me and for my people as well as for our department.

I would like to thank Dr. Emilio Luque for giving the help that I needed since I came to Spain and for standing by me in all situations. It was really joyful and fruitful to discuss and share ideas with him at all levels. He is a nice person to talk with and to join in travelling.

My thanks go to Dr. Ana Ripoll who was always offering me her help with a nice smile. Also, my thanks go to Porfidio Hernandez for the support he gave to me since I came to Spain.

Special thanks go to Dr. Lola Rexachs for the persistent encouragement she presented to me and for standing by me and my family during my stay in Spain.

I am grateful to the UAB for granting me a full scholarship and to the PEACE program (Palestinian European and American Cooperation in Education) for partially supporting my stay in Spain. Special thanks go to all those who made my travel to Spain possible, Dr. Munzir Salah and Dr. Rami Alhamdallah of An-najah University, Dr. Gabi Baramkeh director of the PEACE program, Dr. Luis Lemkowv the Vice president of the International Affairs at UAB, Señora Rosa Maria Vila for preparing all means of my stay in Spain.

I would like also to thank my colleagues in the Computer Science Department at An-najah University in Palestine Dr. Nizar Awartani, Dr. Wa'el Mustafa, Dr. Lu'ai

Malhis, Husam AbdulHalim, Mut’asim Abu Zanat, I thank them for encouraging and supporting me during my study and for taking the responsibility of my academic load when I left for Spain.

I am also grateful to Dr. Dani Franco, Diego Mostaccio, and Josep Jorba for taking the responsibility of my academic load while I was finishing this work.

Thanks go to my colleagues from the Computer Architecture and Operating Systems Group, especially Gemma Roque, as well as to those who have passed through the Group during the preparation of this work for their constant support and encouragement.

I would like to thank my parents who blessed my travelled to Spain and encouraged me all the time.

And finally, I am so grateful to my wife and my daughters for the patience and the suffer they had to bear in order to make this work possible.

Contents

Figures

Tables

Preface

The recent advances and development of inexpensive computers and high speed networking technology have enabled the Video on Demand (VoD) application to connect to shared-computing servers, replacing the traditional computing environments where each application was having its own dedicated computing hardware. The VoD application enables the viewer to select, from a list of video files, his favorite video file and watch its reproduction at will.

A successful VoD application must provide long-lived video streams (e.g. 7200 seconds) which consume high resources such as I/O and network bandwidth to a large number of clients. Therefore, a video server must secure the necessary resources for each stream during the video reproduction so that the clients don’t witness starvation in their buffers.

Early video on demand applications were based on single video server where video streams are initiated from a single server, then with the increase in the number of the clients who became interested in VoD services, the focus became on Distributed VoD architectures (DVoD) where the VoD service is distributed among a set of distributed VoD servers called proxy servers. However, whether the VoD service is presented by a single or multiple video servers, a key stone in the success of the VoD systems is the design of a single video server (proxy server) that can be a building block in the construction of DVoD architecture.

The VoD server must handle several issues in order to be able to present a successful service. It has to receive the clients’ requests and analyze them, calculate the necessary resources for each request, and decide whether a request can be admitted or not. Once the request is admitted, the server must schedule the request and send the video data in a timely manner so that the client does not suffer data starvation in his buffer during the video reproduction. In addition, the server must be able to provide true video on demand (TVoD) service, by which the user can interact with the server and invoke VCR functions and receive the response in a short period of time.

The type of data transmission determines the type of the VoD service. During the last 15 years the VoD service has been delivered mainly in two ways; one-to-one and one-to-many. In one-to-one policy, each client is assigned a channel for receiving the video data so that he can interact with the server and invoke

VCR-Like functions at will. However, the server runs out of resources after accepting a small number of clients. On the other hand, in the one-to-many policy, the server broadcasts the video stream allowing many users to view the video data causing better use of the system resources and high performance. However, the client in this type of service is almost passive and can't interact with server freely.

This thesis presents the design and implementation of a video proxy server (VPS) that provides scalable interactive video on demand service yet it maintains the performance that can be gained from broadcast transmission. The VPS consists of three main parts. The first part is the Admission Control Module (ACM) which receives the clients' requests, negotiates the required resources, and decides whether to accept or reject a client based on the available resources. The second part is the Resources Management Module (RMM) which manages several shared resources, such as the CPU, Memory, Network, and the Disk, and reserves them. These resources are managed by four brokers: the CPU broker, the Memory broker, the Network broker, and the Disk broker.

The third part is the CB_MDA algorithm which controls the flow of the data by regulating the resources assignment and scheduling the video streams. The CB_MDA provides TVoD service and VCR-functions without the need to reserve dedicated resources (such as the bandwidth) such these which were proposed in the literatures. Rather, it uses all the available resources in order to send work-ahead video data with the intention of reducing the service time and recovering these resources whenever they are needed. The CB_MDA uses a hyper technique which combines multicast and unicast channels for transmitting the video data. The multicast streams transmit a video file from the beginning while the unicast channels are used to join the later arrivals to the appropriate multicast stream.

To validate our work, we have studied and conducted exhaustive experiments in order to validate the CB_MDA within the design of the VPS. In addition, we have created real experimental platform which can be used to conduct further experiments with different parameters.

The thesis further goes beyond the design of the VPS and presents a video client architecture that can synchronize with the server and work as a plug-in for reproducing the video data by different players.

The thesis organization

The first chapter gives an overview about VoD systems and their architectures (single and Distributed VoD). Also, it highlights the main components that should exist in a VoD server such as the admission control (AC) subsystem, the resources management subsystem using enhancement of the OS kernel and/or creating a middleware approach, and the storage subsystem. In addition, it explains the types of the video transmission followed by the main scheduling policies used for guaranteeing proper video on demand service. Then, it mentions the main network protocols that can be used for the communication and video transmission.

At the end, chapter one makes conclusions from what has been mentioned before and makes a summary of the main objectives of the thesis followed by the main contributions.

- Chapter two concentrates on the design and implementation of our video on demand server. The main objectives are to design a VoD server that can work as a video proxy server in DVoD systems, to provide a scalable VoD server, to allow users to interact with the server, to provide true video on demand, to guarantee continuous video reproduction without jittering, and to support instantaneous VCR-like functions. Therefore, the chapter shows the architecture of our video proxy server (VPS) and shows how the main components are laid and connected to each other. It describes the functionality of the ACM (admission control module and the RMM (resources management module). In addition, it presents an analysis for four admission policies which have been implemented by any of these brokers. These policies are the Maximum Policy, the Adaptive Maximum Policy, the Average Policy, and the Adaptive Average Policy.
- Chapter three focuses on the CB_MDA which is responsible for controlling the flow of the video data from the server to the client. It describes how the CB_MDA guarantees the QoS of the clients being in service, and it shows how the algorithm prohibits data starvation, overflowing the bandwidth, and overflowing the clients' buffers. Also, chapter three shows the working principle of the algorithm CB_MDA which adapts dynamically to the changes during the video reproduction. The CB_MDA takes into considerations that the clients can be heterogeneous in terms of the capacity of their machines specifications, and it uses a combination of multicast and unicast channels for data transmission. The multicast streams transmit a video file from the beginning while the unicast channels are used to join the later arrivals to the appropriate multicast stream. Also, it tries to create work-ahead video data in the appropriate clients' buffers in order to reduce the service time by implementing pre-fetching and patching techniques.

- Chapter four presents and demonstrates the experiments which have been carried out to verify the viability of the server components with the focus on the CB_MDA. It discusses the results obtained from the admission policies which have been implemented in the resources brokers. The results show the difference between the admission policies and show that, with these policies, the brokers are able to reserve the necessary resources and to block any request that might affect the QoS of other clients being in service. In addition, chapter 4 presents a detailed discussion and analysis to the results obtained from experiments carried out with the CB_MDA. The results show that the algorithm is capable of increasing the system performance by joining the later arrivals to appropriate multicast channels so that a group of clients can view the same video data yet none of them lose the initial part of the video thus more clients can be served by the server. Also, the results show that the CB_MDA reduces the service time especially of the short unicast channels which are important for providing the vide prefix and the VCR-like functions.
- Chapter five summarizes the main conclusions driven from the experiments and outlines the future work. In addition, it lists the publications produced from this work and a list of final projects made based on this work.
- Finally, a complete bibliography is provided in addition to two appendices A and B which are related to the video scheduling algorithms and the client architecture respectively.

Chapter 1
Introduction

1.1. Motivation

Video on Demand (VoD) applications, which deliver digitized video to users, are gaining increasing interest and becoming widely-used nowadays due to the recent advances in computing and communication technology coupled with advances in computer, storage, and display technology. They allow remote users to select from a large collection of videos and to playback any of them at any time (Figure 1.1).

Within the applications field (e.g. movie-on demand, distance learning, interactive news), these are expected to provide continuous media distribution to a high number of clients, distributed geographically and with different access speeds to the interconnection systems.

The 1990’s have witnessed an increase in the number of Personal Computers (PCs) in homes and offices, and an increase in the performance of the PCs accompanied by reduction in their prices. This has led to a tremendous increase in the number of computers which have become a part of one global network called the Internet [VEN96]. The Internet, then, made it possible to create a wide spectrum of VoD applications. Some multimedia applications categorized under video on demand are: *Movies on Demand*, *Interactive Video Games*, *Interactive News…*etc (Table 1.1).

This variety in VoD services causes one to speculate that VoD systems will become one of the most important services supported by the next generation of computer networks, video servers and distributed multimedia systems [PAD99]. It is also expected that movies on demand applications will soon replace the traditional video rental stores [TAN03].

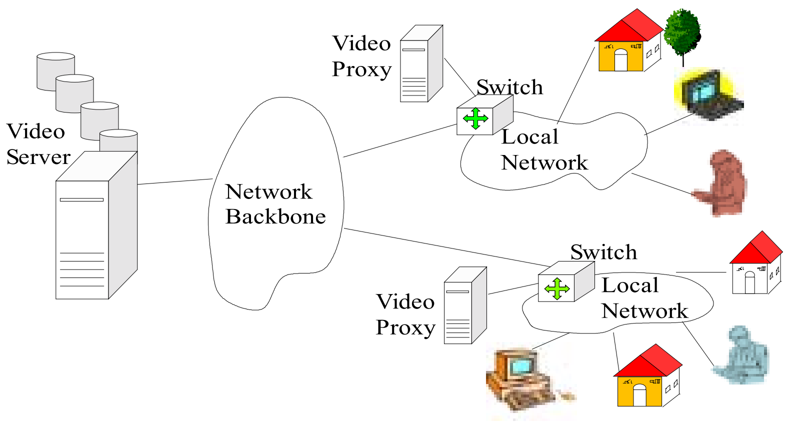


Figure 1.1. Video on demand architecture

Table 1.1. On demand multimedia applications

Application	Description
Movies on Demand	Users are allowed to select a movie and decide when to play it and invoke VCR-like functions at will.
Interactive Video Games	Users can download games to their machines without the need for purchasing them.
Interactive News	Users can view news from selected top stories.
Interactive Advertising	Users can check and purchase commercial products.
Distance Learning	Students can register classes and view classes materials
Catalogues Browsing	Users checks the latest available products
Video Conferencing	Users can negotiate with each other
Tele-Shopping	Users can choose from a list of goods and order their items.

In a report called *Present and Future by In-Stat/MDR* [WPG01] expectations indicate that by the end of 2004, the number of consumers using family oriented on demand IP services will out number the users of adult content services. Also, the report shows that by the end of 2006, family oriented on demand services will overtake the adult content sites in terms of annual revenue. Therefore, as consumer-oriented VoD services over IP become more pervasive, revenue generated by family-oriented VoD services will eventually surpass those of adult content sites.

In the report, the author speculates that by the end of 2006, 40% of worldwide consumers who have high-speed Internet connections to their residences will be using on demand services for which they pay monthly fees. Table 1.2 shows the participation and the share of the countries worldwide in providing VoD services. It indicates that, although Europe is occupying the middle of the world, it needs to spend more in this area of services so that it can compete with other countries.

Table 1.2. Percentage of VoD participants

Countries	%
United States	43.3
Asia, especially South Korea, Taiwan, Singapore and others	37.0
Europe	15.0
Rest-of-the-world	4.7

These expectations have encouraged researchers world-wide to investigate and spend in this area with the hope of providing different varieties of VoD applications. They have concentrated on the main issues that affect the performance of VoD systems in one way or another.

Such issues include; video on demand services and their characteristics, VoD servers architectures and their internal design, video storage management, video data transmission techniques, and scheduling policies which are responsible for admission control and video data delivery.

In addition, the client design and architecture have received great consideration since the video will be played at the client's machine were the VoD service will be evaluated.

1.2. Video on Demand Characteristics

The design of such multimedia services differs significantly from traditional text/numeric retrieval services since the playback of digital video and audio consumes data at a very high rate [GEM95]. Also, video data needs to be played back continuously in a timely manner in order to preserve its meaning, while text data does not have temporal constraint. Therefore, when designing VoD systems, one should take into consideration the VoD service characteristics such as *Long-Lived Session*, *High Bandwidth Requirements*, *Support for VCR-Like Interactivity*, and *Quality of Service* [HUA02].

- ***Long-Lived Session***: a VoD system should take into consideration the support of long-lived sessions during the transmission from the server to the client's playback machine. This is required since some video files require long time of playback. For example, a typical movie-on-demand service might last 90-120 minutes in servicing a single movie. That is, if the video data is sent at a rate equal to real-time play back rate then the server's resources which are needed to maintain this session must be reserved for 90-120 minutes.
- ***High Bandwidth Requirements***: video data transmission requires high I/O and network bandwidth. For example, the server storage I/O and network bandwidth requirements for a MPEG-1 stream are 1.5 Mbps (mega bits per second). Another example is the MPEG-2 standard which specifies a higher compressed bit streams for high-quality digital video at a rate of 2-80 *Mbps*. Thus, a two-hour MPEG-2 video with a resolution 720x486 and bit rate 4.5 Mbps requires about 4 GB of storage. Likewise, MPEG-4, NTSC, and HDTV formats also require high bandwidth. Sending such video data from the server to the client without interruption is a great challenge.
- ***Support for VCR-Like Interactivity***: a client expect the VoD system to offer VCR-like functions, such as the ability to play, forward, reverse and pause. This requirement will increase the cost of the service since it implicates that each user might be assigned a dedicated session in order to give him the total freedom to invoke the VCR functions at will.
- ***Quality of Services (QoS)***: the QoS, that VoD consumers and service providers might be concerned about, includes service latency, defection rate, interactivity, playback effects of videos…etc. Therefore, some mechanisms must be implemented to achieve the best QoS. The acceptance of a new user must not affect the QoS provided to the users being serviced, and adequate

resources must be available throughout the entire system from the server to the user during the playtime [MUN99][REI00].

These are general characteristics which play an important role in the design of video on demand services. However, the kind of application designed might relax some of these characteristics. For example, running video clips does not require as long a time session as running typical movies. Also, in some VoD applications, e.g. in advertisement, the interactivity might not be a requirement and therefore does not need to be applied.

VoD services enable clients to select a video program, often a movie like you would get from the video rental store, and have it sent to them, in a form called a *stream*, over a channel via a network such as a cable or satellite TV network. This way, the clients are imitating television viewers. However, the VoD clients can go beyond the typical television viewers since they will be able to interact with the service and invoke VCR-like functions such as; pause, jump forward, jump back, stop, and so forth like they would if it was running on their own VCR or DVD devices.

Different from the VoD services, television viewers are passive and can’t interact with the service provider and have no control over what they watch, since the service provider is the one who chooses what to broadcast. Thus, VoD applications, in addition to the flexibility in the freedom of choice, have added the interactivity on the television-like programs and introduced what is called Interactive Video on Demand (IVoD) which contributed to the wide increases in the popularity of VoD applications.

1.3. Interactive Video on Demand

Interactive Video on Demand (IVoD) is an extension of video on demand application where the user can negotiate with the service in order to achieve his requirements such as deciding what to watch and when to watch a specific video. In addition, the user would be able (depending on the system implementation) to invoke VCR-Like functions such as the ones which have been previously mentioned.

1.3.1. IVoD Components

An IVoD system consists of three major parts:

- The client with a graphical user interface which allows him to communicate and negotiate with the server and visualize the video.
- The network over which the video data will be delivered, and
- Servers with archives of movies (or whatever the server is offering to show). The clients machines are expected to have a storage medium to buffer parts of the video data.

1.3.2. IVoD Functions

The IVoD system usually provides the same VCR functions that you would see in your VCR system. However, the IVoD applications can go beyond the VCR and provide additional functions that might not have been implemented in the VCR system. The interactive functions that the IVoD applications might provide include:

- ***Play***: start the movie presentation (usually from the beginning of the video file or resume from the stopping point).
- ***Pause***: freeze the movie playback. With this command the user usually will watch a still picture on the displaying machine.
- ***Stop***: stop the movie presentation. Stop function works as double functions in VCR. It stops the playback completely and cause rewind. That is, if the client tries to play the movie again then his request would be considered as a new none active request.
- ***Jump forward***: jump to a particular scene (point) of the movie in a forward direction. This function can be satisfied by two ways; first, the video data from the skew point onward can be available in the user buffer so the request is achieved at the user end; second, the video data is not available at his buffer and the request is satisfied by having the server delivering the video data from the skew point.
- ***Jump backward***: jump to a particular point in the movie in a backward direction. As in jump forward, if the requested video scene is available in the client's buffer then this function will be handled at the user end. Otherwise, the server has to intervene in order to deliver the requested video data.
- ***Fast Forward***: fast forward is different from jump forward in that it browses through the movie in the forward direction with picture and sound on. In fact, such a service is not recommended since it consumes a

lot of resources such as bandwidth. If the user is interested in such a service in order to know where to start from, an alternative way can be implemented. For example, clips or scene from the movie can be browsed so that the user can determine where to start from.

- ***Backward***: backward allows the user to browse through the movie in the backward direction with picture and sound on. As in fast forward, this function is not recommended since it might cause a serious degradation in the system performance. This happens when the user does not have enough buffer to keep storing the already played data and/or the policy of the service does not allow storing the video data on the client's disk. Therefore, if the reviewed data dose not exist in the client's buffer or local disk then the backward command must be satisfied by the server. This indicates that the server must dedicate resources to this command.
- ***Slow Down***: going forward at a lower rate than normal but with picture and sound.
- ***Slow Reverse***: go backward at a slower speed, with picture and sound.

Although the clients will be pleased to enjoy all these functions, their implementation is left to the choice of the designers. That is, it is not necessarily that all IVoD systems are obliged to include these functions or the same functions.

In fact, some functions can be excluded such as fast forward and backward and replaced by jump forward and jump backward respectively, and new interactive functions can be included such as the ability to avoid or select advertisements during the video program downloading, to investigate additional details about news events and to browse, select, and purchase goods. In addition, IVoD might provide the possibility of storing some images and/or parts of the movie.

1.4. Video on Demand Service

From the business point of view, the VoD services delivery tends to be provided in three forms [WPG02]:

- Free VoD services where the users can download free video clips such as movies clips, advertisement, and news.

- Subscription VoD services, where users are obliged to pay a monthly or annual fee so that they can access the service (e.g. Showtime On Demand and HBO On Demand) [WPG02].
- Pay-Per-View VoD services, in these services the movies are ordered at will and, the users don't have to make monthly subscription and pay only for what they watch.

Most likely, the customer would be interested in free delivery of movies on demand service. This service fits perfectly in assumable none profitable organizations and educational institutions such as universities and schools. The users can use an IP address to connect to the video server and browse a list of movies from which they can select their favorite one.

From the server point of view, the video file can be downloaded in its entirety and stored at the client's side before it can be played. This action would have the following benefits:

- Allow the end-user to watch the whole movie from their hard drive. Thus, once the movie is downloaded the user can decide when to play it.
- Allow the user to invoke the VCR-Like functions at will. This would give him the ability to pause, do other activities and resume.
- Eliminate exchanging messages produced between the client and the user, as a result of invoking interactive functions by the user.
- Eliminate the user trips to the video rental store.

However, downloading the whole video file and storing it on the client's hard drive before it can be played back would definitely have some drawbacks on the server side as well as the client side. These drawbacks are stated as follow:

- The client must have a huge hard drive since the compression bit rate of multimedia data is high (Table 1.3). In the past, such solution was not acceptable since the storage was costly. Nowadays, this can be applicable to some home PCs, but not all the clients have this capability. For example a two-hour MPEG-2 movie with a bit rate of 4.5 MB/s would require 32.4 Giga Byte (GB).
- The response time will be high. That is, the client must wait long before he can watch the movie. For example, downloading a two-hour MPEG-2 movie with a bit rate 4.5 MB/s over 10Mb/s network would require 432 minutes. That is, the user must wait 7.2 hours before he can watch the movie on his playing screen.

Table 1.3. Examples of media compression formats

Media type (specifications)	Bit Rate
Voice quality audio (1 channel, 8 bit samples at 8kHz)	64 Kb/s
MPEG encoded audio (equivalent to CD quality)	384 Kb/s
CD quality audio (2 channels, 16 bit samples at 44.1 kHz)	1.4 Mb/s
MPEG-1 encoded video	512 Kb/s – 1.5 Mb/s
MPEG-2 encoded video	1.5 – 5 Mb/s
MPEG-4 encoded video	40 Kb/s – 1 Mb/s
NTSC quality video	27 MB/s
HDTV quality video (1280 X 720 pixels/frame, 24 bits/pixel)	81 MB/s

- The server performance in term of the number of users connected to it is degraded since most of the user would turn their back and cancel their request if they don't get the service in an acceptable short response time.

An alternative way to downloading and storing the whole video file at the client buffer is to stream the video data to the client and start the playback right after the arrival of the video data to the player buffer. This would be done by delivering the video data over a network in real time to the user's set-top box or PC, and store part of the video in the client's machine. At the same time, the server must be able to discover that it has the capability to stream the video data at a rate greater than the real time rate and find out the clients who can benefit from the available resources and send them video data at a rate greater than the video rate consumption.

Also, the server should be able to notice that there are new arrivals who request the same movie file at the same time or within a short interval of time between them. In this case, the server must be able to serve all of them as if they were one request. This way, the number of clients served by the server is increased. These issues are discussed further in chapter three where the scheduling algorithms for media delivery are discussed.

1.5. Servers Architectures

For VoD systems to be successful and for their applications to compete successfully with video rental stores, it is necessary to design video servers and media data transmission mechanism which achieve the following objectives:

- Servicing a large number of users
- Permitting users to decide when, where and what to watch
- Providing short response time to the different users' requests.
- Guaranteeing a continuous media without jittering during the playback of the video.
- Supporting instantaneous interactive VCR-Like functions such as Stop, Play, Pause, Jump forward, and Jump backward without delay.

Solutions to VoD applications have been proposed via two types of severs called single server (usually termed as centralized video server) and multiple servers (decentralized servers or distributed servers DVoD) [REI98].

1.5.1 Single Video Server

The basic model of a single video server is illustrated in Figure 1.2. In this model, the video streams are transmitted from a single server, and each user is assigned a stream. The disadvantage of such a model is that, its cost and technical characteristics make it unfeasible to offer acceptable VoD services. Also, this kind of server can't meet the increasing number of clients who are becoming interested in VoD services.

The most important problems that can be noted in single video server architectures are stated as follows:

- Higher cost,
- Non-scalability and
- The inefficiency in the utilization of the available media.

However, depending on the objective of the design, this kind of architecture can be used at small scale as follows:

- It fits well in small institutions where the number of clients is small.

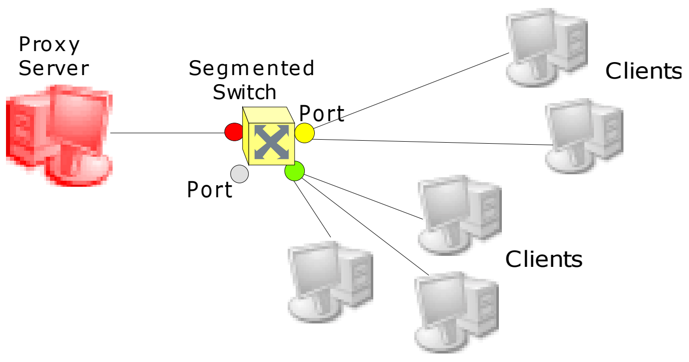


Figure 1.2. Single VoD architecture

- Also, it can be enhanced so that it works as a building block for designing a larger video system where multiple single servers work together to provide VoD services to a large number of users such as the case in DVoD systems.

1.5.2 Multiple Video Servers

Another possible solution would be the distribution of the service in order to handle distributed clients. Figure 1.3 shows multiple local video servers (termed as proxy servers) which feed sets of clients connected to them via local network. Systems based on this approach are called Distributed Video on Demand (DVoD) and it can be demonstrated that with a minimal cost, a solution for distributed continuous media streaming applications is possible [BAR96].

The Distributed VoD systems allow distributing the traffic on the network in order to reduce the bandwidth requirements. The objective of this architecture is attained by distributing the clients and a set of video proxy servers through the network topology, avoiding traffic concentration in the inferior levels of the hierarchy.

A DVoD system requires the arrangement of those servers that offer the video retrieval and playback services in a distributed system, in order to support a large number of concurrent streams. In the literature, arrangements are classified as follows:

- ♦ The use of independent servers, this approach is based on replicating VoD servers close to clients' networks so that these users do not need to access the main server [CHA99].

- One-level proxies, this type of DVoD tries to reduce the size of local servers in such a way that they only store those videos with a higher access frequency; these servers are managed as main-server caches and are called proxies [FAB01].

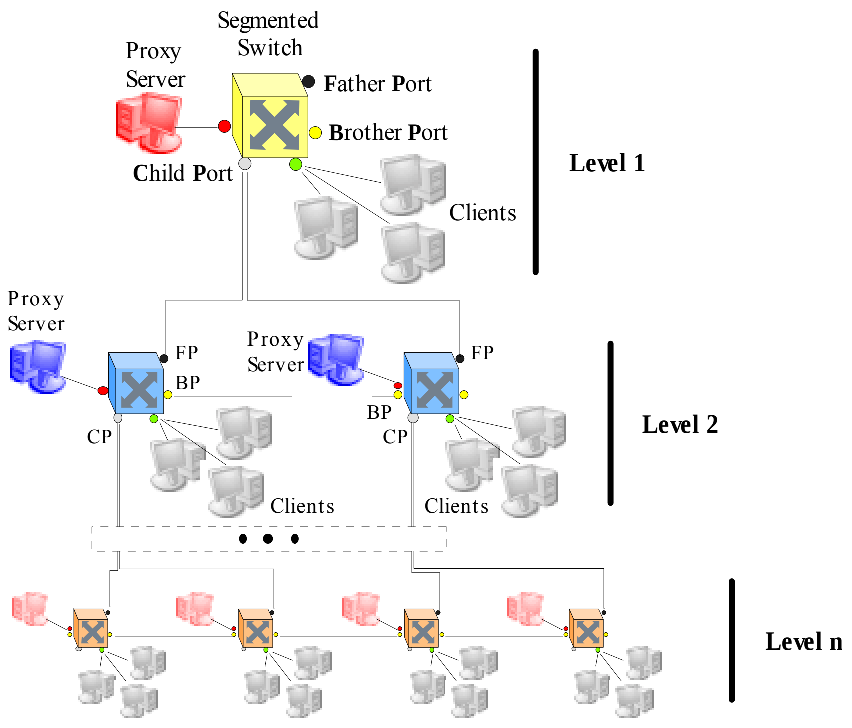


Figure 1.3. Distributed VoD architecture

- Hierarchical distributed systems, hierarchical DVoD systems are based on a network with a hierarchical topology, with individual servers on the nodes and network links on the edge of the hierarchy. The nodes at the leaves of the hierarchy, termed head-ends, are points of access to the system where clients are usually connected [CHA01] [HUA98].

1.6. Single Video Server Architecture

From the previous discussion, it is clear that DVoD system can present a scalable and economical solution to meet the increasing number of the clients who request VoD services. However, for the DVoD to achieve its objective, it is necessary to give great attention to the local video servers (or video proxy serves) which are considered to be directly responsible for delivering the video data to the clients. So, whether the video server architecture is a single video architecture or a local video server that forms part of multiple-video architecture, the internal architecture of these servers must take into consideration some issues that affect the server design and performance.

To understand these issues, we again follow the sequence of making a successful request for a VoD service. In a typical VoD system, the sequence and the actions taken to provide VoD services to a new client are done as follows:

- The client connects to the server using a communication channel
- The server receives the request and analyzes it.
- The server checks to see whether it has enough resource in order to accept the client and to grant him the service.
- If the request is accepted, then the server starts to transmit the video data to the client.

These basic steps give the VoD researchers an idea about the major issues that need to be handled during the VoD design. Some of these issues are related to the server design and others issues are related to the client design. The network, of course, is also an issue that needs to be taken into consideration.

At the server side, the VoD server should have a mechanism for the acquisition of the request and admission control. The admission control must be applied to all resources of the system. These resources include the CPU, Memory, Disk and Network. If the system has enough resources then the server should provide the service. In addition, the server must take into consideration the file system and the way the video data is stored in the system. Also, the server must define an algorithm for scheduling the requests and a mechanism for the transmission of the video data from the server to the client.

On the other hand, the client must have a mechanism such as a GUI from which he will be able to connect to the server, to browse the video files available in the server, to be able to synchronize with the server based on the transmission mechanism, and finally to display the video data.

In the following subsections, the major issues related to the VoD server design are presented. The admission control policies and the resources management system are discussed first since they decide the acceptance or the rejection of the clients. Also the file system and the media storage, where video files are stored, are discussed. Then, the transmission mechanisms are discussed since they decide the type of the VoD service that the server would present to its clients. Thereafter, various scheduling video data delivery algorithms from previous researches are also presented.

1.6.1. Admission Control (AC)

Given the fact that the video data must be continuous, the server must employ an admission control algorithm before accepting new requests. It must ensure first, that adequate resources are available to the new request throughout the entire path from the video server to the client’s presentation device, and second, that the acceptance of a new request does not affect the performance requirements of other clients being already in service

Also, the admission control should be able to negotiate and renegotiate the client’s requirements which will be translated into system resources and quality of service. The client is always looking for the best quality which implicates that the server must meet the performance requirements of the clients.

Prior to admitting a new client for VoD services, the VoD server must have sufficient resources in order to guarantee that the QoS contracted to existing clients is not jeopardized.

To implement admission control, four scenarios can be assumed: *Deterministic Server*, *Statistical Server, Predictive Server*, and *Background (or Best Effort) Server*. These scenarios are explained in more details as follows:

- ***Deterministic***: All deadlines are guaranteed to be met. For this level of service the admission control algorithm considers worst-case scenario in admitting new clients [DEN96]. To implement deterministic service, resources are reserved in worst case manner for each stream admitted. This scenario is used when the client can’t tolerate any deadline violation.
- ***Statistical***: deadlines are guaranteed to be met with a certain probability [VIN94][NER97]. For example, a client may subscribe to a service which guarantees that, 90% of deadlines will be met over an interval. To provide such guarantees, admission control algorithms must consider statistical behaviour of the system while admitting new clients. Implementing statistical service would proceed as with deterministic service, but instead of using worst case values in computing the change to round length, some statistical distributions would be used. For instance, instead of using a worst case rotational delay value, an average value may be used, which can be expected some percent of the time based on a random distribution of rotational delays.

  Providing statistical service guarantees is essential not only due to the variation in the seek time and rotational latency, but also due to the variation in the data transfer requirements of compressed media streams.

To provide statistical service guarantees, a server could employ precise traffic characterizations, rather than the worst-case or the average-case values. It is also possible that when variable rate data is stored, a complete and accurate description of the rate change could be computed, so that the server could use the information during playback to reserve only the required amount of the server resources.

- ***Predictive***: The consumed resources rate is predicted from the history, assuming that the past behavior is an indicator for the future [HOL97]. Thus, the server can predict the resources that will be needed in the future and make the scheduling based on this prediction. Although prediction might give high resource utilization, it provides weak guarantees.
- ***Background(or Best Effort)***: no guarantees are given for meeting deadlines. The server schedules such access only when there is time left over after servicing all guaranteed and statistical clients [GEM95].

1.6.2. Resources Management (RM)

Upon receiving the client's request for VoD service, the request's parameters are extracted in order to calculate the necessary resources in terms of CPU, Memory, Network, and Disk. However, these resources can be shared with traditional best–effort tasks such as background tasks. As a result, these tasks might compete with multimedia applications, such as VoD application, causing resource contention. This resource contention might deprive the VoD application from getting scheduled in a timely manner causing low Quality of Service (QoS) such as, starvation and jitter at the client side. Therefore, there has to be some kind of resource management mechanisms that provide guaranteed video data delivery within specified temporal and spatial constraints. Also, these mechanisms should be able to accommodate changes in resource allocation in order to avoid jittering and starvation.

Operating Systems (OS) such as Linux and Windows were originally designed for best-efforts applications, and they need to be enhanced in order to meet the requirements of multimedia applications which have timeliness constraints. In general there are two ways to solve this problem, they include, *Enhancing the Operating System Kernel* or creating a *Middleware* between the operating system and the application in order to provide QoS to multimedia applications [SHE02]. These approaches are discussed in this thesis.

However, before discussing the two approaches that can be implemented in order to provide resource management to the system resources, the main elements that affect the QoS in multimedia applications are presented first. These elements include the application QoS parameters, the system QoS parameters, and the network QoS parameters.

➢ ***Quality of Service***

Employing resource management mechanisms is very important in multimedia systems in order to provide guaranteed services. The importance comes from the fact that delivering multimedia data over unmanaged resources cause some problems such as jittering or starvation caused by delaying packets delivery. This delay might violate the temporal quality requirements. Therefore, these resources must be controlled so that the QoS required by the users is satisfied. The users usually submit their requirements, which are represented as parameters. These parameters are then translated into resources which will be allocated to the users demands.

To ease the customization of the resources and to allow for flexibility, the International Standard Organization (ISO) has defined a standard for parameterization of the Quality of Service. In general, there are three common types of parameters that should be considered during the design of the resource management [NAH98]. These parameters are:

- *The application QoS parameters*: Describe requirements for the application services possibly specified in terms of,
    1. Media quality which includes the media source/sink characteristics, such as media data unit rate, and their transmission characteristics, such as end-to-end delay.
    2. Media relations that specify the relations among media, such as media conversion, or inter/intra stream synchronization [NAH95].
- *The system QoS parameters*: describe requirements on the communication services and operating system services resulting from application QoS. They are specified in terms of,
    1. Quantitative criteria specifies the parameters that can be evaluated in terms of concrete measures, such as bit rate per second, number of errors, task processing time, data unit size, etc.
    2. Qualitative criteria specifies the expected services needed for provision of QoS, such as interstream synchronization, ordered

delivery of data, error recovery mechanism, scheduling mechanism, etc.

- *The network QoS parameters*: describe requirements on network services, They can be specified in terms of,
    1. Network load, describing requirements on the ongoing network traffic, such as inter-arrival time.
    2. Network performance, describing requirements which the network services have to guarantee, such as latency, bandwidth or jitter (the variance delay across many packets) [FER90].

These parameters can be specified in a deterministic, statistical, or predictive manner. Deterministic and statistical parameters require guaranteed services [FER90], while parameter values estimated from past behaviour requires predictive services [CLA92]. However, if no parameters were specified, best-effort services would be employed.

To understand the nature of the system resources, in [NAH98], they classified resources as *active* or *passive* according to their characteristics. For example, resources such as CPU and network adapter are considered to be active resources since they provide services. On the other hand, the main memory and bandwidth are considered to be passive since they indicate system capabilities required by active resources. Thus, parameters specify requirements on resource capacities allocated to the multimedia system services as well as the service disciplines managing the resources. For example, the end-to-end delay QoS parameter determines the behaviour of transmission services along the path between media source and sink with respect to packet scheduling (bandwidth allocation), queuing (buffer allocation), and task scheduling (CPU processing time allocation) [NAH98].

➢ ***Enhancing of OS Kernel***

The enhancement is done by making changes in the OS kernel. These changes involve adding resource management mechanisms to the OS in order to support multimedia applications and provide explicit QoS guarantees to soft real-time applications. The OS kernel enhancement, of course, has advantages and disadvantages.

The advantages come as a result of having the kernel itself to arbitrate resources among the contending applications. This causes high accuracy and reduces the overheads.

On the other hand, changing the OS kernel requires access to the kernel source code which might not be feasible. Also, a change made in an operating system might not be portable to another OS since the OS architectures vary from one to another.

➢ ***Middleware Layer***

In the middleware approach, the system resources are all controlled by a middleware which works as a layer between the application and the operating system. Thus, requests generated for the system resources by the application go through the middleware. This way, the middleware can arbitrate access to system resources and decide how to allocate resources to various applications providing QoS guarantees to these applications.

As it is the case with OS kernel enhancement, this approach has also advantages and disadvantages. One of the main advantages of the middleware approach is that, it does not require any change in the OS kernel making this approach portable. That is, the middleware can be reused in any operating system. Also, this approach has the advantage of implementing any scheduling algorithm that can provide the best QoS.

On the other hand, the middleware approach has the disadvantage of the inefficiency. That is, since the middleware approach involves creating a new layer between the application and the OS in order to control the resources, the application performance will be reduced as a result of the increase in the runtime overheads. Also, developing a middleware might impose some kind of complexity to the application developer especially if the application needs to be programmed to a new API.

The above discussion shows that there are trade offs between the two approaches. So, the decision is made based on whether to change the OS kernel by creating additional extension or to make all the changes in the middleware. The designers can make their decision based on the application requirements and the system objectives. For example, if the goal is to design a special purpose system, then it could be better to augment the resources management in the OS kernel. Otherwise, middleware mechanism would be much better for general purpose systems where portability is feasible without causing major or any changes. Also, the designer might choose to create a hybrid resources management. That is, some of the enhancements are implemented within the OS kernel and the rest are implemented at the user-level. Table 1.4 present a summary of the different factors that help to make the decision [SHE02].

Table 1.4. Between middleware and OS kernel enhancement

Key points	Enhancing OS Kernel	Use of Middleware
Portability	Not directly portable to another kernel	Portable to different OS
Overheads	Has low overheads	Imposes runtime overheads
OS Design complexity	Imposes a significant challenge on the system designer	The fewer inter-dependencies between the middleware and the kernel reduce the complexity.
Application Design Complexity	Results in low complexity for the application developer	Increases complexity for application developers
Resource management mechanisms	New OS enhancements need to coexist with existing mechanisms making the task of the system designer more complex	Can employ any resource management mechanism to provide QoS support for multimedia applications

1.6.3. Media Storage and File System

Another critical issue in the design of multimedia services is the design of multimedia storage servers that can support continuous retrieval of video data from disks to a large number of clients. Originally, storage servers have been designed to provide efficient access to text and numerical information. This had to be changed as a result of the need to provide multimedia services which involve heterogeneous data types such as text, images, audio, and video. In addition, and as it has been said before, multimedia services require large storage space and large data transfer rate in real-time. Consequently, conventional file systems which were designed for managing textual files do not suffice for managing multimedia objects. Therefore, storage servers must provide mechanisms for storing and retrieving multimedia data in timeliness manner and in large quantities at high speed.

To provide high bandwidth and to support a large number of clients, researchers have focused on replicating video files and distributing these files on disk arrays. To effectively utilize a disk array, video streams have been striped across disks in the array.

Data stripping allows each video stream to be divided into units, called logical blocks or media blocks, which are then distributed among multiple storage nodes. This mechanism achieves high disk bandwidth and maximizes the throughput of a disk array [TEW96], however data stripping creates some problems that need to be solved and, it has some disadvantages [CHO97]:

1. A distributed scheduling mechanism among all nodes of the server is required which imposes clock synchronization problems among all nodes in the server.
2. Relatively, larger start-up latency which refers to the time elapsed since a request is made by the client to initiate a new stream until a fragment of the video stream is received. In no-stripping technique the start-up latency is shorter.
3. Lack of scalability. That is, if more disks or storage nodes are added, the whole data must be redistributed among the disks causing unacceptable overhead and cost.
4. The popularity of movies cannot be considered since replicating popular movies in a server fails to increase the number of clients that can be serviced simultaneously [LIT93].

To reduce the impact of these problems, a hybrid solution which combines data-stripping and no-stripping would be required. In [CHO97], the author presents a hybrid solution where the storage server is divided into server clusters. Video streams are stripped across all the storage nodes in a server clusters. Also, video streams are replicated in all server clusters so that they provide streams to the clients independently.

The hybrid technique is implemented based on the number of server clusters and nodes. That is, if the size of the server cluster equals the number of nodes in the server, video streams are stripped. On the other hand, if the size of the server cluster is equal to one, no-stripping scheme is employed in the server.

➢ ***Load Balancing Techniques***

These techniques deal with balancing the load on the disk array. The placement policy employed can simplify and produce balanced load on disk array causing higher throughput. This policy can be decided based on the characteristics of the media block since each media block might contain either a fixed number of media units (e.g., video frames or audio samples), or a fixed number of storage units (e.g. bytes). But before describing the load balancing techniques, it is worthwhile mentioning some facts about media blocks.

If a media stream is compressed using a variable bit rate (VBR) compression algorithm, then the storage space requirement may vary from one media unit to another. Hence, a server that constructs a media block using a fixed number of media units will be required to store variable size media blocks on the array. On the other hand, if media blocks are of fixed size, then they will contain a variable number of media units. Thus, depending on the placement policy, accessing a fixed number of media units for a stream will require the server to retrieve either a fixed number of variable size blocks, or a variable number of fixed-size blocks.

Due to the sequential playback of media stream, the variable-size block placement policy yields predictable access patterns for the disk array, and thereby simplifies disk bandwidth management. This, however, comes at the expense of increased complexity of storage space management. The fixed-size block placement policy simplifies storage space management at the expense of more complex disk bandwidth management algorithms. Hence, the variable-size blocks are suitable for predominantly read-only environments (e.g. video-on-demand servers), while fixed-size blocks are better suited for environments that involve frequent creation, deletion, and modification of media streams (e.g., multimedia file systems).

In general, multimedia servers might use a combination of static and dynamic load balancing techniques in order to achieve load balancing [GEM95], these techniques include:

- ***Statistic load balancing***: The load balance across disks is done based on the *unit size of the stripe*, the *stripping degree*, and the *replication amount*.

The stripe unit size can be chosen based on whether the multimedia designer is interested in either minimizing the average response or minimizing the variance in response time that lead to decrease in the frequency of playback discontinuities while, in either case, maximizing the throughput.

Small unit size of the stripe produces a uniform load distribution in a disk array. Consequently, the variance in response times is decreased however, the overhead of disk seeks and rotational latencies are increased causing a decrease in disk array throughput. In contrast, large stripe units produce unbalanced load in the disk array and variance in response time however they increase the array throughput. So, the trade off should be done in a way that maximizes the number of requests that can be attended to.

The striping degree depends on whether the system has a small or a large number of disks in the array. If the array has a small number of disks, then striping media streams across all disks in the array (i.e., wide-striping) yields a balanced load and maximizes the throughput [SHE95].

However, in large disk arrays, the media streams are stripped and replicated in subsets of disks in order to achieve load balancing and to maximize the throughput.

Two factors must be considered in order to decide the amount of replication for each media stream. These are: the popularity of the video stream (sometimes termed as hot videos) and the total storage space constraints. Popular movies require more replication than unpopular ones (cold movies), and a larger storage space gives the ability to increase the amount of the replication in order to create more streams.

- ***Dynamic load balancing***: In multimedia servers, the pattern of requests arrival affects the load balance across disks. That is, the requests might overload a disk in disks array. Therefore, the server must employ dynamic mechanisms in order balance load across disks.

For example, multiple replicas for the requested stream can be created and stored on the disks. Thereafter, the server can service the requests from the least loaded disks containing the replicas.

➢ ***Local Storage Management***

Video servers must store the available video files in their storage. In single video servers, all video files are available in the server node and accessible directly by the server. Of course, this would require high disk space.

In distributed systems, local servers (proxy servers) can be implemented based on limited storage capacity in order to reduce storing cost. Therefore, the storage space available in the proxy servers is divided into two parts: one of these parts can function as a cache where, the most popular and requested video files are stored; the other part of the server space can be used for making a distributed mirror of system videos catalog.

The cache allows the increase of the number of requests that can be served locally using a small portion of total server capacity. The inclusion of mirroring schemes aims to reduce the distance of the requests which have failed in their local proxy and to reduce the main server saturation since all requests situated beyond a determined server distance cannot reach the server, thus, less workload. Also, it aims to increases fault-tolerance, since there are several copies of videos within the system, thus, partial system failure does not prevent the rest of the system from continuing its work.

However, the performance of mirroring depends on the distributed architecture and the way the proxy storage is dedicated. For example, one-level proxies achieve a better performance than decentralized systems. The reason is that, when a local-proxy cannot serve a request, this request has to cross one network level to reach the main server. With mirroring scheme, the proxy-miss service distance is affected by distributed-mirror capacity, which depends on the number of proxies situated at the same distance from the local network.

Increasing the dedicated caching in the proxy storage implies a better proxy-hit probability but increases the average distance needed to serve the proxy misses from distributing mirror. On the other hand, if more storage is dedicated for mirroring, the average mirror distance is reduced but proxy-misses are increased. So these two factors are dependent. To enhance the distributed-mirror capacity without modifying the proxy capacity would be done by increasing the number of adjacent proxies from every local network.

➢ *Video Data Retrieval*

The retrieval of video data in multimedia servers can be done based on one of two main paradigms: *server-initiated* or *client-initiated*. In server-*initiated* paradigm, the server generates video streams in a periodical manner with which, the clients will be able to join these streams.

In the client-*initiated* paradigm, the streams are initiated based on the clients' requests arrival. More about these paradigms are discussed in section 1.6.5 in the context of video transmission mechanisms.

Regardless of the data retrieval paradigm, the storage server must employ admission control in order to ensure that the acceptance of a new request for data retrieval does not affect the real-time requirements or the quality of service of the streams already being serviced.

From the above discussion, it can be realized that great attention must be paid to the way the video data must be stored and distributed among the server storage, the way the video data is retrieved, and the amount of the video data that should be retrieved in a timely manner. These factors make multimedia storage servers different from conventional storage servers. These differences come from the fact that multimedia systems require different scheduling algorithms and paradigms to be adopted. Even the choice of the disk hardware affects the storage server since they influence the storage server throughput and play a great role in the success of providing video on demand services.

1.6.4. Video Data Transmission

A crucial point in the design of proxy servers is the media transmission from the video server to the client's playback machine. The video can be transmitted via unicast, broadcast or multicast channels. Therefore, video servers must employ an efficient mechanism for video data delivery in order to improve the efficiency of the service. The way the requests are served and the way the video data is delivered determine the type of the VoD service. In general, current VoD servers can be classified into five categories based on the way the video data is delivered and the interactivity they provide [LIT94][STE95]. These categories are: *Broadcast (No-VoD)*, *Pay-Per-Review*, *Near Video on demand, Quasi Video on Demand*, and *True Video on Demand*.

- ***Broadcast (No-VoD)***

Video broadcasting represents services similar to broadcast TV, in which the video is transmitted from a central server and all of the users are displaying the same view. In this type of service, the user is a passive participant and has no control over the session. It is true that this kind of service does not consume a lot of resources in order to serve large numbers of users however it can't compete with the traditional video rental stores since the user can't interact with the video. Several variations [appendixA] to this kind of systems have been proposed in order to attract the users to the on line VoD services and to give them flexibility to decide when to watch the video and to be able to invoke some VCR-Like functions at will.

- ***Pay-Per-View (PPV)***

PPV provides services in which the user signs up and pays for specific programs, similar to existing Cable TV PPV services. This way, the user will be obliged to pay only for what he decides to watch. Subscribers can use today's set-top decoder box. Previously, the viewer has no control over the movie shown since these set-top boxes do not give the viewers any freedom other than the choice of whether to view the program or not. However, currently this service has been improved so that various scenes can appear on the screen from a set of camera angels (e.g. playing football games). The user, thereafter, can choose from a specific camera.

➢ ***Near Video on Demand (NVoD)***

Typical NVoD systems permit popular movies to be transmitted repeatedly over multiple broadcast or multicast channels to enable multiple users to share a single video channel so that the system cost can be drastically reduced. The down sides are none-interactive control, fixed movie playback and high response time.

➢ ***Quasi Video on Demand (QVoD)***

QVoD presents services, in which programs are scheduled based on a threshold of the number of pending requests. The objective was to batch a number of users who can be serviced at the same time so that the I/O and network bandwidth is reduced. Users can perform at the simplest level temporal control activities by switching to a different group.

In QVoD, the throughput is found to be usually greater than that of NVoD, except for the extreme case of nonstationary request arrivals. This observation is then used to improve throughput without compromising customers' QoS in terms of average phase offset and the corresponding dispersion[ABR97].

➢ ***True Video on Demand (TVoD)***

A common way to provide TVoD services is to allocate a dedicated unicast channel for every user allowing them to select their favorite video to play and perform interactive VCR-like controls at will. In [SUP01], a system has been created based on a multithreaded VoD server which adopted this approach. The design dedicated a unicast channel for each client. This architecture was shown to provide interactive VCR functions and short response time giving the user complete control over the channel. However, the server runs out of resources after a small number of clients are accepted. The reason was that, the resources and the network bandwidth were not efficiently utilized. Therefore, it can’t meet the increasing number of the clients while maintaining a complete TVoD to all clients without delay. The TVoD remains a key issue in designing a successful VoD services.

From the previous discussion, it can be deduced that it is important to use a combination of unicast and multicast transmission techniques in order to provide True Video on Demand (TVoD). Multicast channels are used to initiate new video sessions from the beginning of the video file, while unicast channels are allocated to users who join the multicast sessions in order to transmit the missing initial part of the video (Video Prefix) upon their arrival. Meanwhile the client will buffer the video data from the skew point of the multicast channel. The video rate sent over the multicast channel and the unicast channels is determined by a scheduling algorithm depending on the resources availability and clients' machines specifications.

Also, in the DVoD architecture, it is important to design and implement a video proxy server which will be responsible for providing a direct VoD services to the clients. Due to the long-lived sessions and large bandwidth requirements of VoD services, the architecture of this system requires a careful design and layout of the different components that co-operate among each other to guarantee a successful continuous just-in-time delivery of the requested information.

1.6.5. Scheduling Policies for Media Data Delivery

As mentioned before, a key issue in VoD service is the ability to stream continuous video data from servers to clients across the network providing TVoD service without delay and playback interruption (jitter). Fifteen years ago, most multimedia pundits were predicting that VoD would radically change our home entertainment habits. However, none of the companies that invested in VoD have been able to come up with a single successful commercial system since VoD is still too costly to compete with either video rental or pay-per-view television.

In 1998/1999 Time Warner Cable Comcast piloted a service using Scientific-Atlanta's (SFA) "Explorer 2000" digital set-top box and a SUN MicroSparc/Power TV based STB. Trials by Bell Atlantic and Time Warner proved that though streaming video-on demand was possible, the capital expense of 7000 pound per video stream (per end-user) did not justify the business model [PER01]. If a provider wanted to sell each stream at 2 pound (3€) (i.e. 3€ per movie), viewers would have to buy two movies every day for five years just to cover the initial expense. However, the price for video-on-demand servers has been subjected to Moore's Law, and what used to cost 10500 € now costs less than 1050 € [PER01].

The VoD service is still costly due to the high requirements of server and bandwidth resources (in particular, server I/O bandwidth and network bandwidth). The long-lived nature of digital video would hold these channels for long a time. For example, if a typical two-hour movie is played back at a rate equal to real-time playback rate, then the video channel will be occupied for two hours, and the server must maintain streaming the video data transmission for two hours. This means that the server's resources are also occupied for two hours.

The problem is further complicated by the fact that conventional video streaming systems use a linear playback scheme that forces users to download from the beginning of video. Also, during the playback, the clients may wish to invoke one of the VCR-like functions such as pause, stop, jump forward and jump backward, and they expect that the server should be able to respond without delay.

These factors might have been the reason behind limiting the widespread usage of video streaming over the Internet. Therefore, in order to reduce the VoD service cost and to provide interactive video-on-demand, the server must employ appropriate techniques to efficiently utilize the server resources and stream media data to the clients. In general, there are two approaches for allocating server channels for the delivery video data called: *user-centered approach* and *data-centered approach*.

➢ ***User-Centered Approach***

A conventional VoD system assumes the *user-centered* scheduling scheme in which a user eventually acquires some dedicated bandwidth [AGG96b] [VIS96]. The consumption rate of a video object is equal to the amount of bandwidth necessary to view it continuously. When a client makes a request to the server, the server sends the requested object to the client via a dedicated channel. This scheme incurs high system costs, especially in terms of server storage-I/O and network bandwidths. However, some techniques can be implemented to reduce the cost (See appendix A).

➢ ***Data-Centered Approach***

Data-centered scheduling dedicates channels to video objects, instead of users. It allows users to share a server stream by batching and using the multicast facility of modern communication networks. Also, it has the potential to dramatically reduce the network and server bandwidth requirements. The data-centered multicast VoD service can be either *client-initiated* or *server-initiated* [SHE95] [GAO99].

- In the client-initiated service, channels are allocated among the users and the service is initiated by clients so, it is also known as a *scheduled* or *client-pull* service.

- In the server-initiated service, the server channels are dedicated to individual video objects, so it is also called a *periodic broadcast* or *server-push* service. Popular videos are broadcast periodically in this scheme, and a new request dynamically joins, with a small delay, the stream that is being broadcast.
- Another option would be combining the above two schemes. This combination is called hybrid batching. In practice, it is efficient to use *hybrid batching* that combines the above two schemes.

The delivery of video data to the client involves three steps: (1) reading blocks of video data from the disk (storage node) to a buffer, (2) transmission of the block from the storage node to the network node and (3) the transmission of the block from the network node to the consumer desktop.

➢ *Media Data Delivery Algorithms*

A video server has to supply data blocks of the movie at regular periods to the client. If data is not transferred at regular intervals, the client may experience glitches in the delivery of the movie. To ensure glitch-free service, the video server has to guarantee finishing the three steps of service in a fixed amount of time. Several previous algorithms have been proposed to guarantee the video data delivery in a timely manner. These algorithms are explained in details in Appendix A. In this section we highlight and mention some of the major algorithms used for video data delivery scheduling.

Deadline scheduling algorithms such as [LIU73] and [JEF91] are shown to be optimal however, both of these studies assume that the task completion times are known in advance, but this is not realistic from the point of view of the user.

A slot-based algorithm presented in [RED99] addresses the block transmission from the storage node to the network node over interconnected network but does not consider the block transmission from the network node to the client.

Another approach called batching [DAN94] has been proposed to schedule video requests at the cost of undesirable latency to respond to client requests. The idea is to queue users who request the same video data, then to assign a single multicast channel instead of multiple independent unicast channels in order to reduce resource requirements.

A set of well-known approaches have been presented to improving VoD systems efficiency, such as piggybacking [GOL95] [GOL96] [AGG96a]. The idea is similar to that of batching [DAN94], but the grouping is done dynamically and while the displays are in progress.

Chaining [SHEU97] is also a generalized dynamic multicast technique to reduce the demand on the network-I/O bandwidth by caching data in the client's local storage to facilitate future multicasts. The advantage of chaining is that not every request has to receive its data directly from the server. A large amount of video also becomes available from clients located throughout the network.

The authors of [CAR97] present *stream tapping* that allows a client to greedily tap data from any stream on the VoD server containing video data.

To eliminate the service latency, *patching* was introduced in [HUA98]. The objective of patching is to substantially improve the number of requests each channel can serve per time unit, thereby sufficiently reducing the per-customer system cost.

Broadcasting protocols share the common objective of reducing the total bandwidth required. They repeatedly broadcast the same video over several channels in such a way that a customer may have to wait for a few minutes (e.g. 5 minutes) before he can start watching the video. In [PAR99], the author states two factors that make broadcasting protocols accepted: 1) The saving is considerable since about 40% of the demand is for small number of popular movies (10 to 20) [CLA93] [DAN94] [DAN96], 2) any reduction in the cost of distributing popular videos through the use of more efficient broadcasting protocols will thus have a direct impact on the overall cost of VoD and, ultimately, on its success on the market place.

Pyramid broadcasting [VIS96], permutation-based pyramid broadcasting [AGG96b], skyscraper broadcasting [HUA97], harmonic broadcasting [JUH97] and its variants [PAR98], all, share the same goal and a similar organization (See appendix A).

Asynchronous multicasting [WOO96], [KAL96] intended to improve the system efficiency by allowing the user to join a multicast group and store some video data in local buffer for later use. This is achieved by breaking up the video into segments and sending out these segments at a rate greater than the consumption rate of the video. These approaches send work-ahead data and can be utilized to enhance the system performance.

All practical scheduling policies are guided by three primary objectives: minimize the reneging probability, minimize average waiting time, and be fair. The tradeoffs are between increasing the server performance and the client satisfaction. A hybrid technique would be desirable in order to maximize the server throughput and to provide an acceptable QoS to the clients.

1.7 Network Manager and Transmission Protocols

Network management is essential in order to alert the service provider in the event of failure or degradation in quality of service. Thus, a network manager may play the role of monitoring the network continuously, alerting the people in case of error occurrence, recovery from failures and errors, and reporting solutions [WPG03].

With regard to video transmission from the server to the client, the transmission protocols which manage the video data delivery are considered. In general, there are two important transport protocols that can be implemented for this purpose:

i. The Transmission Control Protocol/Internet Protocol (TCP/IP), and
ii. The User Datagram protocol/Internet Protocol (UDP/IP) [TAN03].

However, other recent protocols have been designed at the top of the TCP and UDP in order to serve the same purpose such as the Real Time Protocol (RTP), the Real Time Control Protocol (RTCP), Real Time Session Protocol (RTSP), and ReSerVation Protocol (RSVP).

In general, the TCP is considered as a connection-oriented protocol since it guarantees the delivery of the packets from a sending machine to a receiving machine in the network without errors. Also, it controls the flow of the data in order to ensure that a fast sender of packets does not overwhelm a slow receiver.

On the other hand, the UDP/IP is a connectionless protocol that does not present any guarantee, and it fits well for the applications that don't require the control of the data flow implemented in TCP, rather, they implement their own flow control mechanisms [TAN03]. The IP protocol is responsible for formatting the data packets and forwarding these packets to their destination avoiding the congestion.

The RTP protocol is built over UDP protocol and, it works together with its RTCP protocol in order to implement a mechanism for video data delivery in real time. It can be used to transfer different data types in real time. The RTP protocol has a field called *Payload-type identification* which permits the receiver to identify the format of the data received.

The RTP, however, does not guarantee the delivery of the packets in sequence. That is, out of sequence packets might be received at the receiver side. Therefore, it adds a *sequential number* at each packet so that the receiver can reconstruct these packets in sequence.

Also, the RTP, by itself, neither offers real time delivery nor guarantees the quality of the service. It assumes that the network layer guarantees that. However, it provides a *time-stamping* mechanism which is added within the transferred data from the server to the client. The time-stamping corresponds to the instant of generating the data packets. When the receiver receives the packets, it can use the time-stamping in order to synchronize the packets and calculate the delay.

Like RTP protocol, the RTCP is also built over the UDP protocol. However, each protocol uses different ports for communication. The RTCP provides quality of service by using feeding back mechanism. That is, RTCP packets are sent to a multicast session. This allows all receivers to receive these packets and calculate the bit rate and the QoS of data transmission through Sender Reports and Receiver Reports.

The receiver reports provides information about the packets lost and the delays which may be experienced and which might cause jitters. The sender can use the feedback in order to adapt to the changes that take place in the system.

The RTP/RTCP protocols support multicast transmission. In multicast sessions, the RTCP controlling packets can be generated by all receivers and received by all members of the sessions. This situation would definitely cause network overhead. An alternative way is to have a process that controls the QoS of all members who are belonging to a specific session.

Although they cause network overhead, RTP protocols would be very efficient in systems that require real-time services since they can guarantee a minimum QoS requested by the receiver.

As a continuation, there is another real-time protocol called Real Time Streaming Protocol (RTSP). RTSP is a client/server protocol implemented at the application level, and offers control over the delivery of video data in real time [WPG04]. It establishes and controls either a single or several time-synchronized streams of continuous media. In other words, it controls the data flow that is transmitted by transport protocols such as UDP, multicast UDP, TCP, and RTP.

The servers and the clients who use RTSP cannot maintain the establishment of a connection. The RTSP determines the session and establish a mechanism for media streaming, deliver the data from the client to the server, close the connection and free the resources. For each session, RTSP defines a unique identifier which identifies the session and its correspondent messages.

In RTSP, the clients can use open and close connection-oriented protocol (TCP) or connectionless protocol (UDP) in order to send requests to the server and to communicate with it. For each requesting message, RTSP assigns a sequence number in the *Cseq* field. So, the answering message will have the same number which corresponds to the requesting message in its Cseq field.

Finally, the RSVP protocol is part of a larger effort to enhance the current Internet architecture with support for *Quality of Service* flows. The RSVP protocol is used by a host to request specific qualities of service from the network for particular application data streams or flows. RSVP is also used by routers to deliver quality-of-service (QoS) requests to all nodes along the path(s) of the flows and to establish and maintain state to provide the requested service. RSVP requests will generally result in resources being reserved in each node along the data path [WPG05].

In order to decide what protocol to choose, one has to decide what kind of VoD services the server is going to provide. This is important since each protocol has its own features. For example, if a video server transmits work-ahead data then a TCP protocol with additional data regulation would work well. RTP protocol generates network overhead and does not guarantee the data delivery but it can work well with addition control protocols and would be a good choice for applications such as video conferencing.

1.8. The Client

The client is the one who connects to the server, negotiates the QoS, and receives the video data which will be played at his display machine. In fact, the real evaluation of the VoD service will be done at the client's side.

Typical video client architecture consists of the following two components:

1. A graphical user interface: by which the user can connect to the server and browse the video files.
2. Video player. In the market, it is possible to find so many different video players. For the researches, it would be better to pick up the ones which have an open source. This will allow researchers to investigate, analyze and make some enhancements, if necessary, to the player. An example

for such player is MPEG Berkeley-based player [WPG06] and Xine [WPG07].

In the past, the client did not have large buffer space and had lower I/O bandwidth than today's clients. That situation put constraints on the quality of VoD applications. The lack of buffer space, for example, forced the designers to provide real-time transmission. That is, the amount of video data transmitted within a period of time (let's say 1 second) must be enough to produce video display on the client's machine for one second. Otherwise, starvation or jitter will occur. This situation raises some problems such as imposing strict deadlines on scheduling the video streams and the system resources in order to meet such service. Another problem would be the reduction in the server performance. That is, even if the server had plenty of resource and was able to provide more video data, yet it was not able to do that and the video delivery was restricted to the client capacity.

Likewise, having small I/O bandwidth reduces the video QoS and the server performance. We know from a previous section (1.4) that the bit rate of the multimedia data is high compared with text material. So, the solution is to send video data at a very small bit rate.

Currently, the client is becoming smarter and more cooperative. That is, the client can help in increasing the server performance and QoS by providing a moderate buffer space and a high I/O bandwidth. These changes will cause the researchers to revise their previous work and to consider the progress of the client's capacities in any future design.

For example, today's clients can receive the video data from multiple channels and buffer work ahead video data. This situation affects positively on the design of the client architecture, and affects positively on the way the scheduling algorithms are designed and the way the video streams are streamed. That is, if the server has plenty of resources, and the client has the ability to receive at a rate greater than the real-time playback rate, then the server can assign more resources to this client increasing its performance and providing better quality of service at the client side.

1.9. Thesis Overview and Contributions

The objective of the thesis is the design and implementation of an efficient video on demand server that could play the role of a centralized VoD server and/or a proxy VoD server in DVoD systems. From the above information, it can be concluded that VoD server architecture can be divided into a set of modules (such as Admission Control module, Resources Management module, Video Storage module, and delivery algorithm) that needs to work and cooperate among each other in order to output a successful job which is providing VoD services. The way these modules are designed and laid out in the server architecture affects the server performance and the quality of service.

In addition, the video streaming mechanisms implemented, determines the type of the service that can be provided. In this work, we provide video streaming using multicast and unicast channels and means of patching and pre-fetching techniques. Multicast channels will be created as a response to a request to a new none-active video file (a file that does not have any active session) or as a response to a request that could not be incorporated to an existing multicast session, so another session for the same video file is produced. The unicast channels, however, will be used mainly to transmit the initial part of the video (video prefix) and support VCR functions.

This approach does not require the split of the channels into dedicated unicast and dedicated multicast channels to provide the VoD service. These channels are allocated dynamically and according to their use. Also, contrary to broadcasting and batching techniques, we provide interactive VCR functions and treat our video files fairly without classifying the video files into popular and non-popular. This fairness comes from the fact that, in the educational centers such as universities, schools and other institutions the clients (e.g. students) usually request old video files such as history, documents or experiments done in earlier years. Thus, we can’t ignore these video files in favor of the recent 10 or 20 popular movies.

Another important issue considered in this work is that, clients usually have different machine capabilities in terms of buffer space and I/O bandwidth. Therefore, our approach takes into consideration this fact and adapts itself to the heterogeneity of the clients who may join the multicast channels. The heterogeneity can have positive impact when the clients have high capacity in terms of buffer space and I/O bandwidth and the server has available resources that can be assigned to these clients. However, heterogeneity may have a negative impact when clients with less capacity are merged into multicast streams which transmit at higher capacity. This will require that transmission rate be adjusted to an appropriate rate that matches the capacity of all clients joining the multicast stream. These problems are tackled by our proposed algorithm called Credit Based Media Delivery Algorithm (CB_MDA).

The contributions of the thesis are the following:

- The design and implementation of a scalable multithreaded video on demand system with concentration on video data delivery algorithm. This system provides interactive video on demand where the user can select his favorite video from videos list and decide when to watch it. The server employs admission control module which enables high requests acquisition and allows the users to get served in a short period of time.
- The proposal and analysis of four representative admission control policies. These policies translate the users' requests into resources using four brokers: CPU Broker, Memory Broker, Disk Broker, and Network Broker. This analysis has required extracting meta information from the video files. We have created some programs to generate the necessary information from MPEG video files.
- Conclusions from the analysis of the brokers and the need to provide interactive TVoD service have led to design and implement a Credit_Based Media Delivery Algorithm (CB_MDA) which regulates and controls the flow of the video data from the server to the client, guarantees on time delivery during the video reproduction, and provides VCR-Like functions such as pause, stop, play, jump forward, and jump backward, yet the algorithm does not require reserving dedicated channels in order to achieve its goals. The CB_MDA uses multicast and unicast channels and implements patching and pre-fetching techniques in order to increase the performance of the server and supply instantaneous VCR commands. The algorithm can, also, dynamically adapt to the system changes such joining and disjoining the service during the video reproduction.
- Providing interoperable client architecture. This is done creating a communication module which is separated from the graphical User Interface and the player decoder modules. The architecture allows other players to be used without significant change.

The characteristics of our VoD server are the following:

- Multithreaded server
- True Video on Demand and short response time
- Free of starvation service
- Scalable video services

In our work, and to be more specific, we use the term VoD in order to refer to Movie on Demand service. The movie on demand system was the choice since, as we mentioned before, this type of applications is supposed to deliver a huge amount of video data per second for a long period of time (e.g. two hours) (Table 1.4). Although other types of video applications include huge amount of video data per second but many of them are played in a short period of time and may not provide interactive VCR-like functions.

Chapter 2
Video-on-Demand Architecture and Design

2.1 Introduction

As it can be seen from the previous chapter, the internal design of the video server, whether the server is working as a single video architecture or as a proxy server in multiple-video architecture, plays a great role in the success of providing a scalable interactive VoD services.

This fact has led us to concentrate on the issues related to the major components that compose the internal design of the server, such as the admission control, the resource management, the video storage, and the internal communication, with concentration on the video data delivery algorithm.

Also, we have seen that a careful design of the client architecture is important in order to synchronize well with the server and to guarantee multiple video screen and interoperable client. For this reason, and to test our design, we dedicated a great work on the design of the client architecture.

Figure 2.1 shows a general view of the main components that will compose our VoD clients/server architecture. As the figure indicates, the clients send their requests as commands (Cmds), the commands are received, processed and the quality of service is negotiated by the admission control (AC) module.

The resource manager provides the AC with the necessary information for decision making. Also, the media delivery algorithm is informed by these information and schedule the media data delivery. The server, in general, needs to store information about the video files and the users who have been admitted by the server. This information is important for calculating the system resources and for deciding the bit rate that can be sent to the clients.

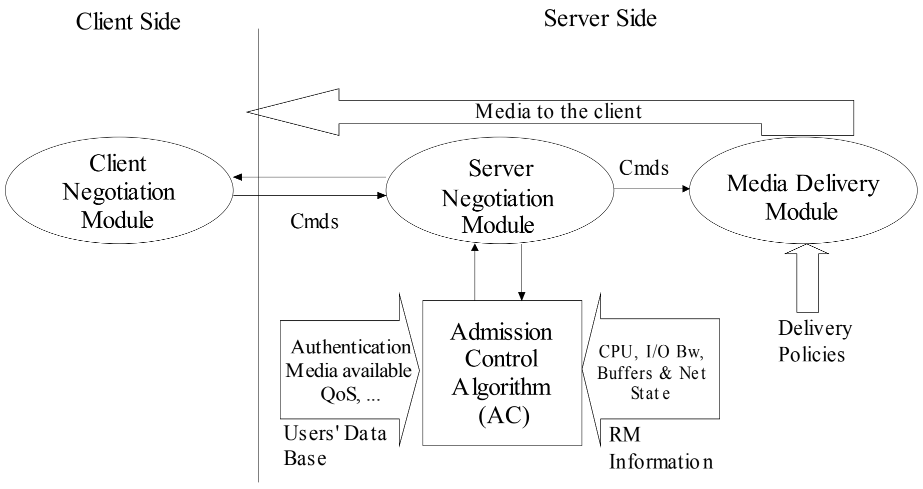

Client Side
Server Side
Media to the client
Client Negotiation Module
Cmds
Server Negotiation Module
Cmds
Media Delivery Module
Authentication Media available QoS, ...
Users' Data Base
Admission Control Algorithm (AC)
CPU, I/O Bw, Buffers & Net State
RM Information
Delivery Policies

Figure 2.1. Main components of a VoD server

2.1.1 Server Design Objectives

In this chapter, the focus is on the design and implementation of the admission control and resource management sub-systems as well as the internal communication between them. From now on, the terminologies video server and proxy server are used interchangeably to indicate the same thing which is a single video server responsible for a direct video data transmission to the user. Also, the words client, user, viewer, and customer are used to refer to the same person (male or female) who requests the VoD service.

The entire work which includes the server design and the architecture has been implemented with the following objectives in mind:

- Designing a VoD server that can function as a central VoD server or as a Proxy Video Server that can be used as a building block in Distributed VoD (DVoD) architectures such as the one presented by our group [COR02].
- Providing scalable service in order to support a large number of users.
- Permitting users to decide when, where and what to watch.
- Providing a True Video on Demand (TVoD) service with a short response time to the different users' requests.

- Guaranteeing a continuous media without jittering during the playback of the video.
- Supporting instantaneous interactive VCR-Like functions such as Stop, Play, Pause, Jump forward, and Jump backward without delay.
- The final server design could play the role of en educational instrument in educational institution such as universities and schools.

2.1.2 Server Characteristics

The environment where we have implemented our design and tested our objectives assumes off shelf products and has the following characteristics:

- Scalable VoD Server with data storage system based on optimized file-system-oriented server.
- Network without real time protocol. Real time protocols (RTP/RTCP) have been considered for calculating the overhead caused by implementing these protocols.
- Public domain based software (Operating System) such as Linux distributions (e.g. SuSe, Red Hat, Mandrake).
- Control and management of network / storage devices and, CPU scheduling from the user layer. This implicates the creation of a middleware which will take over the control of the corresponding systems.

2.1.3 Server Functionality and Components

Recalling from the previous chapter, a typical Video on Demand (VoD) system is composed of video server/s, clients and communicating channels. Clients connect to the server and select their favorite movies that they would like to play. Servers store movie files, process clients' requests and guarantee video data delivery in timely manner. The network communicates the clients' requests to the server and, in return, it delivers the video data to the clients. The clients can then playback the movie at their displaying machines with the ability to invoke VCR-commands using GUI.

To handle the task cycle of requesting services and delivering data, we have created the server based on modules which work as a client/server. Each module is responsible for executing a specific task and conveying the result to the module requesting the service.

Figure 2.2 shows a high-level overview of our video on demand architecture which consists of two main parts, a Video Proxy Server (VPS) and Client architecture. The VPS includes the main components that compose the server and the internal communication between them. The client architecture is focused on the video daemon, the player and the GUI. The network is expected to provide high bandwidth since VoD systems deliver great amount of video data.

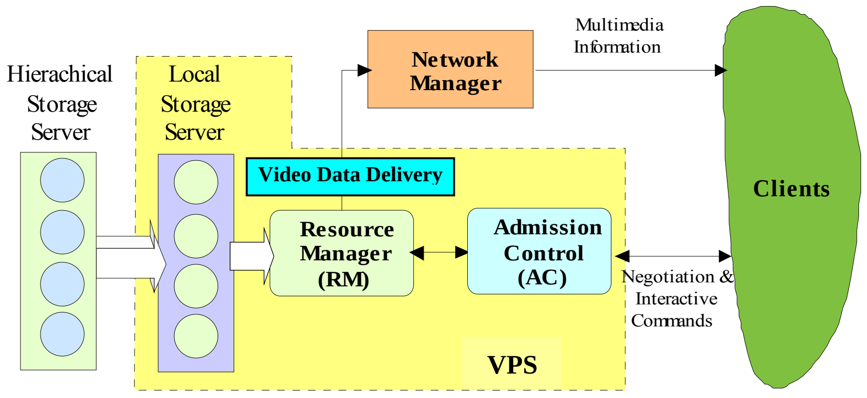


Figure 2.2. Overview of VPS architecture

The next section will focus on the details of the design and implementation of the VPS architecture. It explains the main components of the admission control module followed by the resources management. Also, it analyzes four admission policies which can be implemented in order to manage the system resources. The communication between the different components and the storage management are also presented.

2.2. VPS Design and Architecture

As it can be noted from the previous discussion, requesting movies at will involves steps taken by the server before the service can be granted. These steps are classified and summarized in our architecture as follows:

1. Requests Acquisition and Analysis
2. Admission Control
3. Resources Management and Assignment
4. Video Data Retrieval and
5. Video Data Scheduling and Transmission

To carry out these steps, we have studied, analyzed, and designed different important modules which compose the VPS (Figure 2.3). The **Admission Control Module (ACM)** and the **Resource Manager Module (RMM)** control the admission of the requests and the assignment of the server's resources. The **CB_MDA Module** is designed to schedule the delivery of the video data to the clients. The **Local Storage Server (LSS)** is responsible for storing video files which will be served to the clients, and providing zero delay to the different requests. Also, the communication between the different modules and the control of signals and data flow between the different components is also considered.

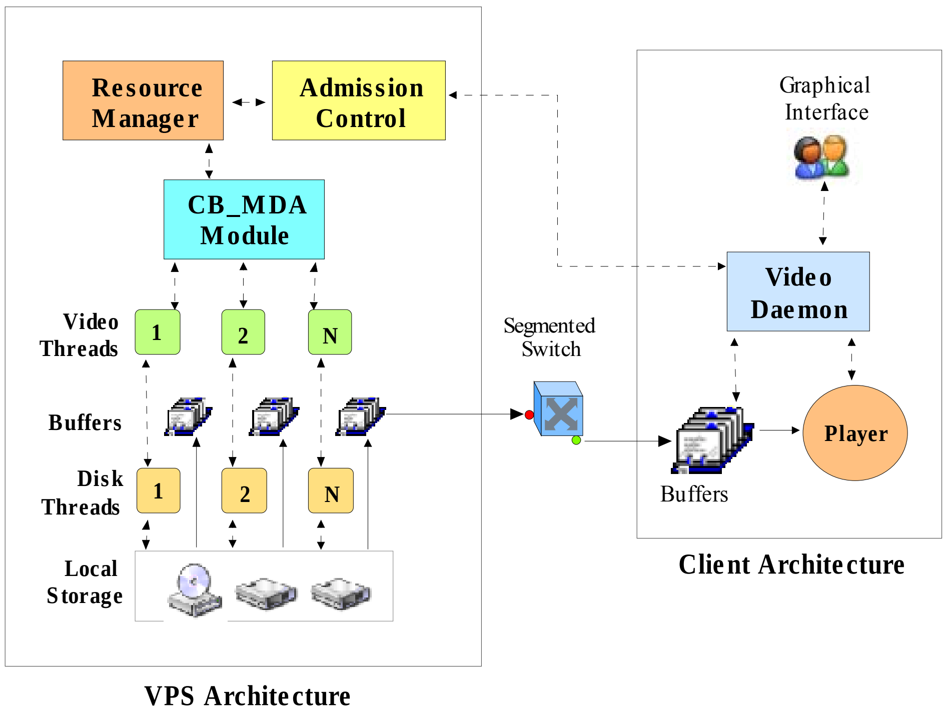


Figure 2.3. Our proposed VoD clients/server architecture

On the other side, we have the client architecture. The client has been designed to test the server viability. In its design, we have considered the data transmission from the server to the client so that the client can synchronize with the server. In the design, the client has been created as an interoperable one. That is, other client can use other type of players such as real player.

The media delivery algorithm and the client architecture are discussed in chapter three and appendix-B respectively.

In VPS architecture, and before we describe its components, it is worthwhile mentioning that we have purposely separated the ACM and the RMM from each other. The decision of having the ACM and RMM as different and separated components is due to two major points: first, reducing response time and second, making the system scalability feasible.

Also, we made separation between video data path and control flow in both, the VPS and the client architectures. This separation comes from the fact that data path and control function imposes different requirements on the underlying system.

Another important principle embodied in VPS and client architectures is the use of IP multicast. Multicast offers excellent scalability which, in turn, enables servicing a large number of clients and provides excellent cost / performance benefits. In spite of these advantages, multicast VoD introduces certain new difficulties in supporting VCR commands such as interactivity with multicast services while improving service efficiency.

The Multicasting transmission and providing instantaneous interactive VCR-commands without glitches are handled by the CB_MDA algorithm, which is discussed in chapter three.

The main components of the VPS architecture and the main issues related to each one of them are clarified with details in the following section.

2.3. Admission Control Module

The Admission Control Module (ACM) makes sure that the acceptance of a new client does not affect the performance requirements of other videos already being serviced, and, that adequate resources are available throughout the entire path from the video server to the client displaying device.

Also, the ACM attends to requests received from clients by analyzing them and deciding, based on information received from the resource manager about the resources availability, whether a request can be attended to, and, given that a request cannot be attended to because of the lack of resources in the system, the ACM renegotiates a QoS for the request so that it can be successfully accepted according to a given policy defined by the resource manager.

2.3.1 Requests Handling

For the ACM to handle the requests arrival and the negotiation and renegotiation with the client, we have created three sub-modules to manage these actions. The sub-modules are: *Requests Acquisition Sub-Module*, *Requests Admission Sub-Module*, and *Renegotiation Sub-Module.*

- ❖ Requests Acquisition Sub-module: This sub-module is a single listener thread that listens on a well-known port for incoming client requests. It is, simply, responsible for receiving the requests invoked by the clients. Once it receives a request, it puts it in the arrival queue and signals the requests admission module in order to handle this request.
- ❖ Requests Admission Sub-module: This sub-module is responsible for negotiating with the client and deciding, depending on the available resources, whether a petition can be attended to or not. In the case of rejection, the request can be renegotiated.
- ❖ Renegotiation Sub-module: given that a request has not been accepted because of the lack of resources in the system, this module can renegotiate a QoS for the requested video data so that it can be successfully accepted according to a given policy.

2.3.2. Communication Commands

For the ACM to communicate with the clients and other parts of the server, we have created a set of commands in order to distinguish the requests type from each other. These commands include:

1. ***NEW***, means that the client is new and has no profile in the system. For this request the ACM creates a thread in order to negotiate with the client. Then, it informs the client about the negotiating port. At this stage, the client synchronizes with the server and the thread sends a list of available video files in the system along with their format, size, frame rate, and playing time.

   The idea behind having a thread to negotiate with the new clients rather than the ACM itself is to achieve high requests acquisition by the server and to achieve a very short response time from the client point of view.

2. ***LIST***: This command is sent by the ACM to the client, and it indicates to the client that the attached data represents a list of the video files available in the server.

   On the other side, the client has a GUI where he can browse the video files and some information related to these files. He selects his favorite video file and sends his request accompanied with the command PORT.

3. ***PORT***, means that the client has chosen the file, and he wants to know the port where he can receive the media data from. The ACM asks the resource

manager to create two kinds of threads which are: video retrieving thread “Vthr-i” and video delivery thread “Dthr-i”, for data retrieving from the disk and data delivery to the client respectively. Thereafter, the ACM informs the client about the receiving port.

As it can be seen from the above command, we are making another separation between the video data retrieval and the video data transmission to the client. This separation of functionalities is very important for three main reasons.

i. It helps the data delivery scheduling algorithm (CB_MDA) to assume zero delay in data retrieval. And this is can be guaranteed by any of the disk scheduling that could guarantee zero delay scheduling. However, in our case we are relying on the disk broker which has been designed by our group.

ii. The separation creates flexibility in choosing or changing the disk scheduling algorithm at will without the need to make changes in the video data delivery algorithm.

iii. This separation also increases the server efficiency since it reduces the communication overhead that would be generated between the threads and the system resources if each thread had to investigate the resources availability. Instead, the resource manager reports periodically, or as a result of changes in the system, the resources availability to the ACM, then the ACM uses this information to decide whether to admit a request or not.

4. ***CMND***, means that the client is registered as an active one and is sending one of the interactive control commands (Play, Pause, Jump forward, Jump backward, Stop). The ACM responds by forwarding this request to the delivery algorithm (CB_MDA) which will reschedule this request according to its type.
5. ***RESOURCE***: The command is sent from the ACM to the RMM in order to check resources availability. The RMM investigates its resources such as the disk, memory, CPU and the network and then, reports to the ACM.

2.4. Resource Manager Module (RMM)

It becomes clear that, prior to admitting a new client for VoD services, the VoD server must make sure that it has sufficient resources in order to guarantee that the QoS contracted to existing clients is not jeopardized. Therefore, we have considered and implemented the resources management in our VoD system so that the resources can be guaranteed and reserved for the accepted requests during the entire video session.

Unfortunately, variable bit-rate video streams create difficulties in determining the amount of resources needed. Of course, this has created problems for us in determining how to guarantee and reserve the required resources.

To solve this problem, two solutions that can be studied and implemented are presented. The first one is called optimistic solution, with which, the reservation of resources can be made based on the minimum bit-rate. However, this approach may overload the resources when unpredictable behavior occurs. A second solution would be based on pessimistic approach. The pessimistic approach generates an under-utilization of the resources since the maximum bit rate would be expected.

The assignment represents a trade-off between both approaches [NAH99]. That is, shall we create an in-between or an intermediary solution where we create a deterministic mechanism to determine the required resources, or create a predictive solution augmented by mechanisms in order to adapt to the changes that may occur during the video sessions?

To answer these questions, in this section, we explore the impact and the effects of the deterministic and predictive policies on the server performance. These policies are applied to the server resources which are managed by RMM.

In our work the Resource Manager Module (RMM) translates the user's requests into QoS parameters, requests the necessary resources and reserves them using four brokers (threads) which are: *CPU broker, Memory broker, Network broker,* and *Disk broker* (figure 2.4). Each broker works separately and reports to the RMM about its status according to a predefined policy. If all of them report positive confirmations to the RMM, the petition is then accepted and registered in the admitted users list.

The CPU broker calculates the CPU computing time of the requested media data, the memory broker computes the memory needed to buffer the media data before delivering it to the client, the required network bandwidth is calculated by the network broker and, finally, the disk broker calculates and guarantees the requested bandwidth. The data generated by these brokers is translated into information used by the ACM in order to decide whether to accept or reject a request. Also, this information is used by the CB_MDA for scheduling the video threads which are responsible for video data delivery.

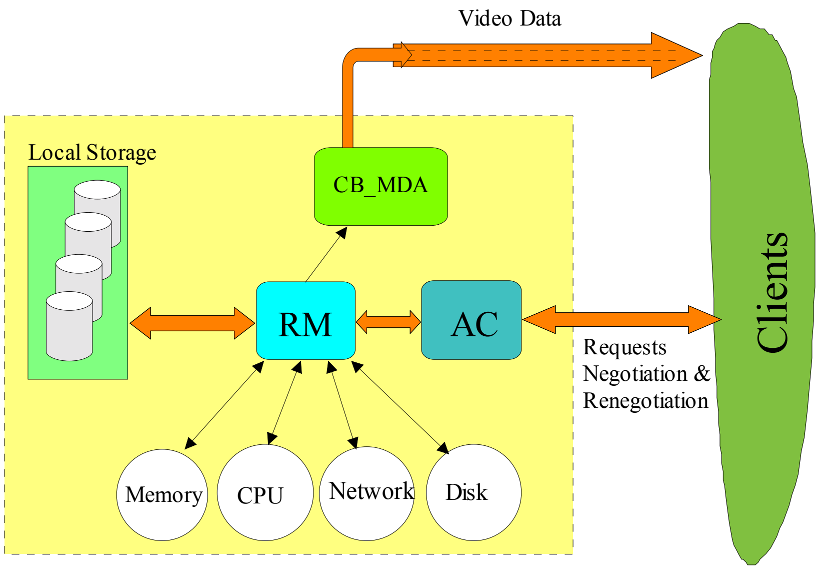


Figure 2.4. Resources Mangaement Brokers

The brokers have been designed as an intermediate layer, called middleware, between the application layer and the operating system layer (Figure 2.5). The decision to implement a middleware is based on the fact that, normal operating systems don't offer a QoS required by VoD systems.

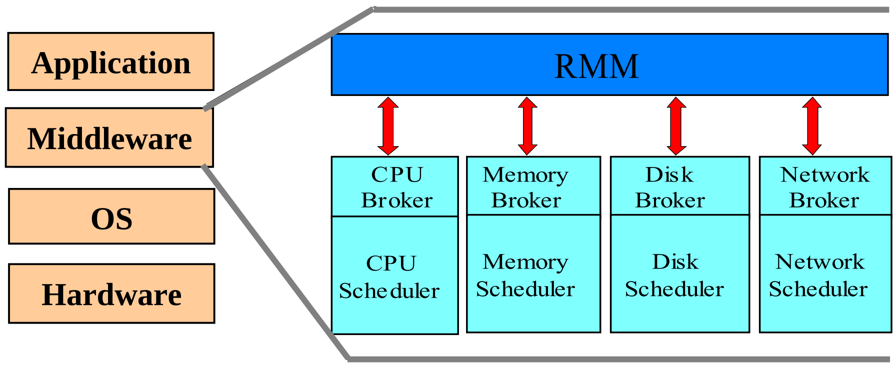


Figure 2.5. Middleware implementation of the brokers

Therefore, and as we have mentioned in the previous chapter, for an operating system to provide a QoS, its kernel must be modified and the reservation algorithms are contained in the kernel. Being included in the kernel, a higher resolution and exact time can be obtained in order to reserve the resources. However, the modification process of the kernel is complex, and the service will be always committed to the implemented algorithms inside the kernel.

On the other hand, the middleware has the advantage of applying and select between different scheduling policies. Also, the middleware approach permits to adapt to the characteristics of the hardware.

In the brokers design, we have studied, analyzed, and implemented four specific policies for the admission control [QAZ03a]. These algorithms are: Maximum Policy (MP), Adaptive Maximum Policy (AMP), Average Policy (AP) and Adaptive Average Policy (AAP). The adaptive policies are developed to improve the application throughput.

To test and to evaluate these algorithms, we have used MPEG-1 and MPEG-2 format for the examples analyzed in order to present a real comparison between them. The MPEG files such as MPEG-1 and MPEG-2 are compressed files which consist of a sequence of three types of frames I, P, and B. These frames are explained as follows:

- I-frames (Intra-coded images) are coded as a standalone images that have no reference to any other frame in the sequence of the frames.
- P-frames (Predictive-coded images) are coded based on information from the previous I-frame and/or all previous P-frames located between the previous I-frame and the P-frame. This implicates that, the decoder software needs these frames to be available in order to generate the new decoded frame.
- B-frames (Bi-directionally predictive-coded frames) are coded based on information from the previous and next I-frames or P-frames. These frames can’t be coded unless and until the following I-frames or P-frames are processed in order to obtain the reference points.

The algorithms can work with two modes, frame-mode where video data is sent frame by frame, and block-mode where a block of video data, instead of frame, is sent. The algorithms in their version of frames (I, P, B) and blocks (a packet of fixed number of bytes), are pertaining to the Deterministic and Predictive categories and, they are analyzed and implemented in the CPU, memory, disk and Network brokers.

However, since the CPU broker is the first one to start the discussion with, it can be realized that it has more details than the other brokers. This situation of unbalance shall not indicate less importance of the other brokers rather, it refers to the fact that, the policies are introduced the first time in the CPU broker. So, more details would have been needed to create an understanding to the implementation of the different policies. Thus, in the subsequent brokers, it would be enough to make references to the same concepts which have been mentioned in the CPU broker.

The analysis of the algorithms in the context of the resources management brokers, of course, requires some information about the contents of the video files such as; data format, file size, frames sizes, frame rate per second. This information has been extracted from the video files as it is explained in the following subsection.

2.4.1. Obtaining Metadata

The algorithms require the extraction of meta-information from MPEG video files, and the creation of necessary tables which are used to keep track of the system status. For this reason, we have used a Berkeley analysis tool called mpeg_stat [http://bmrc.berkeley.edu] along with our programs in order to generate the necessary information files from the video files. The information generated from these file is also used to make some calculations such as the computing time of accessing a frame or a block of video data and to create necessary files and tables used by these algorithms.

For each mpeg video file (e.g. Futurama206.mpg) a series of meta-information files are generated with different extensions. One of the files which, has been used as a base for generating other information, is the one with the extension .sz (e.g. Futurama206.sz). Table 2.1 shows the kind of the information that would appear in such files like Futurama206.sz.

Table 2.1. Snapshot from a video file

Frame No.	Frame Type	Frame Size
0	I	20792
1	I	108320
2		
.		

The table shows that Futurama206.sz contains three columns. The first one refers to the frame number. This number is sequential and indicates how many frames are in the video file. The second column corresponds to the type of the frame. As we said before, MPEG format contains three types of frames, the Intra frame I, the prediction frames P, and the bidirectional predictive frames B.

From the (.sz) files we have generated useful meta-information for the analysis of the algorithms. For example, we have extracted the frames I, P, and B which have the maximum sizes. Also, we have counted the number of I frames, P frames, and B frames in each file. In addition, we have calculated the average sizes of these frames (I, P, and B). This information has been stored in files which have (.meta) extension.

The algorithms are analyzed with frame-based and block-based modes. That is, the video data can be retrieved from the storage disks frame by frame or block by block. Therefore, it is important to determine the block size (e.g. 32KB) of the video data that will be sent to the client. Based on the block size, we calculate the number and the average numbers of the I, P, and B frames in each block. The generated data for each video file is stored with (.bk) extension. Table 2.2 shows the structure of the (.bk) file.

Table 2.2. Frames sequence from a video file

Block No.	Frames per block
0	IIBPBB
1	PBBPBBP
2	BBIBB
3	PBBPBBP
.	
.	

Table 2.2 shows that Block 0 has a combination of I, P, and B frames. The next block has only two types of frames (P and B). The number of blocks depends on the block size, and the bigger the block size is the larger the number of frames is.

In order to incorporate new files, we basically have created a file called movies.info which contains all available video files. The file includes information about each video file such as Identification number (Id), file name, dimx and dimy (Table 2.3). Using movies.info file, we call a process which creates a directory for each file. The directory is named after the Id of the video file.

The previous table indicates that the Futurama206.mpg has an Id equal to 1. Thus, a directory named after the movies Id is generated starting from a directory called meta directory and, all the generated file will have the same name (Futurama206) but with different extensions such as, Futurama206.bk, Futurama206.info, Futurama206.zs…etc.

Table 2.3. Partial list of video files

Id	File name	Dimx	Dimy
1	Futurama206.mpg	352	240
2	BC.mpg	352	240
3	Christmas.mpg	240	160
4	Flash.mpg	352	240
5	Ontario.mpg	352	240
.	.	.	.

In addition to what we have mentioned above and to facilitate the admission control and to permit the adaptation of the VoD system to the characteristics of the hardware used, we have used the above information in order to calculate the computing time of the largest I frame size, the computing time of the largest P frame size, the computing time of the largest B frame size, the computing time of the average I frame size, the computing time of the average P frame size, the computing time of the average B frame size, and the computing time of the data block of each movie. This information is made available to the different algorithms by a table called mptable (Movies Profile Table), and will be used mainly when the algorithms calculate the CPU resources needed to satisfy a client's request.

Another table called mmtable (Movies Meta Table) is also made available to the algorithms. This table contains the total number of the frames, the number of I frames, the number of P frames, the number of B frames, the largest size of I frames, the largest size of the P frames, the largest size of B frames, the average size of I frames, the average size of P frames, the average size of B frames, the block size, the number of blocks in the video file, and the number of frames per block. This information is used based on the algorithm used. For example, the Maximum Policy uses the largest frames while, the Average policy uses the average sizes of the different frames.

The importance of the information extracted from these files can be seen more clearly when they are implemented during the analysis process of these algorithms in the context of the resources brokers.

2.4.2. Overview of the Applied Policies

In the MP policy (Maximum Policy), the brokers consider the size of the largest I-frame and maintain this value during the whole service. Therefore, this policy achieves high quality of service since the maximum value is maintained all the time, however it miss utilizes the server resources since it might reserve more resources than what would be actually needed by each request.

To overcome this problem, the AMP (adaptive maximum policy) takes into considerations the average size of the largest I-frame, P-frame and B-frame during the service. The AMP achieves high performance since the amount of the reserved resources is reduced, yet it provides an acceptable high quality of service.

In the AP policy, the brokers considers the video average sizes of I-frames, P-frames, and B-frames, while in AAP policy the brokers consider the characteristics of the video (increase or decrease of the quality of service when the number of I-frames is increased or decreased) using the values of the AP policy. The performance results are discussed in chapter four and reported in [QAZ03a].

The brokers have been designed with the flexibility of applying any of these policies at the starting time of the server. This depends on the server load. For example, if the server is lightly-loaded then the brokers can make the reservation of the resources based on MP. Otherwise, they might apply the AMP in order to reduce the amount of the resources assigned to the requests so that more clients can be accepted.

The adaptive algorithms show that they use an exponential weighted average to guess the future resource needs of a stream, rather than just using the maximum or mean over the whole period. Thus, they provide some smoothing and so a closer matching of resources used to those reserved and so higher capacity.

2.4.3. CPU Broker

The CPU broker is responsible for reserving the necessary resources and scheduling them based on a policy defined by the RMM. In this broker, there are three parameters which will be considered in calculating whether a request (req_i) will be admitted or not. These parameters are CPU computing time C_i, CPU cycle time T_i and CPU utilization U_i.

The translation of C and T are calculated differently from one algorithm to another, except for the CPU utilization, which can always be expressed as it is stated in (1).

$$U_i = C_i / T_i. \qquad (1)$$

In the analysis of our VoD application, the computing time of the video frames and blocks have been generated off-line. Thus, the CPU utilization is calculated by dividing the computing time of a frame or block by how many times the computing time has to be done.

In the frame-based, the computing time is changing while the period time remains constant however; in the block frame the computing time remains constant while the period time is changing. The reason for this difference refers to the fact that video files consists of frames of different sizes. The frequency of playing these frames per second (fps) determines how many times the calculation is done. On the other hand, a block of data consists of varying number of frames which makes the number of calculations varying from time to time.

Whether frame-based or the block-based is implemented, the criterion to admit *req*$_i$ is based on the following admission test:

$$U_i + \sum_{j=1}^{N} \frac{C_j}{T_j} \leq 1 \qquad (2)$$

where N is the number of attended requests and U_i is the CPU utilization of the *req*$_i$. If this test is fulfilled, the CPU broker will attend to the new request and register it in the list of the admitted requests.

The parameters C and T which intervene in the admission test are calculated according to the implemented policies as follows:

- ***Maximum Policy with Frame-based mode***. This policy considers the computing time (ct) of the largest frame in the video file. This implicates that the computing time of the largest I-frame *max(I)* is considered, and this value is maintained throughout the entire service. Thus, *Ci* is assigned the computing time of the larges I-frame as in (3). With regard to *Ti,* since the mode is frame-based, in (4) *Ti* is calculated by dividing 1 by the number of the frames.

$$C_i = ct\,(max_i(I)), \qquad (3)$$
$$T_i = fps_i^{-1} \qquad (4)$$

The fps represents the frame rate of the video file per second (*fps*). As we said before, fps is extracted from meta data files and used to calculate *Ti*.

Thereafter, the values of (3) and (4) are substituted in (1) in order to calculate the cpu utilization so that the admission control test is done according to (2).

• ***Maximum Policy with Block-based* mode**. With this policy, it is assumed that all frames are of type (I) since the largest frame is considered. Therefore, the computing time C_i of the block (*ctBlock*) will be as in (5) and, it will be constant all the time since the block size is constant too.

$$C_i = ctBlock \qquad (5)$$

Ti, in this case, is calculated differently since the number of the frames varies from block to block. Therefore, the number of frames per block (*fpb*) is calculated as in (6) where, the *BlockSize* is defined by the application (e.g. 32 KByte, 64 KByte…etc), and then, Ti is calculated as in (7).

$fpb_i = BlockSize / Size(max_i(I))$	(6)
$T_i = fpb_i / fps_i$	(7)

Thereafter, and likewise, the values of (5) and (7) are substituted in (1) in order to calculate the cpu utilization so that the admission control test is done according to (2).

• ***Adaptive Maximum Policy with Frame-based mode***. The Adaptive Maximum policy is different from the Maximum policy in that it considers the computing time of the largest I-frame (maxI), P-frame (maxP), and B-frame (maxB), and, it adapts to the changes while the film is running. The computing time of these frames is calculated. In addition the frequency of the frames appearance in the video file must be known. This frequency is represented as the percentage or the probability of the frames appearance in the video file.

With *Frame-based* mode, the steps for testing the admission control are done as follows:

1. The ct of I-frame is assigned the ct of the largest I-frame as in (8a), the ct of P-frame is assigned the ct of the largest P-frame as in (8b), and the ct of B-frame is assigned the ct of the largest B-frame as in (8c).

$ct_i(I)=ct(max(I)),$	(8a)
$ct_i(P)=ct(max(P)),$	(8b)
$ct_i(B)=ct(max(B)),$	(8c)

2. The C_i is calculated as in (9) by multiplying the ct of each frame by the probability of its appearance in the video file. $\overline{p_I}$, $\overline{p_P}$, and $\overline{p_B}$ are the frequency of the frames appearance in the film.

$C_i = ct_i\,(I)\,\overline{p_I} + ct_i\,(P)\,\overline{p_P} + ct_i\,(B)\,\overline{p_B}$	(9)

3. Since the policy is frame-based, Ti is calculated the same as it is stated in (4). This equation is stated below for the ease of reading it.

$T_i = fps_i^{-1}$

4. The *adaptive mechanism,* (10) (11), consists of recalculating the computing time in terms of the frequency of each frame type that appears in the future time t where $t>1$. The degree of adjustment is considered as parameter λ where, ($0 \le \lambda \le 1$) and is used for each $p_{k,}$ where k can be I-frame, P-frame, or B-frame. The probability of the appearance of future frames in (t+1) is obtained using the meta-information extracted from the video files. Therefore, we can predict the type of the frame before sending it. If lambda is equal to 1 then the adaptation is totally predictive.

$C_i\,(t)= ct_i(I)\,p_I(t) + ct_i(P)\,p_P(t) + ct_i\,(B)\,p_B(t)$	(10)
$p_k(t) = (1-\lambda)p_k(t-1) + \lambda\, p_k(t+1)$	(11)

5. Ti is computed as in 4 since the mode is frame-based, and the Ui is calculated by using (11) divided by (4).

- ***Adaptive Maximum Policy with Block-based mode.*** With *Block-based* mode, having in mind the maximum sizes of I-frame, P-frame, and B-frame, the weighted size $\overline{Size}$ of a frame is calculated as in (12). Thereafter, the number of frames per each block (at the application level) is obtained as in (13). With this value, the period T, when the server has to send, is computed as in (14). The computing time of a block is constant while the time period depends on the client's petition. It is interesting to notice that if the number of frames per block becomes 1 then this method works well the same as the frame-based method.

$\overline{Size_i} = max_i(I)p_I + max_i(P)p_P + max_i(B)p_B$	(12)
$fpb_i = BlockSize\,/\,\overline{Size_i}$	(13)
$T_i = fpb_i\,/\,fps_i$	(14)

Thus, the Ui will be equal to *ctBlock/Ti*. As in (11), the *adaptation* depends on the knowledge of the contents of the next block to be sent. Thus, the adaptation equation for block-based mode is rewritten as in (15).

$T_i(t) =((1-\lambda)\, fpb_i(t-1) + \lambda\, fpb_i(t+1))/ fps_i$	(15)

- ***Average Policy with Frame-based mode***. In Average Policy with *Frame-based* mode, the ct is equal to the ct associated with the average sizes of frames (I,P,B). Thus (8a, 8b, 8c) are rewritten as follows:

$ct_i(I)=ct(avg(I))$	(16)
$ct_i(P)=ct(avg(P))$	(17)
$ct_i(B)=ct(avg(B))$	(18)

The translation for C and T is then equivalent to that exposed in (9) and (4) respectively. Since this policy is non-adaptive, its values are maintained during the entire service.

- **Average Policy with *Block-based* mode**. The *Block-based* mode takes into account the average (avg) sizes of I-frame, P-frame, and B-frame and then, it calculates the weighted size of a frame as in (19) then, *fpb* and T_i are computed as in (13) and (14) respectively.

$\overline{Size}_i = avg_i(I)\, p_I + avg_i(P)\, p_P + avg_i(B)\, p_B$	(19)

- ***Adaptive Average with Frame and block modes.*** Finally in Adaptive Average with *Frame and block modes*, at the beginning, the equations of the previous policies are used. The difference is that the values are adapted to the characteristics of the movie decreasing or increasing the QoS when the number of frames (I) decreases or increases. The *adaptive mechanism* for the frame-based mode is obtained from (10) and (11) while for block-based mode it is obtained from (15).

2.4.4. Network Broker

This broker applies the same policies used with the CPU broker. However, only the mode guided to frame-based makes sense since the network subsystem works with constant-size packets and is independent of the characteristics of the movie.

The parameters that the network broker must know are the following: Bandwidth (B), Data Unit Size of the protocol (MTU), peer-to-peer delay (D) and type of reliability (T). The type of reliability in our case is determined by the IP protocol that is a service guided to best-effort datagrams and does not offer any guarantee in the resource reservation. To make such reservation, it is necessary to compute the bandwidth for req_i (including headers overheads of the protocol) and the peer-to-peer latency in order to conduct the admission test.

To compute the necessary bandwidth ($Breq_i$) for req_i we have to calculate the packet rate ($PacketRate_i$) and the frame size ($FrameSize_i$) which has different values for each policy. With Maximum policy, the frame size is calculated as in (20). In Adaptive Maximum we use (11) to calculate it as in (21). In the Average policy, it is computed as in (22).

$$FrameSize_i = max(I) \tag{20}$$

$$FrameSize = max(I)\ p_i(I)+max(P)p_i(P)+max(B)\ p_i(B) \tag{21}$$

$$FrameSize_i = \overline{size}(I)p_i(I)+\overline{size}(P)\ p_i(P)+\overline{size}(B)\ p_i(B) \tag{22}$$

Regarding the adaptive average, the values of (22) are used then, (11) is applied to get the adaptive $FrameSize_i$. Thereafter, we calculate the theoretical bandwidth without including the protocol headers as follows:

$$PacketRate_i = (FrameSize_i\ /\ BlockSize\ )\ fps_i \tag{23}$$

where *BlockSize* is the network packet. Then, the real bandwidth is computed including protocols headers (e.g. TCP/IP). For these calculations, there was a need to know the real packet size (*BlockSize')* of the network. This value depends on the physical network characteristics and the protocols used. For testing the brokers, FastEthernet and TCP/IP (IPv4) have been considered. The protocol defines the value of MSS (Maximum Segment Size)=1526 bytes and the MTU (Maximum Transfer Unit) =1460 bytes thus,

$$BlockSize' = \lceil (BlockSize/1460*66) \rceil + BlockSize \tag{24}$$

$$Breq_i = PacketRate_i\ *\ BlockSize' \tag{25}$$

Thereafter, peer-to-peer latency is calculated to check whether the delay is too high for providing a requested QoS. The network broker executes a one-packet ping to compute this value. In WANs, this analysis would have to be increased. Finally, the Network Admission Test is executed as follows:

$$Breq_i + \sum_{j=1}^{N} Baccepted_j \leq NB \qquad (26)$$

where, NB is nominal bandwidth

2.4.5. Memory Broker

As a rule, the memory subsystem is very complex, however, just as in [NAH98], only the main memory is considered. The QoS parameter that we are interested in is the quantity of necessary memory needed to attend to *req*$_i$, and it is calculated as follows:

$Mem\_req_i = Number\ of\ packets * c$	(27)

where the number of packets is obtained by dividing the block size by the size of the network packet that will be sent over the network, and c is the buffer size in number of packets that has the functionality of reducing the jitter [VEN98]. The criterion to admit (or not) a petition from the perspective of memory subsystem is based on the following Admission Test:

$$Mem\_req_i + \sum_{j=1}^{N} Mem\_req_j \leq Available\ Memory \qquad (28)$$

If the test is true, *req*$_i$ is accepted and the RMM is informed.

2.4.5 Disk Broker

In the design of the storage server, the focus has been on providing continuous video streams and maximizing the number of streams that can be provided at the same time. To achieve these objectives, the design has concentrated on analyzing the data distribution on the disks and the policies implemented for data retrieving. The disk broker provides a mechanism for admission control in order to block the admission of requests that might affect the Quality of Service (QoS) and cause jittering at the client side. The implementation of the storage subsystem and the broker have been realized and presented as a joint project at our department [YAN03].

The disk broker determines and manages two resources of the storage system which are the bandwidth of the storage system and the available space for storing more video data.

As far as our work is concerned, the disk broker must reserve the necessary bandwidth during the video reproduction, and the storage server must provide a large number of streams and short response time. For the disk broker to do its work, it calculates the requested bandwidth based on the meta-information extracted from the video files. The meta-information includes the distribution of the frames, the size of each frame, and the variations between the bandwidth during the reproduction of the video.

Like other brokers, the disk broker adopts two approaches; the optimistic approach and the pessimistic approach. In the optimistic approach, the bandwidth is calculated based on the maximum size of I frames, while in the pessimistic approach, the bandwidth is calculated based on the average size of different frames.

In addition to the above approaches, the disk broker makes a predictive calculation of the amount of the necessary resources which are requested by the RMM based on the past history. In general, the Broker neither accepts nor rejects a petition based on the video contents only. Rather, it makes a constant monitoring and calculates the system throughput Th_i during a cycle *i*. From this parameter the adaptive average policy is applied to calculate the throughput as follows [YAN03]:

$$\text{Th} = \propto * \; Th_i + (1-\alpha) * \text{Th}, \text{ where } 0 < \propto \; < 1.$$

Another parameter needed is T which is the cycle time. The test that the broker uses to accept a request is:

$$\sum_{j=0}^{N} (FrameSize_j * fps_j * T) / Th_{disk} \leq T$$

From the point of view of the media delivery algorithm (CB_MDA), it is assumed that the storage server with the cooperation of the disk broker makes the video data available in an exchanging buffer from which, the video thread sends the data to the network according to the CB_MDA scheduling.

When the client decides to watch a specific video data, his request is analyzed and translated into system parameters by the RMM. The disk broker guarantees the requested bit rate per second. That is, the client’s request is translated into a bandwidth that has to be provided by the disk storage during a period of time. During this time, the bandwidth will be reserved all the time during the service. Therefore the CB_MDA expects two things from the storage system:

- Providing a short delay for data retrieving from the disk. The disk broker has achieved a delay of 2 to 3 microsecond. This delay does not affect the scheduling of the algorithm since it is close to zero delay.
- Maximizing the number of the video session that the disk can provide. This has been achieved by providing video data stripping among a set of disks (RAID system).

After incorporating the new disk storage, with little changes into the server architecture, the disk storage gave an almost zero delay (2 to 3 microseconds). This is done by making the video data available in a specific buffer for the algorithm delivery. However, we have faced some problems related to the new disk storage. The problem was that, for large files (long-lived streams) the system was interrupted in the middle of the service. Our interpretation is referred to some leak of memory that needed to be fixed.

2.5 Server Internal Communication

It becomes clear that the architecture of the proposed video proxy server (VPS) consists of a set of modules where each module may consist of sub-modules. These modules and sub-modules require some kind of communication and exchanging of messages between them. To understand the sequence of the communication and the flow of data and messages between the different parts of the server, the communications between these parts can be classified into four main categories as follows:

- Client-ACM communication
- ACM-RMM communication
- CB_MDA-RMM-Client communication
- Client-Client communication

Within each category, there are also local communications between the parts that compose each module. Figure 2.6 shows a complete picture of the communications between all parts that compose the video server and the client.

2.5.1 Client-ACM communication

This communication is a bidirectional communication between the server and the client. The contact is ignited by the client using sockets in order to communicate with the ACM. The ACM represents the entrance and the acquisition point through which all requests generated by the clients are received at this point. The requests can be requesting new video, invoking VCR commands, or renegotiation the QoS.

After delivering his request to the ACM, the client can continue his work or simply go to sleep until he receives an answer. Meanwhile, the ACM analyzes the received request and tries to satisfy it. Satisfying the request might involve the interference of other modules or sub-modules of the ACM. That is, the ACM might need to get in contact with other parts of the system in order to respond to the client's request. The type of the request determines what kind of communication is to be taken and at which level.

If the client is a new arrival, then the ACM assigns to it a thread in order to present to him initial services such as sending him a list of the available video files and negotiating with him the QoS. Meanwhile, the ACM goes back to listen to new requests or gets involved in doing other work such as contacting the RMM to investigate the resources availability. As a continuation, the ACM can always communicate with the thread through shared memories such as global variables and global structures for queuing the arriving requests and maintaining the negotiation status of each client. Meanwhile, the client continues his communication with the thread using sockets.

2.5.2. ACM-RMM communication

The communication between the ACM and RMM is done using sockets and, it works in client-server manner. That is, the ACM requests a service from the RMM and the RMM responds by presenting a service to the ACM. The ACM might request a report about the resources availability in general or check whether a request with specific QoS parameters can be accepted or not.

The RMM might have the information available and send them to the ACM or, it might need to get in contact with its brokers before it can give the response. All the communication between the ACM and the RMM are done by sockets. The reason for this is to give the possibility of having the ACM on machine A while the RMM is residing on machine B. This is done when there is a need to replicate the video server in order to distribute the service among a set of servers. In this case, the ACM will centralize the requests and decides to which server the request can forwarded.

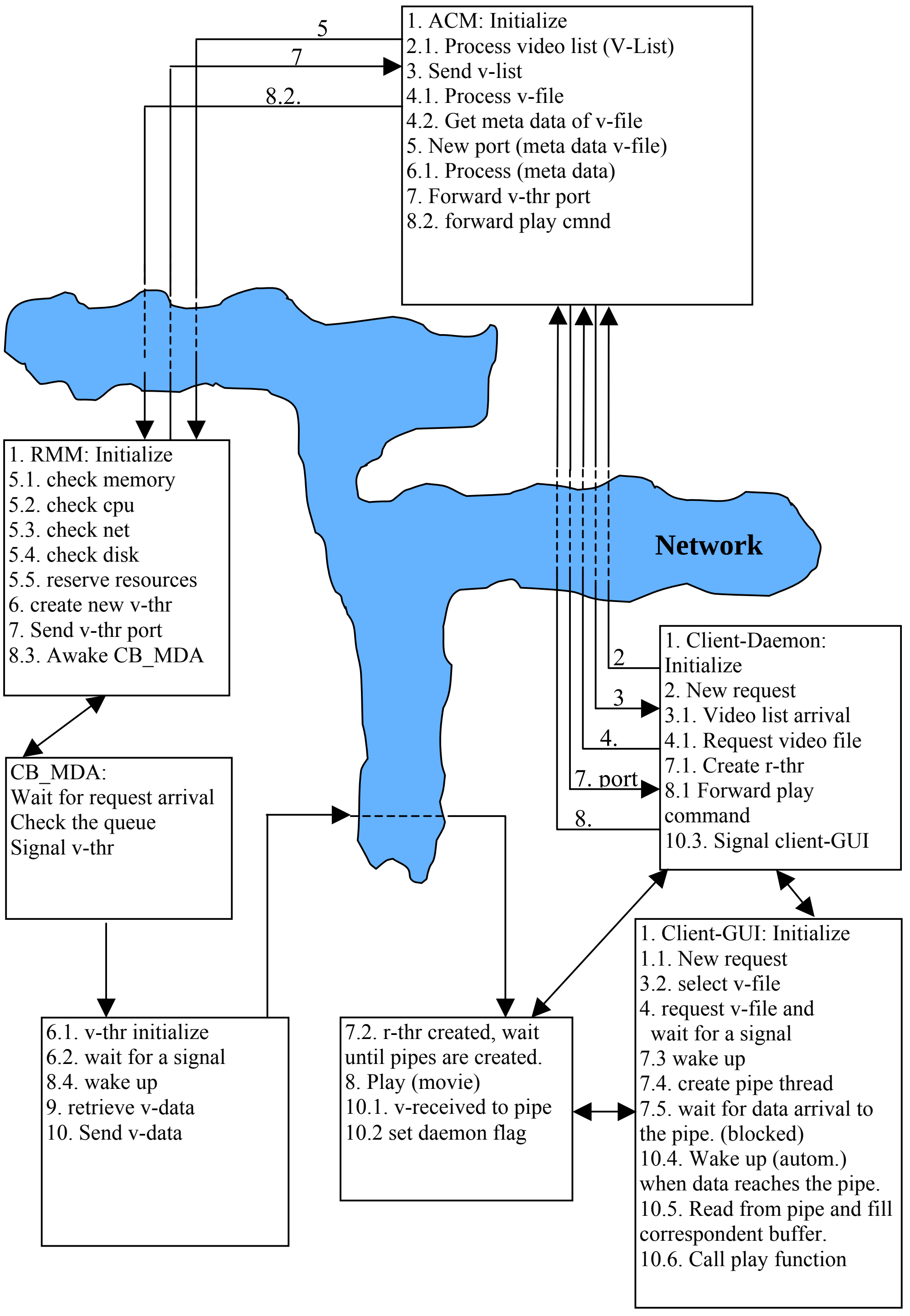


Figure 2.6. Sequence of communication between VPS modules

2.5.3. CB_MDA-RMM-Client communication

As it has been mentioned before, the CB_MDA is responsible for regulating the video streaming from the server to the clients. The details of this algorithm are presented in the next chapter. The implementation of the CB_MDA is done in a multithreaded fashion. The CB_MDA needs to get information from the RMM about the amount of the available resource in the system in order to schedule the delivery of the video data. This communication between the CB_MDA and the RMM is done by using global variables and structures.

With regard to the client, the client never gets a direct contact with the CB_MDA since all requests generated by the clients are forwarded by the ACM. There is only a one way contact from the RMM to the client in order to stream the video data from the media storage to the client's displaying machine. The clients command (e.g. pause, stop…etc) are received at the ACM first and then forwarded to the RMM and the CB_MDA in order to be processed. The client is informed about where to receive the video data from during the negotiation session between the ACM's thread and the client.

2.5.4. Client-Client communication

The client design and implementation is explained with details in appendix B. However, to complete the discussion about the communication, we only mention that, the client consists of three parts that communicate between each other. These parts are; the GUI, the communication module, and the player. The GUI allows the client to connect to the video server and other parts of the client. The communication module works as a communicating point between the video server and the client's GUI and player. The player displays the video data.

The communication between the client and the server is done by sockets. The communication between the clients' parts has been also achieved by using sockets. The reason for that was to present some kind of flexibility in case we decide to separate these parts from each others.

Chapter 3
Credit Based Media Delivery Algorithm

3.1 Introduction

As we mentioned before, a key issue in VoD service is the ability to stream continuous video data from the server to the clients across the network providing TVoD service without delay and without playback interruption (jitter). In this chapter we present a simple yet a starvation-free algorithm called Credit Based Media Delivery Algorithm (CB_MDA) [QAZ03b]. The algorithm is data-centered and implements client-initiated scheme (client-pull). The main objective of the CB_MDA is to provide TVoD service, short response time and scalability. Scalability is defined as the ability of the system to scale well as the number of the clients is increased. That is, the server will be able to cope with the increasing number of the clients without significant changes. To achieve these objectives, the CB_MDA uses multicast and unicast channels for transmitting the video data.

The unicast channels will be used as short ones to transmit the video prefix to a later arrival client while the client is merged to an appropriate multicast stream and buffering the media data from it. Also, the unicast channels will be used to serve interactive service such as the case when the client invokes any of the VCR-like functions. During the delivery process, the algorithm takes into consideration the server's resources and the client's buffer space and I/O bandwidth.

The algorithm provides video streams using means of patching and pre-fetching. Multicast channels will be created as a response to a request to a new none-active video file (a file that does not have any active session) or as a response to a request that could not be incorporated to an existing multicast session, so another session for the same video file is produced. The unicast channels will be used mainly to incorporate new requests into the server and transmit the initial part of the video (video prefix).

Our approach does not require the split of the channels into dedicated unicast and dedicated multicast channels to provide the VoD service. These channels are allocated dynamically during the service. Also, contrary to broadcasting and the classical batching techniques, the interactive VCR functions are provided and the video files are treated fairly without classifying them into popular and non-popular ones. This fairness comes from the fact that video files can be any thing other than movies. For example, in the educational centers such as universities, schools and other institutions, the clients (e.g. students) usually request old video files such as videos related to history, documents or experiments done in earlier years. Thus, these video files must not be ignored in favor of the recent 10 or 20 popular movies.

Another important issue we consider in this work is that, clients usually have different machines capabilities in terms of buffer space and I/O bandwidth. Therefore, our approach takes into consideration this fact and adapts itself to the heterogeneity of the clients who may join the multicast channels. The heterogeneity can have positive impact when the clients have high capacity in terms of buffer space and I/O bandwidth and the server has available resources that can be assigned to these clients. However, heterogeneity may have negative impact when clients with less capacity are merged into multicast streams that transmit at a higher capacity. This will require that the transmission rate be adjusted to an appropriate rate that matches the capacity of all clients joining the same multicast stream.

These problems are tackled by our proposed algorithm, which is called, Credit Based Media Delivery Algorithm (CB_MDA). In the next subsection we present the essence of the CB_MDA followed by its implementation. Thereafter, we present an example which clarifies the working principle of the algorithm. Due to the fact that requests can arrive at any time and the clients can be heterogeneous, in subsection 3.4 we explain how the algorithm handles these requests and adjusts dynamically the transmission rate of the streams.

3.2 The CB_MDA Algorithm

The essence of the CB_MDA is based on two principles: first, the clients' requests can be served through a combination of multicast and unicast channels, second, there will be periods of time during which the servers resources are not utilized, and there are heterogeneous clients whom their machines can cope with these resources. Therefore, the idea is to observe the resources availability and the clients who can receive video data at a rate greater than the movie reproduction (playback) rate and, then, assign these resources to those clients who can receive and cope with more video data in their buffers.

Of course, this requires that the CB_MDA has knowledge of the server's resources availability and the user's machine characteristics such as buffer space and I/O bandwidth. Before discussing the algorithm and for the sake of clarification, Table 3.1 contains some of the abbreviations which will be used throughout this chapter.

To achieve its goal, the CB_MDA translates the information generated by the RMM into *Credit Values* (CV) which will be assigned to the ongoing streams in the proxy server. These CVs make representation of the amount of the data that the algorithm can transmit to the appropriate clients' buffers during a period of time L, and they are ranging from 1 to V. The *Value* (V) represents the maximum credit value that the client can receive during *L*, and it is calculated in terms of the client's buffer space and I/O bandwidth. A *Credit Value* (CV) equal to 1 (CV=1) is interpreted differently from the point of view of the client and the server.

Table 3.1. The CB_MDA abbreviations

	Definition
L	Slot of time (e.g. 1 second) during which a set of video streams are scheduled.
CVs	Credit Value/s, makes representation of the amount of transmitted video data.
Ci	Client i
V	Max. credits that *C* i can receive during *Li*
CV	A credit value where, (1 ≤ CV ≤ V)
maxcv	Maximum credits value at slot *Li*
SiCV	The CV assigned to Stream *Si.*

From the client (player) point of view, a CV=1 means that the algorithm must make available in the client's buffer, during L, an amount of video equal to the Frames playback rate Per Second (*fps*) of the video. That is, the received video data is sufficient to make, at least, 1 second of normal playback during L.

In fact, this is equal to real-time play back rate. If the CV was less than 1, and the client's buffer was empty, then the client would definitely face video jitter.

From the server point of view, however, CV=1 represents the required resources (such as the CPU computing time, the size of the memory buffer, the network bandwidth, and the disk bandwidth) that guarantee the transmission of the amount of the data needed by the player to make a 1 second of video reproduction.

If a client *Ci* requests a video stream *Si*, the algorithm checks to see whether the video is active (that is there is an ongoing session/s for that video) or not. If the video is not active a multicast stream is initiated. However, if the requested video is being broadcast (in other words, it is active), the CB_MDA determines whether the client can join the multicast channel or a new multicast stream must be initiated.

If the client can be merged to a multicast channel, then the algorithm creates a unicast channel in order to transmit the video prefix to the clients' player. The decision when to initiate a new multicast channel requires a detailed investigation which is beyond of this work. However, we have conducted some experiments in this matter in order to explore a close to realistic result value that we can utilize to test our algorithm. The results indicated that a period of 10 to 15 minutes would give good results for a two-hour video file in terms of:

1. System throughput
2. Resources utilization

A period greater than 15 minutes would allow more users to join the multicast channels. However, this period would increase the number of unicast channels. Also, a value less than 10 minutes would increase the number of multicast channels. Consequently, the system will be less able to support interactive VCR functions and merge new clients to the multicast channels.

During the transmission of the media data, the CB_MDA does not distinguish between the multicast streams and unicast streams during the scheduling process. Whether the initiated stream is multicast or unicast, the CB_MDA assigns a CV to each stream where, CV is ranged from 1 to V.

The client whom his request invokes the stream initiation determines the maximum V of the stream based on his machines specifications such as the buffer space and the I/O bandwidth. This condition will be relaxed when we discuss the adaptation process of the algorithm when new clients who have less capability will be incorporated in multicast channels where the clients who have been joining these multicast channels were having greater capabilities.

The CV is also determined in terms of the credits availability in the server, and it can be increased or decreased dynamically during the service. A CV > 1 means that the algorithm is able to create a pre-fetched media data reserve in the client's buffer. The more the server can send video data ahead of its playback time the more the server can supply interactive requests when the system gets saturated. That is, if the CB_MDA reaches a point where all the CVs are assigned, and all streams have CV equal to one each, and a new user arrives or a client invokes a VCR action, the CB_MDA recollects the credits from the streams which have been previously assigned CVs > 1 in order to serve the new requests. In other words, some streams which have pre-fetch video reserve will be ignored for a while in subsequent scheduling slots in order to serve the new request.

After the initiation of the streams the server begins to transmit the video data into the network. Upon receiving the media data, the player starts playing back the video and, under normal circumstances, it continues the playback of the video unless it is interrupted by one of the VCR-like commands, which will be conveyed to the server.

The clients can be homogeneous or heterogeneous. They can be new arrivals or older clients who are being served. The algorithm explores these situations and tries to handle them accordingly.

3.3 The CB_MDA Implementation

Before and during the scheduling process, the CB_MDA needs information related to the requested video streams such as video size, frame rate, and reproduction time. This information, which is termed as Meta information, is created off-line for each video file and made available for the Admission Control Module (ACM) and the Resources Manager module (RMM). Thus, for each accepted request, the RMM provides the CB_MDA with this information. The CB_MDA, then, maintains a list of all active streams along with their Meta information.

Also, the algorithm needs to maintain information about the status of each stream during the scheduling process. Therefore, for each stream *Si,* it uses a set of variables (Table 3.2) which keep track of the scheduled streams and reflect the status of the scheduling process. These variables are summarized in Table 3.2.

Table 3.2. The CB_MDA variables

	Definition
L_i	Slot of time (e.g. 1 second).
S_i	An active stream
S_iCV	The Credit Value that the algorithm assigns to S_i during L.
S_iV	The maximum credit value that can be assigned to S_i
S_iTCV	Total Credits Value assigned to S_i up until L_i
S_iCCV	The total Consumed Credits Value as a result of the playback of S_i.
S_iPRV	The total Pre-fetch Reserve Value of the media data downloaded from S_i in the client's buffer.
S_iP	The time needed to Play S_i. It is obtained from Meta information of video files. (e.g. 3600 seconds).
S_iET	The Expected finishing Time of S_i. That is, the time when the server finishes transmitting the stream S_i
S_iF	The maximum number of frames in S_i
S_ifps	Frames rate Per Second of S_i

Moreover, the algorithm needs to know the maximum number of credits value that it can assign during a period of time *L*. This value is determined based on information received from the resources manager (RMM).

To start with the scheduling process, the algorithm divides time into slots of length *L*. In our implementation, *L* is determined in terms of the periodicity of the normal frames playback rate, which is equal to one second. In other words, the video characteristic is measured by how many frames are displayed per second. Therefore, the player needs to receive a number of frames per second (*fps*) in order to make a one second of video reproduction. The reproduced frames, based on the buffer size, can remain or be removed from the client's buffer.

During each slot L_i, the algorithm determines the maximum number of credits (maxcv) that it can assign during L_i and finds out the streams that can be scheduled during L_i as follows:

- The algorithm picks up the next stream S_i which has the smallest pre-fetch video data reserve S_iPRV in its buffer,
- It assigns to S_i a CV (termed S_iCV) where, $1 \le S_iCV \le S_iV$.

- It increments S_iTCV, which is the total credits value assigned to S_i, as in (1).

$$S_iTCV = S_iTCV + S_iCV \qquad (1)$$

- It increments S_iCCV, which represents the consumed credit value that would be consumed from S_i by the end of the slot L_i as a result of the video reproduction, as in (2). The S_iCCV is always incremented by 1 during normal playback since the CV of the video reproduction rate is always equal to one.

$$S_iCCV = S_iCCV + 1 \qquad (2)$$

- It calculates the amount of the pre-fetch reserve of S_i (S_iPRV) that would be in the client's buffer by the end of L_i, as in (3).

$$S_iPRV = S_iTCV - S_iCCV \qquad (3)$$

- And finally, it transmits the video data at a rate equals to the stream's credit value S_iCV.

The above steps, which are represented in figure 3.1, show the process of choosing only one stream for scheduling during L_i. However, within the same slot, the algorithm repeats this process over and over until no more streams can be scheduled and/or no more credits can be assigned.

During each iteration, and since the server has finite transmission rate, the algorithm tries to avoid over utilizing the server resources. It uses a variable (W), which keeps track of the number of credits assigned to the scheduled streams at each iteration during slot L_i. Thus, before the algorithm intents to assigns a CV to a stream S_i, it checks to see if

$$W + S_iCV < maxcv$$

If the condition holds, Si is scheduled and W is incremented as in (4).

$$W = W + S_iCV \qquad (4)$$

At the end of each slot, W is reset to zero.

Likewise, the client has finite buffer space and I/O bandwidth. Therefore, the algorithm makes sure that the transmission neither overflows the client's buffer nor over-saturates the client's I/O bandwidth during each slot. To achieve these goals the following two conditions must hold before the transmission can take place:

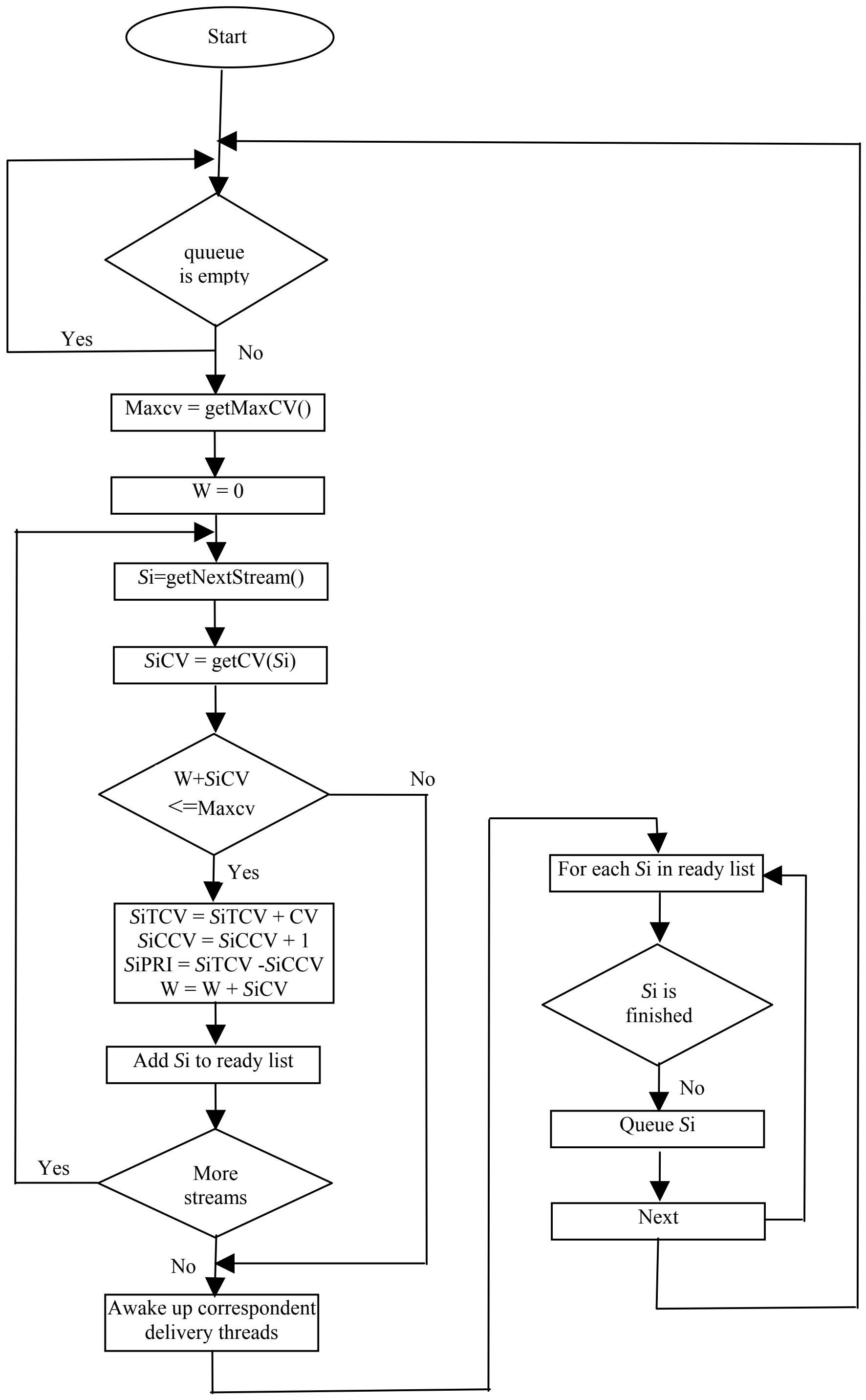


Figure 3.1. CB_MDA flowchart

$$S_iCV \leq S_iV \text{ and}$$

$$(S_iPRV + S_iCV - 1) \leq S_ibuff$$

The first condition is always guaranteed since the CB_MDA has knowledge of the client's buffer space and I/O bandwidth and, thus, never assigns to S_i a CV > S_iV. The second condition, and to simplify the discussion, assumes that the reproduced frames are removed from the buffer. Therefore, it considers the pre-fetched reserve of S_i (S_iPRV), the current assigned CV of S_i, and the consumed value *CCV*, which is always one during normal playback, in its calculations. If both conditions hold, the stream is scheduled. If the second condition does not hold, then the CB_MDA tries to reduce the S_iCV until the second condition holds or $S_iCV = 1$. If S_iCV reaches 1, at this point, the S_iCV will become 1 in the subsequent slot/s since the buffer has become full. This means that, the transmission rate will be equal to the playback rate during the next slots.

Another policy that can be implemented is to leave the consumed frames in the buffer since the client has the habit of reviewing the last few seconds. Therefore, the second condition can be rewritten as follows:

$$S_iCCV + S_iPRV + S_iCV \leq S_ibuff \text{ or simply}$$

$$S_iTCV + S_iCV \leq S_ibuff, \text{ since } S_iTCV = S_iPRV + S_iCCV.$$

If the above condition does not hold the algorithm must overwrite the video data that has already been played back. The algorithm can keep overwriting the played back video data until S_iPRV is equal to $S_i$buff. This is possible if the client pause playing the video while the server continues sending the video data.

The CB_MDA can calculate the expected finishing time of a video stream (S_iET) during any slot L_i since it can calculate the playing time (S_iP) of each video stream (e.g. 3600 seconds). This is needed when the number of active streams becomes larger than maxcv, so the algorithm tries to create a new stream but all CVs are assigned to all streams. Therefore, it calculates the finishing time of the streams to see which one is close to finish so that its credits can be reassigned to the new request. The playing time of S_i (S_iP) is obtained from the Meta information of video files and calculated as follows:

$$S_iP = S_iF \,/\, S_ifps \qquad (5)$$

Where S_iF is the maximum number of frames in S_i and, S_ifps is the frame rate per second in S_i. Thus, the S_iET is calculated as follows:

$$S_iET = S_iP - S_iTCV \qquad (6)$$

Based on S_iET, the server can know when the streams are finishing and, thus, decide whether to accept more requests or not.

The admission test is carried out by the CB_MDA when the number of the active streams exceeds the maximum CVs (maxcv) in order to protect the QoS of the ones who are being served. The streams which exceed the maximum CVs are called Extra Streams (ES). The admission control consists of calculating whether the new stream causes starvation in any of the existing streams. The calculation is done as follows:

For the first extra stream, it calculates the number of slots which are needed before the shortest stream (the closest stream to the finishing time) can be finished, then it calculates the total pre-fetched reserve value (PRV) in the clients buffers. Based on these calculations, the admission test is carried out as follows:

((Total PRV+ ES)/ (ES)) >= (total required slots/maxcv+ES)

If this condition holds, a new stream will be generated. For the next extra stream, it calculates the number of slots which are needed before the next shortest stream (the next closest stream to the finishing time) can be finished, then it calculates the total pre-fetched reserve value (PRV) in the clients buffers. Thereafter, the admission test is applied. This process is repeated for each extra client who causes the initiation of a new stream.

Case Study 1:

To clarify the algorithm more, and without loss of generality, suppose that the algorithm can assign up to 5 credits (maxcv = 5) during a slot and, in an instant during the service, say by the end of slot L_i, we have four streams *S*1, *S*2, *S3* and *S4*, each with a CV, buffer space (buff) and I/O bandwidth as elaborated in Table 3.3.

Table 3.3. Initial values at slot *Li*

	S1	S2	S3	S4
	CV=1	CV=2	CV=2	CV=0
buff	20	10	10	20
I/O	1	2	3	2
	PRV=1	PRV=9	PRV=8	PRV=11

As the Table 3.3 indicates, during slot L_i, *S1*, *S2, S3, and S4,* each has a CV equal to 1, 2, 2, and 0 respectively. The CV of each stream is subjected to change dynamically during subsequent slots. The buffer space (buff) of the clients connected to *S1*, S2, *S3,* and *S4* can take up to, as a maximum, 20, 10, 10, and 20 credits respectively. The I/O bandwidth of the clients receiving from *S1*, *S2*, *S3,* and *S4* is 1, 2, 3 and 2 respectively. The buff and I/O values are fixed since they represent the clients' machines specifications.

Also, suppose that the pre-fetch reserve values (PRV) by the end of L_i of streams *S1*, *S2, S3,* and *S4* in the clients' buffers are 1, 9, 8 and 11 respectively (Table 3.3).

Table 3.4 shows the status of the streams after the subsequent scheduling during slot (L_i+1). The table shows the following things:

- It shows which stream has been chosen at each iteration.
- It shows how the PRV of each stream has changed and
- It shows how the variable W, which is reset to 0 at the beginning of each slot, keeps track of the credits assigned during the slot.

The following steps clarify how the algorithm has scheduled the streams of table 3.4 taking into consideration the credits availability in the server (5 credits in our case) and the clients' machine characteristics (see Buff and I/O in table 3.3).

Table 3.4. Streams scheduling during *Li+1*

	S1	S2	S3	S4	w
Iteration1	CV=1	-	-	-	1
Iteration2	-	-	CV=3	-	4
Iteration3	-	CV=1	-	-	5
	PRV=1	PRV=9	PRV=10	PRV=10	

The scheduling of the subsequent slot (in our example L_i+1) is always based on the result of the previous slot (in our example *L*):

- During L_i+1, the CB_MDA finds the stream with the lowest PRV from table 3.3. This table indicates that S1 has the lowest none scheduled stream. Therefore S1 is chosen, and the algorithm assigns to S1 a CV equal to 1 (CV=1) since the I/O of S1 can't take more than one credit. Then, the algorithm increments the PRV of S1 and the global variable W by the value of the CV which is equal to 1, then it decrements PRV by 1 since the playback consumption rate is always equal to 1. Thus the PRV of S1 remains 1 and W becomes 1 (table 3.4). This process called the 1st iteration (*iteration1*), and is shown in table 3.4.

- The algorithm, still, has four more credits that can be assigned. Therefore, it checks to see if there are other streams that can be scheduled.
- At the 2nd iteration (*iteration2*), the CB_MDA again looks at table 3.3 in order to pick up the next stream, S2, S3 or S4. Following the same procedure, table 3.4 shows that S3 has a lower PRV than that of S2 and S4. Therefore, S3 is scheduled, and the algorithm assigns to S3 a CV equal to 3 (CV=3) since this value neither overflows the buffer space nor saturate the I/O of S3. The PRV of S3 is incremented by 3 and then decremented by 1, and W is incremented bye 3 too. Thus, during slot Li+1 the PRV of S3 becomes 10 and W becomes 4 (see table 3.4, *it2*).
- Two more credits can be assigned, so the algorithm repeats the same process and finds out that S2 has a lower PRV than that of S4. Thus, S2 is selected to be scheduled. Although, S2 can receive 2 credits, however the algorithm assigns to S2 only one credit (CV=1) since it can’t assign more credits than maxcv (W becomes equal to maxcv). Then, it increments the PRV of S2 and decrements it by 1, thus, it becomes (remains) 9, and increments W which becomes 5. Table 3.4 shows this last iteration (*iteration3*).
- All the credits have already been assigned during this slot (*L_i+1*). S4 is the only one that has not been scheduled. However, the algorithm decrement the PRV of S4 by 1 since by the end of each slot, a credit value equal to one is consumed from all ongoing streams. Therefore, all ongoing streams must be decremented.
- In the above example, in order to ease the understanding of the scheduling process, we were decrementing the PRV of each stream at the end of each iteration. In fact, this process is done by the end of each slot. That is, the algorithm runs through all ongoing streams which are in the queue and decrements their PRV by one.

Case study 2:

To further show how the algorithm distributes the credits among the streams, we will go through the algorithm during *Li+2*. Table 3.5 shows how the streams have been scheduled during this slot by showing how the values are changed during the different iterations. Following the same procedure as above, the scheduling is done as follows:

- The algorithm uses the PRV of the streams from Table 3.4 (slot *L_i+1*) to decide which stream to choose.
- W is reset to 0 (W = 0).
- S1 is scheduled first since it has the lowest PRV, and a CV=1 is assigned to it. Thus, the PRV of S1 becomes 1, this value is calculated as follows:

  (1+1-1),

  and W becomes 1 too since (W=W+CV) (Table 3.5, at *Iteration1*).
- At *Iteration3*, S2 is scheduled next, its CV=2, its PRV becomes 10, (9+2-1), and W becomes 3.
- Two more credits are left.
- S3 and S4 have the same PRV. According to their order in the queue, S3 is chosen at *Iteration3*. Notice that the algorithm assigns only one credit (CV=1) to S3 since any value greater than one would overflow the client's buffer of S3. The PRV of S3 becomes 10, and W becomes 4.
- Finally, the algorithm chooses S4 at *Iteration4*, and assigns to it a CV equal to 1 since W becomes 5 (W = maxcv).

Table 3.5. Streams scheduling at *Li+2*

	S1	S2	S3	S4	W
Iteration1	CV=1	-	-	-	1
Iteration2	-	CV=2	-	-	3
Iteration3	-	-	CV=1	-	4
Iteration4	-	-	-	CV=1	5
	PRV=1	PRV=10	PRV=10	PRV=10	

From the above examples we can reach the following conclusions:

1. The algorithm protects the overflow of the clients' buffers. This is done by verifying that PRV+CV-1 ≤ client's buff.

2. It does saturate neither the server nor the clients' I/O bandwidths. This is clear since the algorithm verifies that W ≤ maxcv, and the CV ≤ the client's I/O.
3. It distributes the CV dynamically, and this is important when new request are accepted by the server or some clients finish the service.
4. It can create a pre-fetch reserve (work ahead) in the clients' buffers.
5. It maintains a balanced pre-fetch reserve in the clients' buffers since the client with the lowest PRV is selected.

Although, during the media data delivery, some streams are no longer able to receive more than what it can be consumed, these streams might have had pre-fetch reserve in their buffers which can be recollected. That is, during urgent situations and when there are no alternatives, these streams can be ignored for a period of time equal to their PRVs. Also, if the normal playback does not get interrupted by any of VCR commands, the service time for the streams will be reduced by their pre-fetched values. This way, some clients can get finished before the time that would be needed if the stream were transmitted at real-time rate, new requests/clients enter the system and, thus, the throughput of the system is increased.

During the VoD service, new clients might arrive to the systems and others may leave the systems. Some clients may invoke VCR functions such as jump forward or jump backward and others may simply pause the playback. These situations require the algorithm to be able to adapt itself to these changes so that:

1. New clients can be incorporated in the service.
2. Interactive VCR actions can be supplied.
3. Accepting any of the above action does not interrupt other client being in the service
4. The QoS contracted with the clients being served is maintained or negotiated as it has been agreed upon.

In the following discussion, we show how the algorithm handles these situations and how it incorporates itself to the changes.

3.4 Dynamic Adaptation

During the delivery service new requests might arrive and/or old users might leave the system. The requests could be requests for a new video data that cause the initiation of new streams, requests for video data that has ongoing streams, or commands for VCR functions

In this section we describe how the algorithm adjusts the delivery service based on the type of the requests made. In general, the requests can indicate one of the following situations: New user arrival, User departure (leaving the system) or Invocation of VCR functions.

3.4.1. New User Arrival

The arrival of new users is not necessarily causing redistribution of the system credits. It depends on whether the requested stream is active or not, and whether the request arrives at the beginning of the stream initiation or at some time later. Therefore, we show how the algorithm handles each case without violating the continuity of other streams being in service. In general, there are two cases to be considered:

Case 1: Request for active stream

In this case, the client is asking for video data which is about to be streamed or have already been streamed. That is, if the request arrives at the beginning of the stream initiation, then the new request can be incorporated in this stream and the video data will be multicast to both users.

Clearly, this situation causes positive impact since the same resources are used to serve more than one user at the same time, and thus more performance is achieved. If the request, however, arrives some time after the initiation of the stream, then, the algorithm applies patching technique where the video data from the arrival point of the request upward is patched and the video prefix is served using unicast channel.

If the server has plenty of credits, then the other ongoing streams will not be affected by the new requests. Otherwise, the algorithm will make a redistribution of the credits. If the algorithm can manage to assign a CV greater than one to the unicast channel, then the usage time of the unicast channel is reduced leading to the release of the unicast channel earlier than the time that would be needed if the data were sent at real-time rate.

Case 2: Request for a none-active stream

If the client is requesting a non-active stream, then a new stream must be initiated. The algorithm checks to see whether it has enough resources in order to grant them to the new stream or make a redistribution of the resources. To clarify the credits assignment, suppose that the system status is as it appears in

Table 3.6a, and the algorithm can assign up to 5 credits and, suppose that a new client requests a new stream S4 with buffer space and bandwidth specifications equal to 20 and 2 respectively, then the algorithm must redistribute the credits of S3 in favor of S4 or ignore S2 and S3 in subsequent slots, since they have pre-fetched reserve in order to satisfy the new request.

Following the principle work of the algorithm, the credits will be redistributed and S4 will be, at the beginning, scheduled repeatedly since it has the lowest PRV (Table3.6b).Table 3.6b indicates that for the next subsequent slots, S4 will be scheduled most frequently since it has the lowest PRV.

Table 3.6a. Streams status after S4 arrival at *Li*

	S1	S2	S3	S4
buff	20	10	30	20
I/O	1	2	3	2
	PRV=1	PRV=9	PRV=9	PRV=0

Table 3.6b. Streams scheduling at *Li*+1

	S1	S2	S3	S4	w
it-1	-	-	-	CV=2	2
it-2	CV=1	-	-	-	3
it-3	-	CV=1	-	-	4
it-4	-	-	CV=1	-	5
	PRV=1	PRV=10	PRV=10	PRV=1	

3.4.2. User Departure (leaving the system)

When a user finishes the playback or quits the system for a reason or another then, depending on whether the user disjoins a multicast or a unicast channel, there is a possibility of having some released credits. These credits can be assigned to other streams.

Case 1: Quitting a multicast channel

Quitting the service can be temporary (such as pause command) or permanent such as stopping the service or finishing receiving the necessary video data. If the client invokes the pause command then two scenarios can be implemented as follows:

- The first scenario is to let the server continues delivering the video data until the client's buffer becomes full, then the client will disjoin the stream. As a continuation, when the client decides to continue the reproduction of the video the server gets informed about that and tries to join him with the same stream implementing patching or, to join him with an appropriate active stream which transmit the same video content.
- The second scenario is simply to disjoin the client from the stream. Thereafter, when the client invokes the play command the server tries to rejoin him with the same stream or an appropriate one.

If a user makes an early quit of a multicast channel and he is not the only one connected to that channel, then his quit does not have neither positive nor negative impact on the server's credits. That is, his quit does require resources and does not release the multicast channel and thus, the credits will remain assigned to the multicast channel until it is finished or all the clients are quitting the multicast channel.

Case 2: Quitting unicast channels

In fact, unicast channels are allocated as short channels for providing the video prefix when the client joins a multicast channel. Therefore, quitting or releasing unicast channels can happen for two reasons:

- The first reason is that, the server has finished the transmission of the video prefix and thus, the channel is released and its credits can be redistributed among others.
- The second reason is that, the client is making an early quit of the service. In this case, the same scenarios which have been mentioned in the previous case can be considered in order to see how to adapt to the client behavior and decide whether to fill the client's buffer or not.

In general, quitting unicast channels always leave some available credits. Therefore, these credits can be distributed among other streams so that, the pre-fetched reserve is increased in the clients' buffers.

Increasing the pre-fetched reserve causes a bigger distance between the delivery time of a movie and its deadline. Bigger distance means that the client is less dependent on the system. Thus the system finishes servicing some users before the time that would be required if the credit values were equal to 1 all the time.

3.4.3. VCR-Functions

The algorithm allows VCR-Like control such as pause, forward and backward temporal jumps. These interactive actions are performed with minimal delay. VCR-functions might have positive, neutral or negative impact on the performance.

Whenever a user invokes an interactive action, the client sends a message indicating the command invoked (stop, pause, forward, backward) to the server. Suppose that a client who is connected to stream S_i pauses the movie. Upon receiving the VCR command, the server checks to see whether S_i is a multicast or unicast stream. If S_i is a unicast stream, the server can put on hold S_i until a different command is received or simply continue streaming the data until the client's buffer is full or the video prefix is totally transmitted.

However, if S_i is a multicast channel then, the user disjoins the multicast S_i while S_i keeps going on in order to serve the other members of S_i. Meanwhile, the disjoined client's buffer status is kept at its level. A different policy would be to continue delivering the video data until the client's buffer is full.

Suppose that the user makes a forward temporal jump equal to *n credits*. If $(TCV+n) < TCV+PRV$ then the algorithm discard n credits from the head of the buffer and the PRV is set to (PRV = $PRV - n$) and the CCV is set to ($CCV = CCV + n$). *If* $n >= PRV$ then all the data will be discard from the buffer and the PRV is set to 0 (PRV = 0) and the *CCV* is set to (*CCV* = CCV + n).

It is important to always keep track of the *TCV* in order to calculate the finishing time. This way, the algorithm might accept new requests with the hope that some users will disjoin soon.

Finally suppose that the user makes a backward temporal jump. Likewise, the algorithm checks to see whether the jump is within the buffered data or not. If jump equal to –n is within the buffered data then PRV = PRV – (-n), else all pre-fetched data are simply discard and the PRV set to zero (PRV = 0) and the values of TCV and CCV are updated accordingly.

In term of the impact of each command on the server, pausing of streams actually improve the performance since the data can be accumulated in the user's buffer and other active streams can be assigned more resources increasing their *CVS* and PRVs during subsequent slots.

Frequent forward commands degrade the performance since pre-fetch buffers would be frequently set to zero. Leaving some of the played back video in the buffer can reduce the impact of backward jump. This can be noticed during the peak time when the server cannot assign a CV > 1. In this case, the server will never be able to have the buffers full. Thus, it is much better to leave the portion of the played movie since the client has the tendency of replaying over and over again the last few seconds of the played video.

3.4.4. Reducing Unicast Usage Time

The objective this subsection is to utilize the server resources as much as possible. In the final implementation, multicasting has been the choice in order to increase the server performance in terms of the number of the clients that can be accepted for the service. Again, the CB_MDA presented the solution by implementing patching and pre-fetching techniques. The idea, as it has been mentioned above is to have the later arrivals to join existing multicast sessions if the requested video contents matches the contents of the multicast sessions. At the same time, the client should receive the video prefix. This is can be done by employing a unicast channel for sending the video prefix.

This indicates that, the unicast channels are becoming a critical issue. That is, the server should always have unicast channels so that the later arrivals can join the active sessions. Therefore, these unicast channels must be allocated and released as soon as possible. To solve this problem, the algorithm utilizes the server resources and sends work-ahead data over the unicast channels in order to reduce the usage time of these channels.

Figure 3.2 and 3.3 show the implementation and the time needed for transmitting the video prefix over unicast channels without pre-fetching and then with pre-fetching respectively. Figure 3.2 shows the patching technique with which a video server periodically "broadcasts" a video object via a number of dedicated multicast channels. If a new client joins a multicast channel then, he is allocated a unicast channel in order to start the playback of the missing initial video data (video prefix). Meanwhile, the data from the multicast channel is patched and stored into the user's buffer figure 3.2a. After finishing playing back the video prefix, the unicast channel is released and the player continues playing back from the buffer, figure 3.2b. Of course "Patching" assumes that the client has sufficient I/O bandwidth and buffer space to receive from multiple channels and to buffer the video data.

Figure 3.2a shows that the unicast channel will be occupied during the time ***t*** in order to download the initial part of the video and, at time ***2t***, figure 3.2b, the unicast channel will be released. This shows that unicast channels play an important role in the success of "Patching". Therefore, it is important to reduce the usage time of the unicast channels whenever it is possible in order to serve other new requests and/or VCR-like functions. The Credit-Based Media Delivery Algorithm (CB_MDA) which schedules the data delivery can be used to reduce the usage time of the unicast channels.

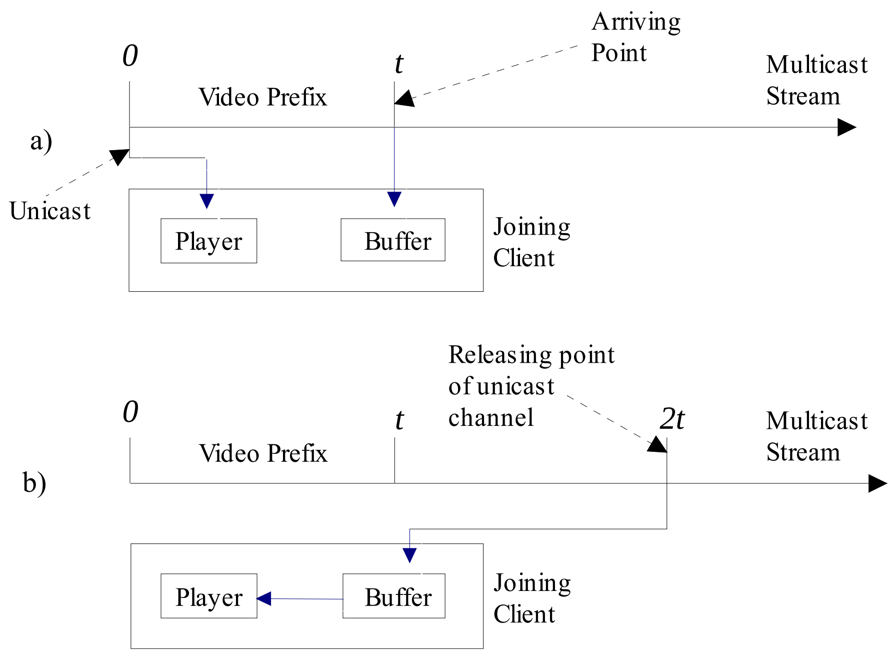


Figure 3.2. Multicasting with classical patching
a) Client arrival, b) unicast channel release

In the implementation, it is proposed another buffer called Unicast Buffer (Ubuffer) which will be receiving the video prefix at a rate greater than the playback rate (figure 3.3). When a new client (request) is arrived, first, the user, as in patching, is allowed to join a multicast channel and, the video data received from that channel is stored in the client's Mbuffer, at the same time, the user is allocated a unicast channel with a data rate equal to "*icv*" where "*icv*" is an initial credit value defined by the CB_MDA, and when "*icv*" is greater than 1 , the data rate is greater than the playback rate, second, the video prefix goes, first, to a unicast buffer (Ubuffer) and then to the player as it is shown in figure 3.3a. Thus, the time needed for transmitting the video prefix is *t/icv* (figure 3.3b), and the unicast channel is released at *t+t/icv*. The player, however, continues playing back from the Ubuffer. Only at *2t* (figure 3.3c) the Ubuffer is released since the playback rate is always equal to credit value "1", and the player then continues playing the video from Mbuffer. It can be noticed that, when the value of *icv* > 1 then the time needed for downloading the video prefix will be reduced for a period of time equal to *2t – (t + t/icv)*.

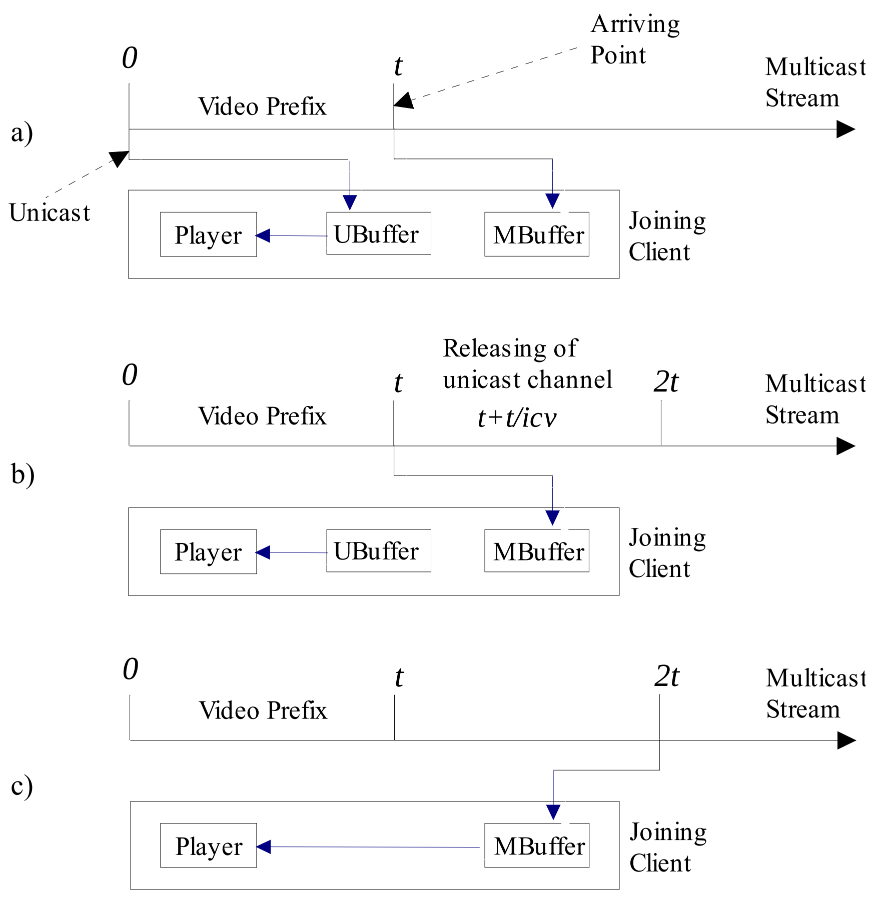


Figure 3.3 Multicasting with CB_MDA patching

a) Client arrival, b) Unicast release, d) UBuffer release

These techniques were important for providing multicast transmission, incorporating new clients in the multicast session, and creating work-ahead video in the client's buffers. The experiments in chapter four have shown that the CB_MDA outperforms other delivery algorithms such as the batching and the classical patching, as well as the periodic broadcasting techniques. This performance is measured in terms of the response time, the number of request served by the server, and the implementation of VCR-Like commands.

Chapter 4

Implementation and Experiments

4.1. Introduction

The implementation of the Video Proxy Server has been done by integrating the different parts of the system in order to examine the overall behavior of the server. This chapter demonstrates the implementation of the Video Proxy Server (VPS) followed by the experiments conducted for the main parts of the architecture.

These parts include the resources manager and the admission control policies as well as the delivery algorithm (CB_MDA). To check the viability of the different parts of the system, a series of experiments have been organized and conducted in stages.

In the first stage, the experiments measure the time needed for retrieving the data from the disk, and they measure the round trip time needed for attending to a request in order to measure the response time. Thereafter, the experiments go one level up in order to test the admission control policies (Maximum Policy, Adaptive Maximum Policy, Average Policy, and Adaptive Average Policy) which are implemented in the brokers of the resource manager. The experiments of this level reflect the necessary resources which will be needed for serving a specific request. These resources are the CPU, the Memory, the Network, and the Disk.

At the top of the architecture sits the CB_MDA algorithm which will be tested exhaustively since it is responsible for scheduling and regulating the flow of the video data from the server to the clients.

4.2 The VPS implementation and Experimental Study

As it has been mentioned in chapter two, the architecture is based on a set of modules. These modules are: the admission control which works as a negotiating and acquisition as well as admission module, the resources manager which employs four brokers for the resources reservation, and the delivery algorithm (CB_MDA) which schedules and regulates the streaming of the video data from the server to the client.

The admission control and the resources managers have been created as standalone programs, and the communication between them has been done by using sockets programming. The reason for the separation has been done based on the following reasons:

a. Providing scalability. This means that, if there is a need for the server to extend its operations and include multiple servers, then this is guaranteed without making significant changes in the admission control and the resources manager programs.

b. Allowing the admission control module to play a central role in controlling all requests received from the clients.

4.2.1 Development steps

The implementation of the server has been created from scratch and it has been achieved in stages. The reason for creating it from scratch rather than using built in functions and tools refer to the fact that this server will work as a building block for the Distributed VoD (DVoD) architecture which has been explained in chapter two. Therefore, there is always a need for testing the effect of each part on the over all design of the architecture, which requires the availability of the source code. In addition, our code gives us the flexibility to face any changes and improvements that might be needed in the future such as adding new modules, modifying some of the modules, and/or replacing one of the modules by another one. The major programs of the VPS cover the following parts:

- The acquisition of requests and the negotiation with the clients.
- The admission control policies for resources management.
- The scheduler of media data delivery (CB_MDA) and the delivery threads which are responsible for delivering the video data.
- Network communications with multicast and unicast.

- A client program which requests a service and synchronizes with the server in order to receive from multicast and unicast streams.

At the beginning, we have studied the characteristics of the video format and the transmission of the video from the server to the client. This study has shown that the VoD applications require long-live sessions with high system resources. Also, it has shown that the clients expect to get the service in a very short period of time. Consequently, and to start with, the concentration has become on the response time followed by the problem of the server resources. Thus, the first task has become exploring and guaranteeing a short response time. The response time has been classified as of two types:

- Connection response time. It is the time needed to establish a connection with the client for negotiation.
- Delivery response time. It is the time needed to deliver the required video data when a client selects a video file and/or invokes a VCR-like function.

In both cases the response time has to be as short as possible. The first problem is solved by implementing multiple threads in order to negotiate with the clients. The second problem, however, is discussed with details in the experiments of next stages.

As a continuation, the first task which has been achieved was creating a module which worked as an entrance and admission point for the arrival of all requests. This module is called Admission Control Module (ACM) and its preliminary tasks were:

- Listening at a specific port for the arrival of new requests.
- Analyzing these requests and then,
- Serving them.

However, a problem has occurred when a new request arrives while the previous one is in service. The new request had to be queued until the first one is done. To solve this problem, the ACM uses multiple threads for serving the arriving requests. That is, when a request arrives, the module (as a father) assigns a thread to the new request in order to negotiate with it while the previous request is being served. The process of communication between the client and the server has been done as follows:

1. The client connects to the server.
2. The ACM realizes that this client is a new one and creates a new thread for serving him and goes to listening state.

3. The thread negotiates with the client. In case of no confirmation the thread informs the ACM and dies.
4. In case of confirmation, the thread also informs the ACM and dies, then
5. The ACM creates another thread for retrieving the video data and transmitting it to the client.

4.2.2 Stage One: Preliminary Architecture and Response Time

In this stage, and as a preliminary architecture, we have created a multithreaded ACM, a multithreaded client, and a reader thread for reading and transmitting the video data to the client. The client, so far, was built in text mode with two threads. One of the threads was dedicated for receiving the video data, and the other was dedicated for invoking VCR commands such as Play, Fast Forward, Slow forward, Pause, and Stop.

To validate this architecture and to verify its viability in a real system, a series of tests have been accomplished in a real environment based on clients/server module. The real framework (VoD server) is based on Linux OS (SuSE, Mandrake, and RedHat Distributions) and a set of clients who are independent of the OS, but for the testing purposes they also have used Linux based machines.

With regard to the network, a 100Mbits local area network (LAN) has been used with an average workload of 25% to evaluate the performance of the service under unfavorable conditions. This criterion has been chosen to extract conclusions with respect to the server installation in systems with very high load.

The first series of experiments have been carried out in order to calculate the Round Trip Time (RTT) which is defined as the time elapsed from the moment the client requests a video file until the media data (or fragment of it) arrives to the client. These experiments have been carried out with 2 clients, 4 clients, 8 clients, 16 clients, and 32 clients running on different PCs. These clients have been served by sending to them packets of 1KB (KiloByte), 2KB, 4KB, 8Kb, 16KB... and 100KB. The response time has been calculated for different combinations of clients in different PCs for different file sizes with different packet sizes from 1KB to 100KB. The experiments have shown that a client from the group of 16 clients with a file size of 95KB required a Round Trip Time of 10,9 milliseconds (for each client).

Other groups of experiments have been conducted for the same reason (figure 4.1). The first group includes 8 clients, the second group includes 16 clients, and the third group includes 32 clients. The first group was distributed into two processors (PCs) where four clients were running on processor A requesting file I, and the other four clients were running on processor B and requesting file II. Thereafter, the number of clients has been increased to 16 where, 8 clients were running on processor A requesting file I, and the other 8 clients were running on processor B requesting file II. Then, the number of the clients has been increased again to 32 clients where, 8 clients were running on processor A requesting file I, and another 8 were running on the same processor but requesting file II. Then, another 8 clients were running on processor B requesting file III, and the other 8 were running on the same processor but requesting file IV.

Figure 4.1 shows the system stability (Server thread - Client thread) measuring the round trip time for sending 32 Kbytes multimedia packets for different configurations of clients and different files. Also, it shows that the response time was fluctuating between 25 ms to 40ms. This time would be acceptable for two reasons; first, it does not violate the real-time requirements where 1Mb (MegaBits) to 4.5 Mb must be present at the client buffer so that he can make 1 second of video reproduction, second, this time is considered as small enough for providing short response time during the negotiation, which does not involve transferring large data, between the client and server.

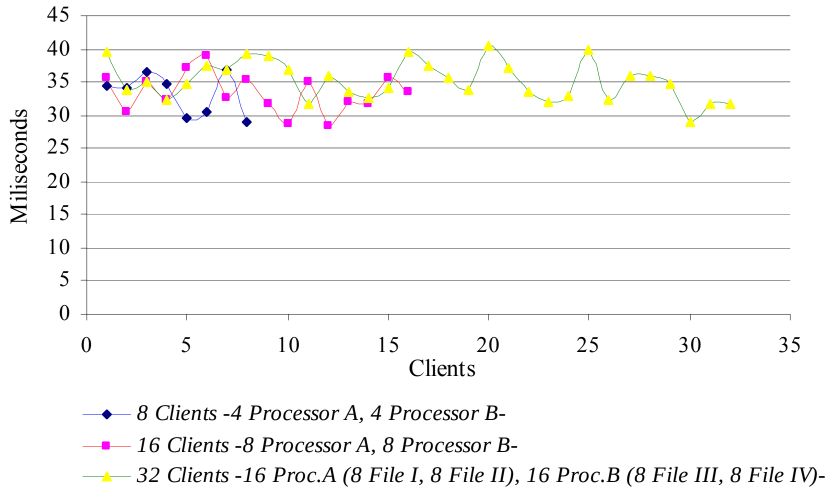


Figure 4.1. Average Round Trip Time for 32kbytes packet size

Figure 4.2 shows the result of executing (300 times for each configuration) different clients requesting the same and different files with the server sending 1,3 Mbytes per second for each client. A set of statistical values has been generated for the Round Trip Time (minimal and maximum value, standard deviation, geometric average and mean deviation) valuing the capacity of the VoD Server and the clients.

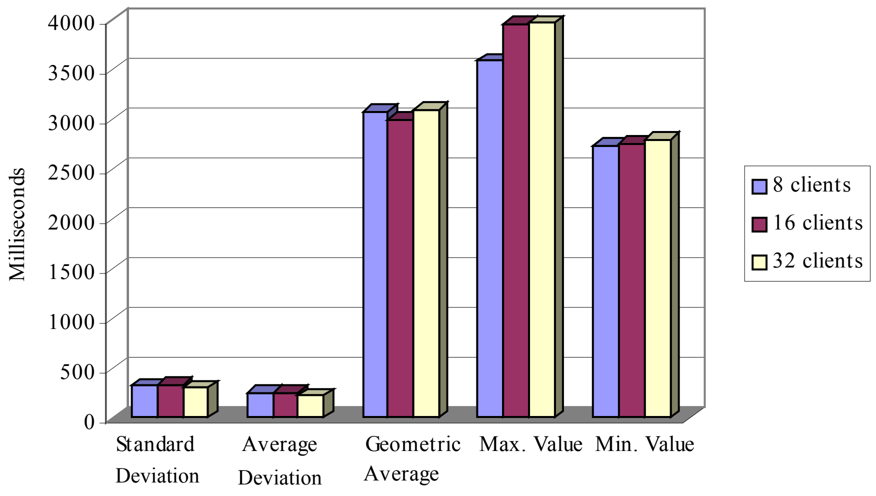


Figure 4.2. Round Trip Time Statistics for 1,3Mbytes data size

In conclusion, the experiments have shown that retrieving 32 KB or 64 KB from the disk was giving the best response time. Therefore, a packet size of 32KB has been recommended for video retrieving from the storage system. However, the video data delivery must be regulated in order not to overflow the clients' buffers and over-saturate the bandwidth.

In general, these experiments have driven us to concentrate on other parts of the systems such as, increasing the number of users served at the same time, regulating the delivery of the transmission, creating graphical user interface for communicating with the server and visualizing the video, and guaranteeing a dual service which does not accept new arrivals who might affect the quality of service.

The results and the progress in the implementation have motivated us to create a tool for visualizing the video data. Therefore, a video client has been created in order to synchronize with the server and to watch the movie as well as to make an evaluation to the server. This part has been tested successfully and has been presented as an undergraduate final project at the Computer Science Department of the University Autonomy of Barcelona [AppendixB][VALL02].

The client player was able to connect and communicate as well as to negotiate with the server successfully. The server has been placed at the department of computer science at UAB. The clients have been tested from the labs of UAB and Sabadell which is located at about 10 Km north of the UAB campus.

4.2.3 Stage Two: RMM and Admission Control Policies

The first stage of the implementation has shown the need for guaranteeing the service for multiple clients so that new arrivals don't affect the ones being in service. That is, the quality of service (QoS) of the clients being in service is not jeopardized. Therefore, sufficient resources must be assigned to these clients.

As it has been stated in chapter two, the traditional operating systems don't guarantee the QoS for the long-live sessions in multimedia applications. Also, the operating systems treat all requests equally unless priorities are defined. This implicates that a new request might acquire the resources of others being in service causing an interrupt in the reproduction of the video at the client side. Therefore, there was a need for creating a middleware with a set of policies that take control over the operating system resources and control their assignment to the different clients. The middleware includes the admission control policies, discussed in chapter two, in order to decide whether to accept or reject a request based on the available resources.

No doubt, this extension in the system design required an extension in the architecture design and implementation, and required the need for investigating the main elements that play a role, directly or indirectly, in the designing process of the architecture such as the CPU, the Memory, the Network, and the Disk.

The implementation of the resources assignment and the admission policies have been done by creating another multithreaded module called Resources Manager Module (RMM). The RMM module consists of the following parts:

- The main program
- The CPU thread
- The memory thread
- The network thread
- The disk thread
- The reader and transmission thread

When the admission control module (ACM) forwards a request to the main program of the RMM, the RMM analyzes the request and translates the requests' parameters into resources based on the metadata related to that request. Thereafter, the RMM awakes up the threads of the CPU, memory, network, and the disk and passes to them the metadata and the policy that they must considered in calculating the amount of the required resources.

The resources threads then make their tests and reservations based on the policy defined by the RMM and inform the result to the RMM. If all the resources confirm the admission then, the request is registered as accepted in the acceptance list and a reader thread is created for retrieving the video data and transmitting it to the client.

For the experiments, the task of the video storage (the disk) has been separated from the rest of the implementation. The reason behind that was due to the fact that typical video files require a huge amount of disk space thus; storing and retrieving these files require especial and careful treatment. For example, the files can be stored in their entirety in sequential order. Another implementation would be stripping the video files among a set of disks. Therefore, this issue has been done by a separated study. The task assigned to that study has provided a very short delay (less than 3 ms) for retrieving the video by the thread which is in charge of delivering the data to the clients.

With the objective of analyzing the RMM along with the admission control policies and of adjusting the parameters within a real system, a set of experiments has been carried out. These experiments have used MPEG video files and required extracting metadata from theses files. The experiments consist of a set of clients requesting services from the VoD server.

To accomplish these experiments, two mechanisms have been used for generating the petitions: gaussian and random. In the gaussian model the interval between two petitions is defined by a gaussian variable $T=G(\mu,\sigma)$ seconds while in the second model the petitions are generated in a random form in the interval [a, b] seconds. The function parameters have been determined based on real studies presented by [VEN97] with $\mu=2.5$, $\sigma=1$, a=0 and b=6. The parameters for the system are: number of frames/sec requested 29, free CPU time 10%, $\lambda=1$(predictive) and $\lambda=0.5$ (arithmetic average), available memory size 64MB, packet size 32 KB, and LAN with FastEthernet.

Figures 4.3 to 4.8 show the result of these experiments for the two-client models working with frame-based (data is retrieved and sent frame by frame, where a frame can be of type I, P, or B) or block-based (data is retrieved and sent block by block, where a block can be of 32KB or 64 KB) and with the different admission policies, where M=Maximum, A=Average, AM=Adaptive Maximum and AA=Adaptive Average.

Figures 4.3 and 4.4 show that, the Maximum policy uses greater resource quantity (CPU and Bandwidth) than the Average policy (a behavior that could be anticipated) and that the CPU quantity used by the block-based mode is slightly inferior to that of the method frames-based mode. Furthermore, the Adaptive Average policy uses smaller resource quantity (CPU and Bandwidth) than the other policies, with the exception of the Average policy.

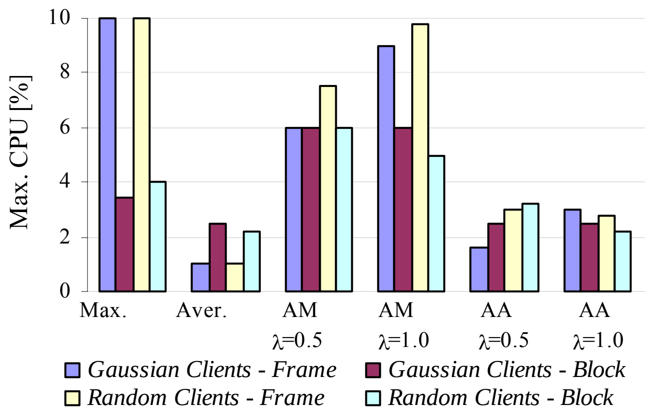


Figure 4.3. Maximum CPU

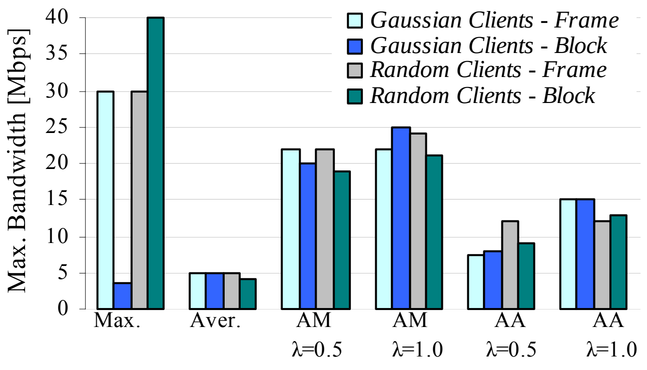


Figure 4.4. Maximum bandwidth

As it can be observed from the figures, the best policy would be the Average policy, but this is the only one that is totally unacceptable due to the great quantity of I frames which are lost. In our case, this policy accomplishes an underestimate of the quantity of necessary resources when the movie contains a high number of I frames that cannot be sent, making this policy unacceptable in terms of QoS.

For the remaining policy, a selected parameter would be the quantity of attended petitions (figure 4.5). In the first place, we need to bear in mind the number of attended petitions (this may be some what inadvisable index, since not all petitions are equal). It can happen that a policy attends to an equal number of petitions of another policy but with a greater consumption of resources. That is, the requests that have been rejected by a policy could be accepted by the other. This can be observed in the throughput of Gaussian clients of block-based with Average policy (Aver.) and Adaptive Average (AA), with λ=0.5 attending to the same number of petitions; however, in the later (AA), the quantity of necessary resources is much greater.

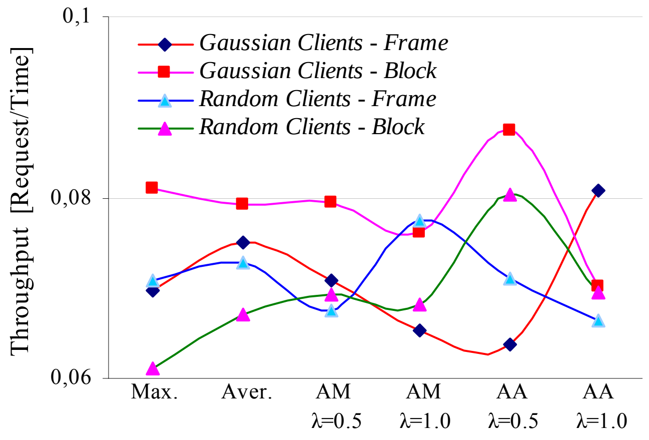


Figure 4.5. Throughput (Number of requests / processing time)

Figures 4.3, 4.4, 4.5, and 4.6, show that the frame-based method gives better results than that of the block-based mode. The justification is that the policies used are adapted better when the processing time is constant and not when the size of the information is constant (block-based working mode).

Because of this, and discarding the average policy (high number of lost frames), frame-based results have been represented in the figures 4.7 and 4.8 as two-axis graph for the CPU-bandwidth utilization and the throughput-waiting time for each policy respectively.

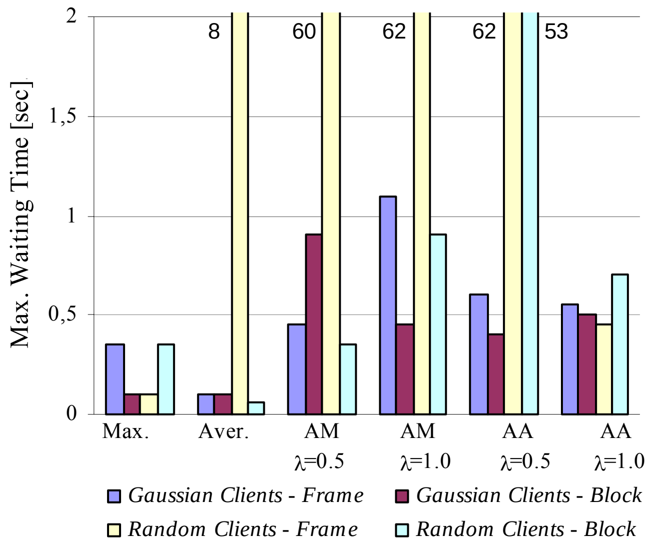


Figure 4.6. Client waiting time and throughput

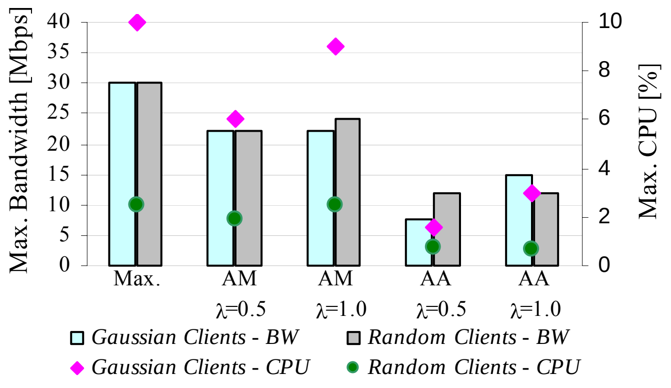


Figure 4.7. Maximum CPU and bandwidth

As it can be observed from figures 4.7 and 4.8, the adaptive policies are better in general, and the Adaptive Average is better than the Adaptive Maximum since it reduces considerably the waiting time. In terms of resources, the Adaptive Average policy is also better since it has a high throughput with low need of resources. Furthermore, these policies have a better behavior when the value of λ is close to 1 and the client's model is Gaussian.

As a conclusion to these experiments, the RMM, which is implemented as a middleware using four brokers, can reserve the necessary resources based on a predefined admission policy and provide a quality of service represented in providing the video data without starvation.

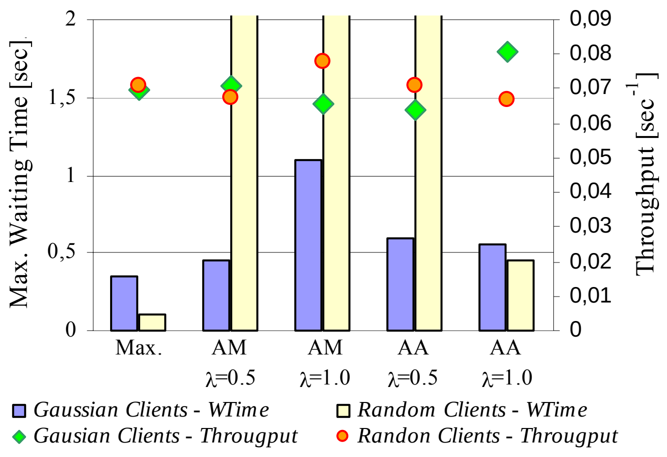


Figure 4.8. Frame-based: waiting time and throughput

However, the resource manger does not make use of the resources availability in order to assign the double or may be the triple of the resources to the clients who have high machine specification so that they get work-ahead video data and leave the server earlier. It can only calculate and reserve the necessary resources for each arriving request.

4.2.4 Stage Three: CB_MDA and Data Pre-fetching

As it has been said before, the brokers can only guarantee the required resources according to a predefined policy. However, regulating the video streaming and controlling the flow of the data between the server and the clients are carried out by the CB_MDA. Thus, the CB_MDA also allows the server to avoid using protocols that might cause network overhead such as RTP.

The main experiments of this stage are intended to test the CB_MDA. These experiments are related to the following issues:

- Combining multicast and unicast channels for video streaming
- Implementing patching technique for the clients who join one of the active multicast streams so that they can buffer from these streams
- Providing the video prefix through short unicast channels so that the client can start right away the reproduction of the video.
- Implementing video pre-fetching with the unicast channels so that they can be released earlier.

These techniques are expected to lead to a better utilization of the resources. As we have mentioned in chapter three, streaming the video data in real time does not give high performance since there are moments when the server has available resources that can be assigned to some clients, who are capable of receiving video data at a rate greater than the real time rate, but the server does not utilize these moments. The alternative solution would be transmitting and controlling the flow of the video data at a rate greater than the real-time rate whenever it is possible so that, the server can finishes serving these clients some time earlier than the time that would be needed if the server were sending at a real time rate only. As a result, and before the peak time, these clients will be finished and new clients will be able to join the service.

Consequently the CB_MDA has presented a solution with which, it discovers the periods of time during which the server resources are underutilized and, at the same time, there are clients who have high machine specifications such as the IO bandwidth and buffer space so that, the CB_MDA assigns to those clients more resources than others creating a work-ahead in their buffers.

In addition, and at the application level, the algorithm guarantees, if there is no network failure, the flow of the data from the server to the client without causing starvation at the client side.

The first set of experiments has been conducted with the transmission being sent over unicast channels only. That is, each client has been assigned a unicast channel. Meanwhile, the algorithm, whenever it was possible, was trying to send the video data at a rate greater than the real time rate to those who can receive it. In general, the experiments verify that, the algorithm can take advantage of the available resources at the server and the client sides and manages to send work-ahead video data. Also, they show how the algorithm adapts dynamically, during the service, to the changes that take place in the server and to the arrival of heterogeneous clients. For example, the change can be due to the arrival of more clients and/or the departure of clients being in service. The incorporation of the new arrivals (clients) is done dynamically without affecting other clients being in service. Also, some clients might have better machine specifications so that more resources can be assigned to them during the service. So, the adaptation to these changes is done on the fly while the server is running.

Experiments set 1:

The objective of the first set of experiments is of two folds: first, testing the ability of the system to handle so many clients that might arrive at the same time (with an interval < 1 sec), second, testing the ability of the algorithm to create a pre-fetched (work-ahead) video data reserve in the clients' buffers so that, the clients can finish before the time that would be needed if the server was sending at the video reproduction rate. The main parameters and characteristics of these experiments are as follows:

- Maximum credit values that can be assigned (Maxcv) = 40 cvs which means that the server can generates 40 streams at the same time, (Recalling from chapter three, a cv represents the amount of the video data that the server must transmit so that the client can play one second of the video).
- Maximum number of clients that have arrived (Max clients) = 40,
- Short video files (e.g. Ontario.mpg with 314 sec). This file is an MPEG format and has this size of about 50MB (Mega Byte).
- The clients I/O bandwidth (Clients' IO) = 2 cvs, which means that he can receives from multiple streams.
- The maximum video data that the client can buffer (Max Buffer) is equivalent to 105 seconds of video reproduction. If the server manages to send 2 cvs during a slot then, 1 cv will be consumed for the video reproduction and the other cv will be buffered at the client side as a pre-fetched reserve (PRV).
- The average arrival time of the clients has been applied as < 1 second and then 5 seconds. That is, the average delay between the arrivals of the requests is less than 1 second (in the first trial) and equal to five seconds in subsequent trials.

The following table presents a summary of the experiment set1:

Experiments Set 1							
Objective	The objective is to the ability of the systems to handle so many clients at the same time, and to test whether the algorithm can create prefetched reserve when the number of the client is equal to the number of the credit values CV.						
Parameters	No. of Clients	MaxCv	IO BW	Buffer size (sec)	Arrival Delay	File Size	Play Time
	40 40	40 40	2 cv 2 cv	105 105	< 1 sec 5 sec	50MB 50MB	314 sec. 314 sec.

In the first experiment, when the arrival delay was less than one second, the server (mainly the ACM) was able to receive and handle all of the clients who have connected to the server. However, the average service time was two seconds for each client. That is, although the clients were arriving every less than one second, the server (in general) needed about 2 seconds in order to verify the client request, create the necessary thread, and start transmitting the video data. As an example, figure 4.9 of set 1 (the word "set" indicates set of experiments) shows clearly that client i+10 and client i+40 have received the first fragment of their video file at about 20 and 80 seconds respectively.

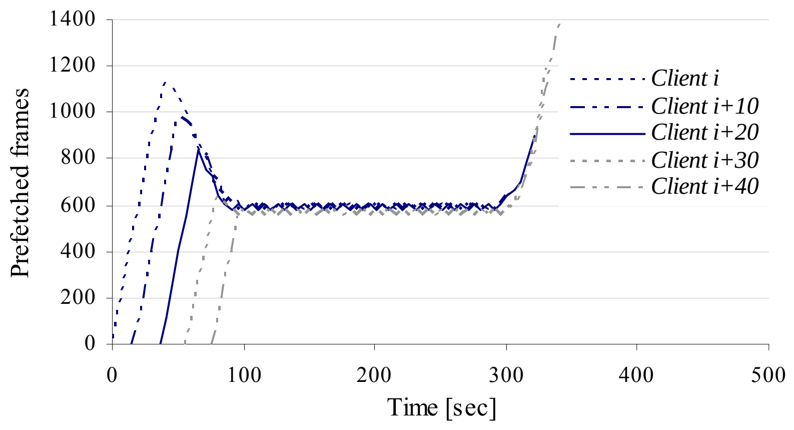


Figure 4.9. Serving 40 clients with arrival delay < 1 sec.

Also, this first experiment (figure 4.9) shows that, even though the server needed around 2 seconds before it can serve each request the algorithm has utilized this little delay and was able to create pre-fetched reserve in the clients' buffers. Thus, the arrival delay can be utilized to create pre-fetched reserve in order to reduce the service time. Therefore, the time delay has been increased to five seconds in subsequent experiments for three reasons; first, as it has been said before, the server needs 2 seconds to handle each new request, second, it is preferable that the user waits 2 to 3 second before he can start the video reproduction so that some video data can be accumulated in his buffer, third, the five seconds value was one of those values generated by Gaussian and random models and used in the RMM experiments. The most important thing is that, guaranteeing that, within this value (5 sec), the last arrival client is entering the server while the first client is still in service.

Figure 4.10 shows the status of the clients' buffers after the algorithm has finished servicing all of them. The graph shows that all the clients have finished with PRV in their buffers. Also, it shows that the clients who have arrived earlier had less pre-fetched reserve values (PRV) than those who came later and, it shows that some clients (the latest arrivals) had their buffers full.

This is true due to the fact that, when the number of the clients was less than Maxcv, the algorithm created PRV in the buffers of earlier clients. Then, when all the clients were present in the system, the earlier clients have been scheduled less frequently in some subsequent slots since they had larger PRV in their buffers than those who came later. Thereafter, those who came first began to exit the system, since they got all the requested video data, and their resources have been assigned to the rest of the clients. As a result, there were moments when the later clients were receiving at their high capacity and their PRV was frequently increasing. For example, figure 4.10 shows that clients 31 to 40 got their buffers full with video data.

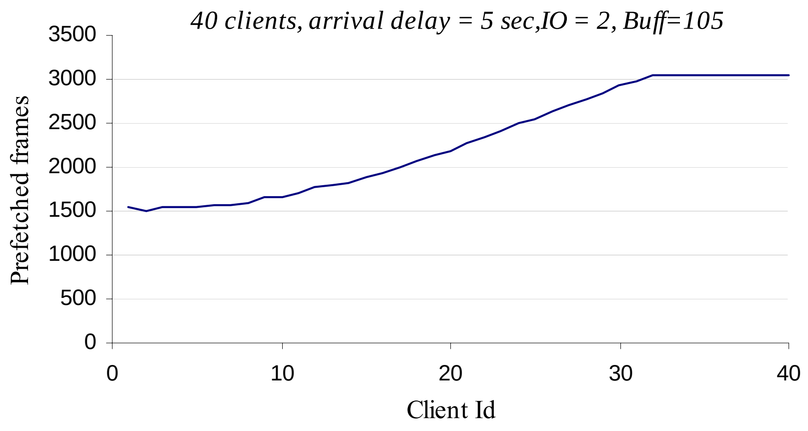


Figure 4.10. The buffers status of 40 clients after being finished

To have a more detailed picture about the buffers status during the service, figures 4.11, 4.12, and 4.13 show how the buffers were evolving during the service for a sample of 3 clients who have the id *i+10*, *i+30*, and *i+40*. The buffer status of these clients has been registered every five seconds since the files sizes were small.

These figures show clearly how the buffers of the earlier clients were frequently increasing with the video data and then, when more clients were arriving, these buffers started to decrease since the clients had to share their resources with the new arrivals. That is, the earlier clients have been ignored in subsequent scheduling slots causing them to consume more than what they used to receive.

For example, figure 4.11 shows that client *i+10* has his buffer increasing frequently during the service. Thereafter, and with the arrival of more clients, his buffer started decreasing, which indicates, that he has been ignored in some subsequent scheduling slots.

Figure 4.12 shows that client *i+30* has been also ignored in subsequent slot but less times than those who arrived before him. Also, it shows that he has utilized the exit of the earlier clients so that his buffer went back to increase again.

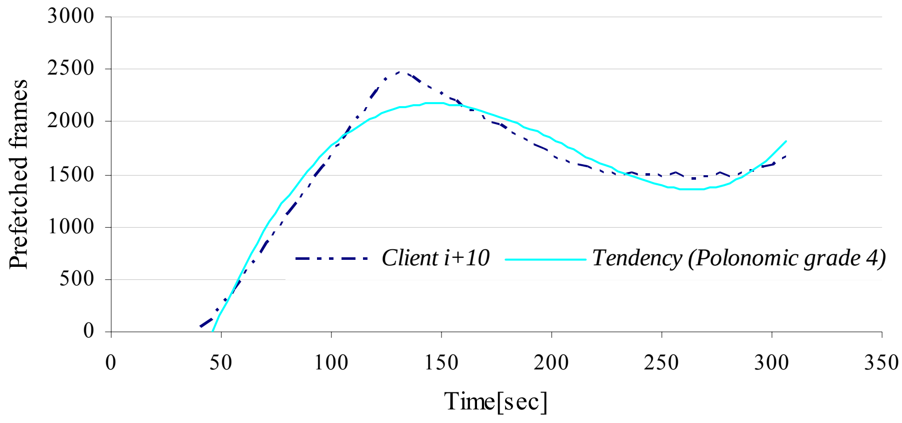


Figure 4.11. The buffer status of client *i+10*

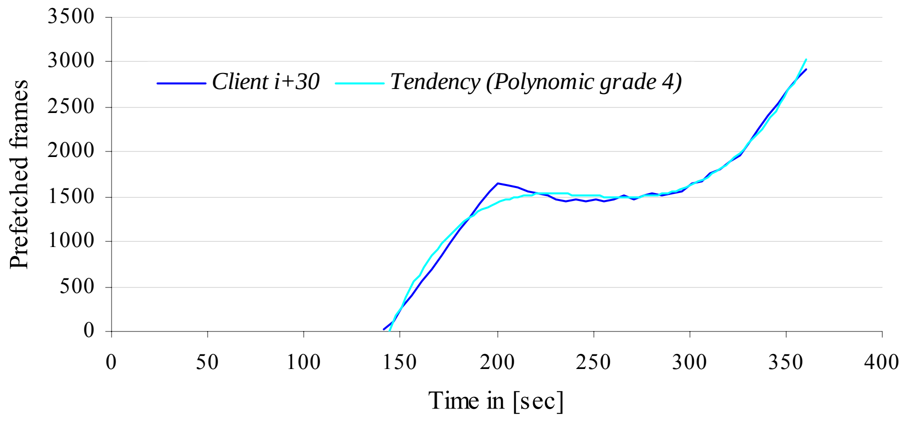


Figure 4.12. The buffer status of client *i+30*

With regard to client *i+40*, figure 4.13 1.5 shows that he has been frequently scheduled in subsequent slots. This is due to the fact that, when the clients have reached a balanced level in their buffers, the earlier ones started to exit the system giving up their resources which have been shared among those who were still in service.

Figure 4.14 combines a set of clients (*i*, *i+10*, *i+20*, *i+30*, *i+40*) in one graph so that, it can be seen clearly how the buffers of the different clients are changed during subsequent slots and how the algorithm brought the buffers into a balanced level.

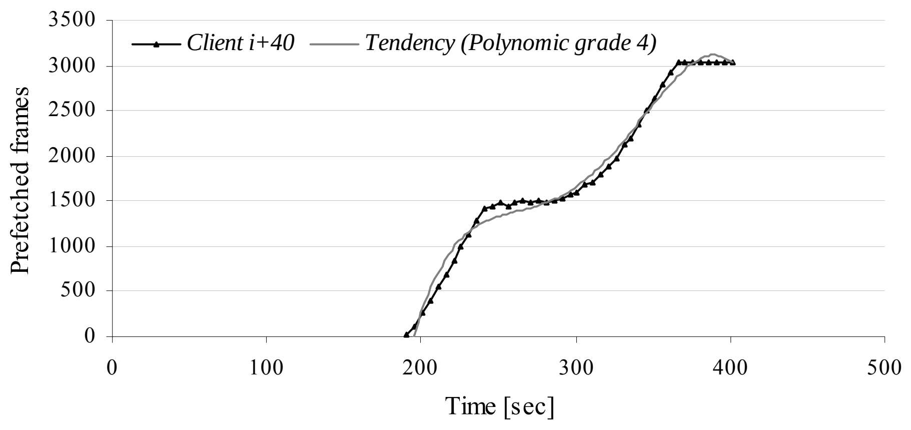


Figure 4.13. The buffer status of client *i+40*

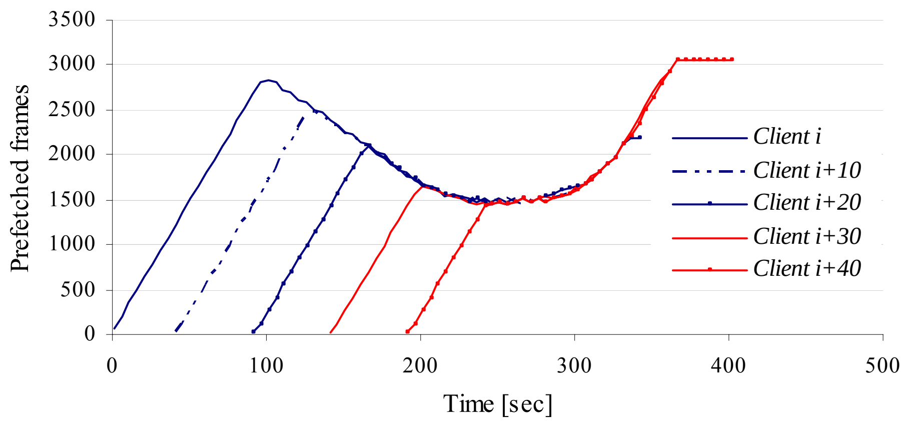


Figure 4.14. The buffers status of the clients' sample

As a conclusion, the first set of experiments, which are represented in the figures 4.9 to 4.14 have shown that, the algorithm was able to create pre-fetched video data reserve in the appropriate clients buffers and that, it was able to maintain a balanced pre-fetched reserve in their buffers.

Consequently, all the clients of the first set were able to finish ahead of the time. That is, the server has finished serving them before the time that would be needed if the server has been sending the video data at the consumption rate of the video reproduction all the time. This is will be beneficial in implementing multicast along with the short unicast channels which will be used for transmitting the video prefix at a rate greater than the real time rate.

Another point that can be deduced from the first set of experiments is that, figures 4.10 to 4.14 have shown that the clients had pre-fetched reserve in their buffers. This reserve represents the resources that have been allocated ahead of the time. The idea now is to think of these resources as borrowed slots that can be taken back from the clients. This can be implemented by ignoring these clients in subsequent slots, as if the borrowed resources are reclaimed, in favor of new arrivals.

Experiments set 2:

In the real life, when some body gives a loan to another, the former is expected to get it back from the later whenever the former asks for it. The algorithm is imitating this situation by assigning more resources, whenever it is possible, to the clients with the intention of getting these resources back whenever they are needed. Getting the resources back means ignoring those clients who have large pre-fetched reserve in their buffers in subsequent scheduling slots without causing starvation in their buffers.

For this reason, another set of experiments have been conducted with the objective of injecting more clients in the service without causing starvation in the buffers of the clients being in service. That is, the total number of the clients that can be served at the same time will be greater than Maxcv, yet the quality of the service is not jeopardized. The parameters of these experiments have been maintained the same as the previous experiments except the number of the clients which has been incremented to 50. The following table presents a summary of the experiment set 2:

Experiments Set 2							
Objective	Check whether the algorithm can cope with more clients by reclaiming the borrowed resources without causing frames loss (starvation).						
Parameters	No. of Clients	MaxCv	IO BW	Buffer size (sec)	Arrival delay	File size	Play time
	50 57 59	40	2 cv	105	5 sec	50MB	314 sec

The experiments have shown that the algorithm has managed to serve all the 50 clients without starvation. Therefore, the number has been increase to 57 and then to 59 clients. Figure 4.15 shows the amount of the pre-fetched reserve in the buffer of each client (defined by a unique id) when there were 50, 57 and 59 clients. As the figure indicates, the curves of 50 and 57 clients show that all the clients had pre-fetched reserve when the last fragment of the requested video file has been received. On the other hand, the curve of the 59 clients indicates that some clients were about to suffer from starvation.

In all cases, the same explanation made on the buffer status in the previous experiments can be implemented on theses experiments. Many clients have finished with pre-fetched reserve in their buffers. This is true since the first clients have collected high reserve before the server has become saturated, and thereafter they have finished before sharing all of the reserve with the rest of the clients. The later clients have utilized these resources to increase their pre-fetched reserve, and the last few of them were even about to have their buffers full before they have finished.

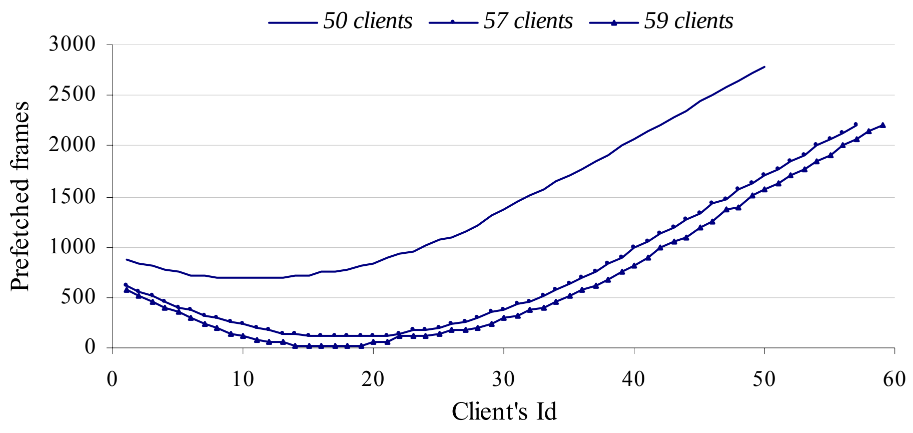


Figure 4.15. The PRV of 50, 57, and 59 clients

To show how the buffers were evolving during the service, figure 4.16, shows a sample of 6 clients chosen from the experiments of 50 clients. The graph shows clearly how the buffers were increasing at the beginning of the service since the number of the clients was less than MaxCV and then, they have been decreasing since more clients have arrived so that, they have been ignored in subsequent slots for the sake of the new arrivals.

In the same way, a sample of 6 clients from the experiments of 57 clients have been combined in figure 4.17 in order to compare it with figure 4.16 and to show how the pre-fetched reserve has been decreasing yet, none of the clients has had his buffer empty. However, when the number of the clients has been increased to 59, some of the clients' buffers have registered zero level in subsequent slots (figures 4.18, 4.19, and 4.20).

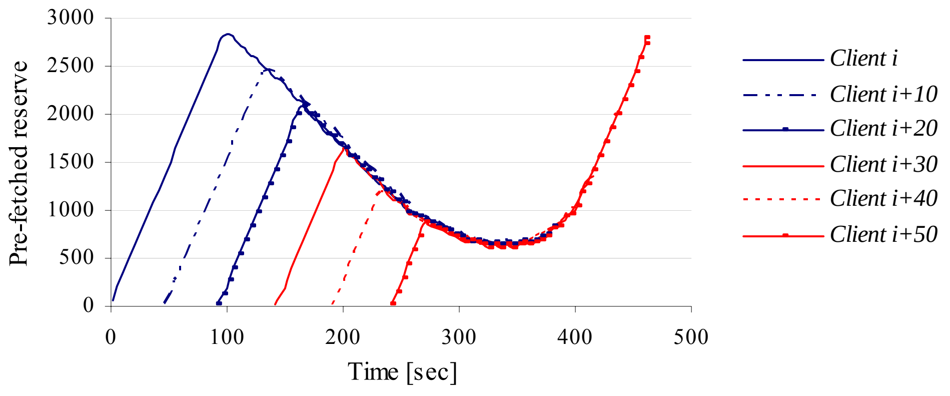


Figure 4.16. The PRV of the 50-client group

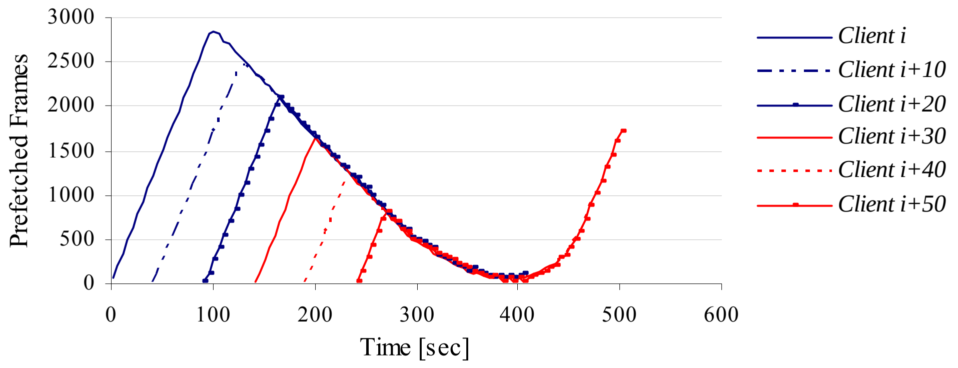


Figure 4.17 The buffers status of the clients' sample 57

The next figures 4.18, 4.19, and 4.20 are present to make a comparison between the pre-fetched reserve of client *i+30* from 50 clients, *i+30* from 57 clients, and *i+30* from 59 clients respectively.

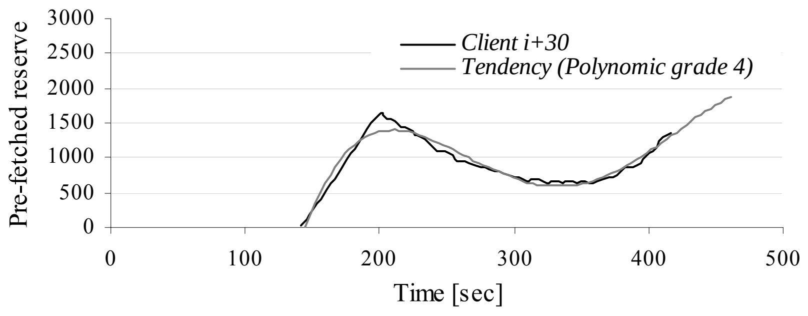


Figure 4.18. The buffer status of client *i+30* from 50-client group

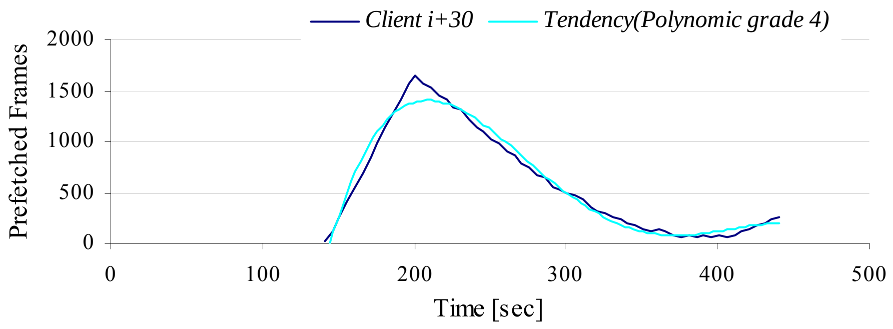


Figure 4.19. The buffer status of client *i+30* from 57-client group

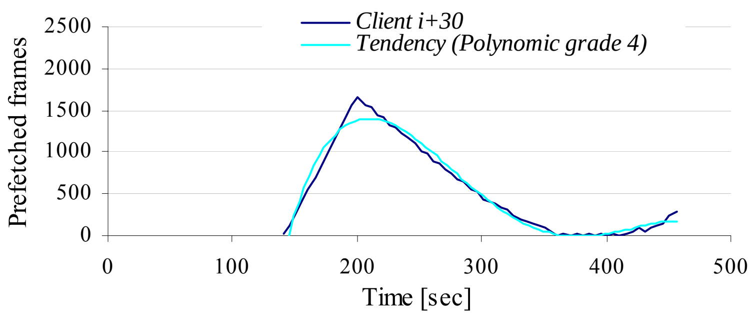


Figure 4.20. The buffer status of client *i+30* from 59-client group

These graphs show the status of the buffer of client *i+30* which represents one of those who arrived in the middle of the service where the server was becoming saturated. Figure 4.20 shows how the reserve in the buffer has almost reached zero level in comparison with the graphs of figures 4.18 and 4.19. The reason for that, of course, is referred to the increase in the number of the clients attended to by the server at the same time.

As a conclusion, any further increase in the number of the clients in the previous experiments will definitely jeopardize the clients' quality of service. However, these experiments have proved that the algorithm was able to recollect the resources which have been assigned to some clients when the server had light load in order to meet the increasing number of the clients. In other words, there were moments when the number of the active streams has become larger than the Maxcv that the server can offer.

Another important point that can be noticed is that, none of the first clients had his buffer full although the server was able to provide more video data and increase the pre-fetched reserve in their buffers. This can be explained as a lack of the IO bandwidth. That is, had the earlier clients had an IO larger than 2, the server would have sent more data to these clients. Therefore, the focus has become on the IO in order to explore how the increment of the IO might affect the number of the clients who can be served by the server.

Experiments set 3:

The objective of this experiment is to test whether the increase of the IO value can prohibit frames loss which has occurred with the 59 clients group (figure 4.20) and allow incorporating more clients in the service. Therefore, the IO has been increased to 3 CVS in order to send the video data at a faster rate whenever it is possible. Then, the experiments have been conducted with 59 clients and then with 62 clients. The following table presents a summary of experiments set 3:

Experiments Set 3							
Objective	Checking the possibility of avoiding frames loss, which was about to occur in the previous experiment, and increasing the clients if the IO value of the clients is increased.						
Parameters	No. of Clients	MaxCv	IO BW	Buffer size (sec)	Arrival delay	File size	Play time
	59 62	40	3 cv	105	5 sec	50MB	314 sec

To make a comparison, figure 4.21 shows the 59-client curve of figure 4.15 when the IO was 2, in addition, it shows the curves of 59 clients and 62 clients when IO became 3.

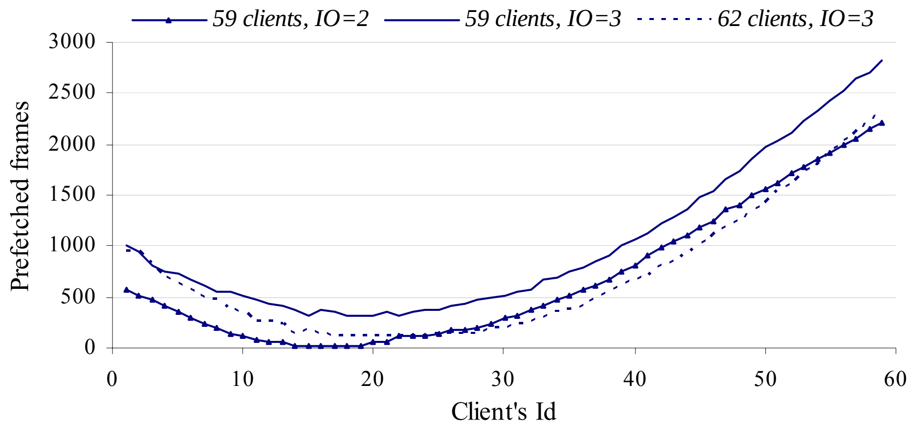


Figure 4.21. The PRV of clients groups with IO=3 CVs

The 59-client curve with IO=3 shows that, with the increase of the IO, none of the clients' buffers has reached zero level. On the other hand, the 62-client curve also shows that, the increase in the IO value resulted in an increase in the number of the clients served by the server yet, none of them has suffered from the lack of data and many of them finished with a pre-fetched reserve.

To have a more clear picture about how the pre-fetched reserve was changing during the service, figure 4.22 and 4.23 show the buffers status during the service for the client *i+30* from 59-client group and 62-client group when IO=3. Notice that, the status of client i+30 will be shown in all the experiments as a base for comparison.

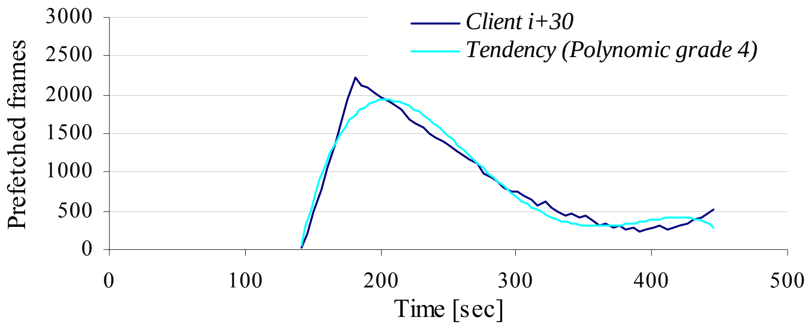


Figure 4.22. The buffer of client i+30 from 59-client group with IO=3

The graph of figure 4.22 shows clearly that, when IO became 3, client *i+30* of group 59 clients did not suffer any more from the lack of video data and that, he has finished with a pre-fetched reserve causing him to finish receiving the last fragment of the data ahead of the time.

Also when IO is equal to 3, figure 4.23 shows that, this client (*i+30*) was about to have his buffer empty and suffer from the starvation. However, the experiment did not register any suffer at any of the clients of this group. To verify this, figure 4.24 shows a global picture for a sample of 7 clients from the 62-client group.

The graph of figure 4.24 shows that the pre-fetched reserve was increasing in the buffer then it has fallen down as a result of receiving more clients. During the period 375 and 450, the clients were about to suffer from the jitter but this did not happen.

As a conclusion, the increase of the IO bandwidth has produced better results especially for the earlier clients. This means that, if the transmission of the video can be at a faster rate then some clients will get more reserve before the server gets saturated.

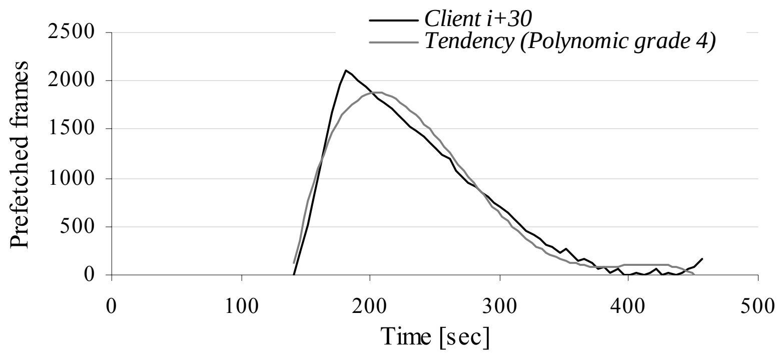


Figure 4.23. The buffer of client i+30 from 62-client group with IO=3

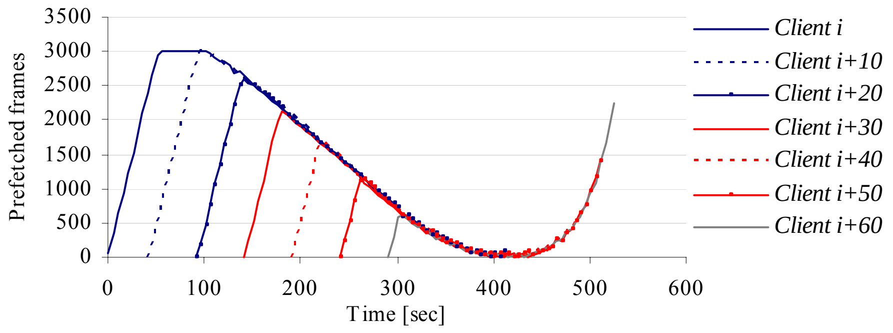


Figure 4.24. The buffers status of a sample of 7 clients

An interesting point figure 4.24 shows is that, the buffers of the earlier clients (*i* to *i+10*) have reached full level since the curves have reached a stable point. This means that their buffers have become full before the system has become saturated. Now, the question is:

What would happen if the buffers were larger? In other words, had these clients had bigger buffers, would they have more pre-fetched reserves which will lead to better results in terms of prohibiting frames loss?

Experiments set 4:

The objective of these experiments is to test whether the increase of the buffer improves the server performance. The same parameters of experiments set 3 were used except the buffer size which has been increased to 157. The following table presents a summary of experiments set 4:

Experiments Set 4							
Objective	Checking if the increase in the buffer size can prevent frames loss when the clients number reaches 62.						
Parameters	No. of Clients	MaxCv	IO BW	Buffer size (sec)	Arrival delay	File size	Play time
	62	40	3 cv	157	5 sec	50MB	314 sec

Figure 4.25 shows a comparison between the pre-fetched reserves when the buffer size was 105 (dotted line) and when the buffer size has become 157 (Continued line). It is clear from the graph that the earlier clients have utilized the server resources before it has become saturated. Figure 4.26 shows that client *i+30* from 62-client group has survived the lack of the data after increasing the buffer size.

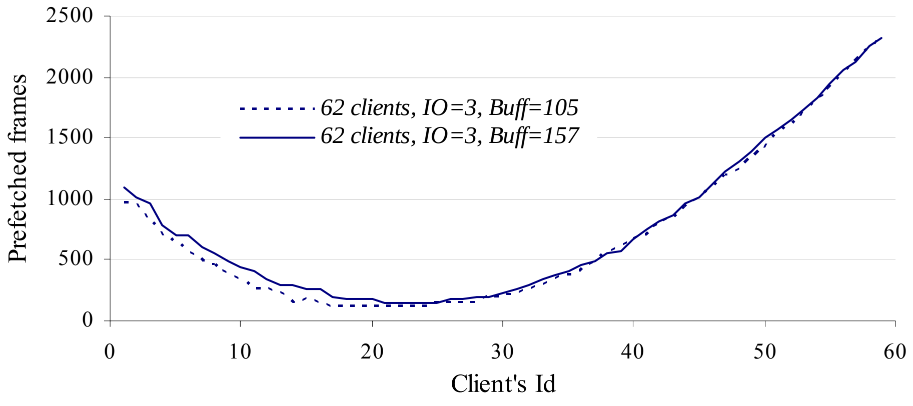


Figure 4.25. The PRV of 62 clients after increasing the buffer

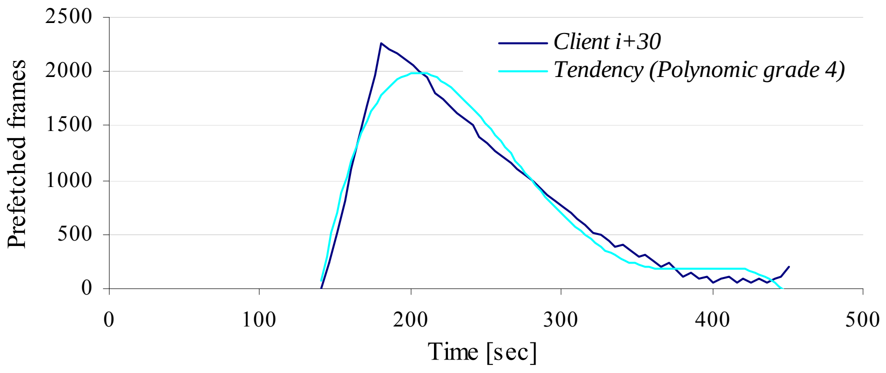


Figure 4.26. The buffers status of 62 clients after being finished

Figure 4.27 shows the pre-fetched reserve for a sample of 7 clients after the buffer has become 157. In comparison with figure 4.24, this graph (4.27) shows that clients *i* to *i+10* were no longer able to have their buffers full. This means that any further increase in the buffer space will be useless unless it is accompanied with an increase in the IO value.

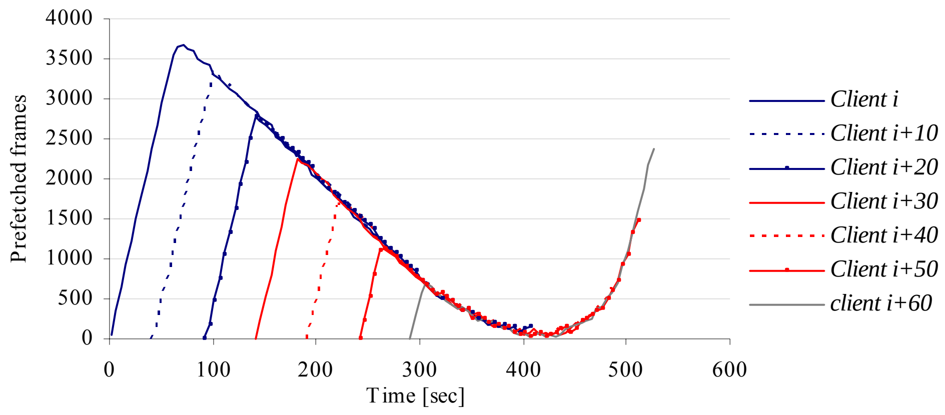


Figure 4.27. The buffers status of clients' sample from 62-client group

As a conclusion, the increase in the buffer size was beneficial in case of the earlier clients. As a result, it is not necessary to have all the clients had large buffer size. For example the data in the buffers of the clients, *i+30* to *i+50*, did not even reach 60% of the buffer size. Thus, a bigger buffer size will be useful only and only when the server has available resources and the client can receive at a high rate.

As a quick summary to all sets of the previous experiments, the algorithm has shown its capacity to adapt itself to the changes of the parameters dynamically. Also, it has shown that, in some moments, an increase in the IO bandwidth and the buffer size resulted in an increase in the number of the clients served by the system.

In addition, there were moments when the number of the clients was larger than the server resources due to the fact that, the algorithm was sending work-ahead video data to the appropriate buffers so that it can ignore those clients who have PRV for the sake of coping with the later arrivals. Furthermore, and consequently, the algorithm managed to reduce the service (finishing) time as it has been explained before giving the chance for more clients to enter the system.

After finishing these sets of experiments, we have calculated the percentage that the value of the average arrival represents with respect to the play time of the video (file length). This value is calculated by dividing the average arrival time by the video length as follows:

Ontario file:
TheAverageArrivalPercent = The average arrival time/FileLength Example: FileLength = 314 seconds The average arrival time = 5 TheAverageArrivalPercent = 5/314 ≈ 0.0159 = 1.59%

Thereafter, other sets of experiments have been conducted with a video file which was 4 times larger than that of the previous experiments (e.g. Futurama = 1402 sec.). The average arrival time for the new file has been calculated based on the same percentage (1.59%) with regard to the file length used in the previous experiments as follows:

***Futurama file*:**

TheAverageArrivalTime = (1.59 * FileLength)/100
TheAverageArrivalTime = (1.59 * 1402)/100
TheAverageArrivalTime = 22.2918

Thus, the average arrival time is approximately 22 seconds. This value has been applied for the new file along with the rest of the parameters such as the buffer space and the IO bandwidth.

For testing purposes, and in the same way as in the previous experiments, the buffer of the client has been assumed to be 30% of the length of the file. The value of the client's IO has been also used in the same way (e.g. 2 and 3 CVs).

Experiments set 5:

The objective of these experiments is to check the behavior of the algorithm when the same percentages of the buffer size along with the same value of the IO and the same number of clients as well as the average delay time are applied to video files which are 4 times bigger than the previous small files. The characteristics and the parameters of the next experiments have become as follows:

- Maximum credit values that can be assigned (Maxcv) = 40,
- Video file length = 1402 sec, (220MB)
- The average arrival time of the clients = 22 seconds

The rest of the parameters such as the buffer space, the IO, and the number of the clients were changing with each set of experiments. For the first set of experiments the selected parameters were as follows:

- The maximum video data that the client can buffer (Max Buffer) = 467, which represents 30% of the file length.
- Maximum number of clients that have arrived (Max clients) = 40, 50, 57, and 59.
- The clients I/O bandwidth (Clients' IO) = 2.

The following table presents a summary of experiments set 5:

Experiments Set 5							
Objective	Using the same percentage of the buffer size and the arrival delay used in the previous experiments and apply them to a video file which is four times bigger in order to se if the algorithm can maintain the same behaviour.						
Parameters	No. of Clients	MaxCv	IO BW	Buffer size (sec)	Arrival delay	File size	Play time
	40 50 57 59	40	2 cv	467	22 sec	220MB	1402 sec

These values coincide with the values used in the first set of experiments when small files were used (e.g. Ontario). The only difference is the value of the buffer size which represents 30% of the video file length (1402) and the average arrival time (22 sec.) which represents 1.59% of the file length.

Figure 4.28 shows the status of the buffers in four tries when the number of the clients was 40, 50, 57, and 59. Again, the figure shows that the algorithm was able to create pre-fetched reserve in the same way with the small files and, it shows that the curves of this figure have the same tendency as the correspondent ones in figures 4.10 and 4.15.

A little comparison of this graph with the correspondent graphs of the previous experiments shows clearly the similarity between them. In the correspondent 59-client curve of the smaller files, the curve indicated that the clients were about to suffer from the starvation. Likewise, the 59-client curve indicates that some starvation has occurred. This is can be explained as a result of rounding the delay time to 22 instead of 23 seconds.

The rest of the curves (40-client, 50-client, and 57-client) indicate that all the clients of these curves have finished before their buffers reached zero level, and the algorithm did not register starvation in subsequent slots.

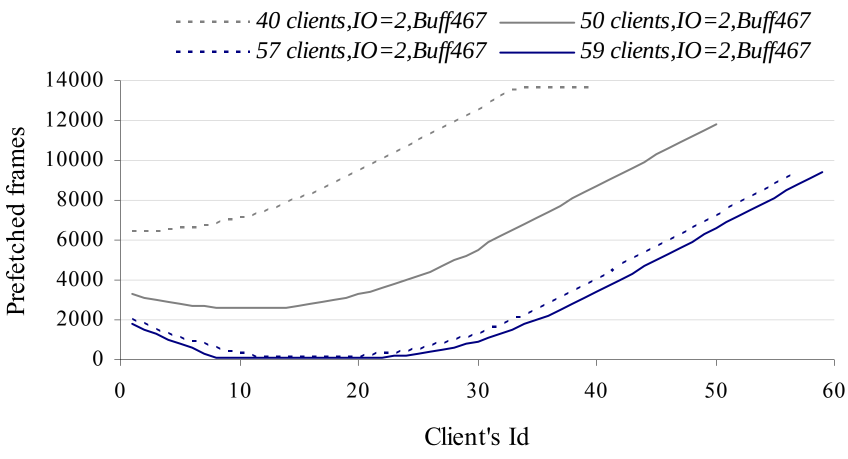


Figure 4.28. The buffers status of the clients' groups 40, 50, 57, and 59

Before commenting on the 59-client curve, it is worthwhile mentioning that the algorithm is able to block the clients and the requests that might cause frame loss to any of the clients being in the service. The frame loss has been permitted in order to study and analyze the parameters which might have some influence on the server performance.

The 59-client curve shows lack of video data in some moments. Figure 4.29 shows how the pre-fetched reserve has evolved in the buffers of the 59-client group, and it shows that increasing the number of the clients to 59 has caused frames loss in some clients' buffers (figure 4.29).

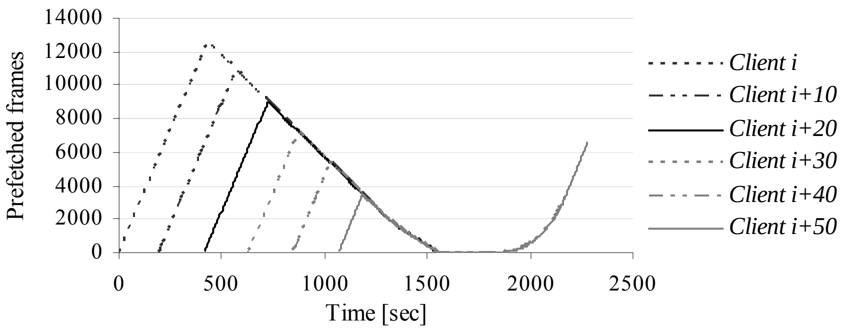


Figure 4.29. The buffers status with bigger files for 59-client group

With 59 clients, the algorithm has registered frames loss in 270 slots from 2376 scheduled slots. These 270 slots were distributed only among the last 50 clients since the first 9 clients did not suffer any loss.

Excluding the 9 clients who have survived the loss, the average loss per client for the rest of the clients is 270/50 = 5.4. This value is equivalent to approximately 157 frames (5.4 * 29frames per second).

The total number of the frames received is 1402 * 29 = 12618 frames.

The percentage loss thus will be 157/12618 = 0.0124.

This problem, the frames loss, has almost occurred in the experiments of the small video files when the number of the clients reached 59 clients and, it has been solved by increasing the IO from 2 to 3. The question is: would the increase of the IO value prevent the frames loss in the same way it did with the small files?

Experiments set 6:

In the next experiments the IO has been increased in order to see whether the frame loss can be eliminated. Thus, the parameters of the next experiment have remained the same except the IO which has become 3. The following table presents a summary of experiments set 6:

Experiments Set 6							
Objective	Testing if an increase in the IO value and buffer space with big files using the same percentage of the small files will eliminate the frames loss appeared in the previous experiments.						
Parameters	No. of Clients	MaxCv	IO BW	Buffer size (sec)	Arrival delay	File size	Play time
	59 62 62	40	3 cv	467 600	22 sec	220MB	1402 sec

With the change in the IO values, figure 4.30 shows that the frames loss does not exist any more and that, the first 10 clients have even got their buffers full during the service before they have fallen down.

Similar to previous experiments, the graph shows that, had these clients had more buffer space they would have maintained more pre-fetched reserve. Therefore, and to prove this assumption, another set of experiments has been conducted after increasing the buffer space to 600 (40% of file length) and the number of the clients to 62, but jitter has been witnessed.

Then, the number has been increased to 63 causing some starvation (figure 4.31). With the increase of both the IO and the buffer space, the first clients have increased their reserve but were not able to get their buffers full. This means, that any increase in the buffer space will be worthless unless it is accompanied by an increase in the IO.

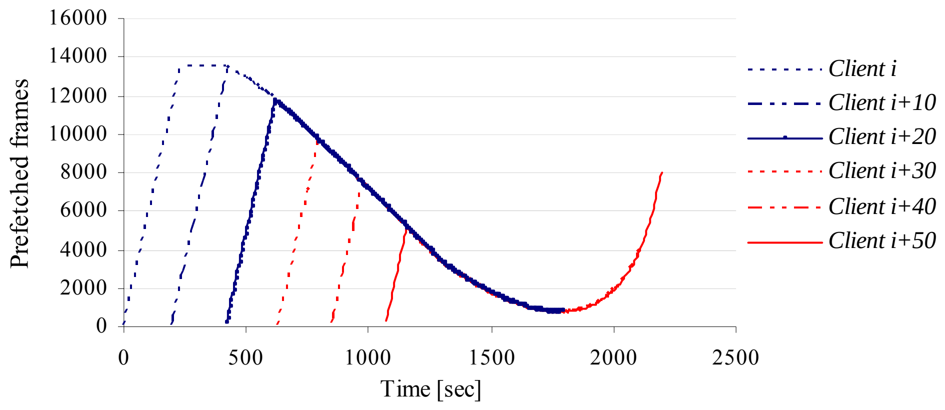


Figure 4.30. No frames loss is witnessed after increasing the IO

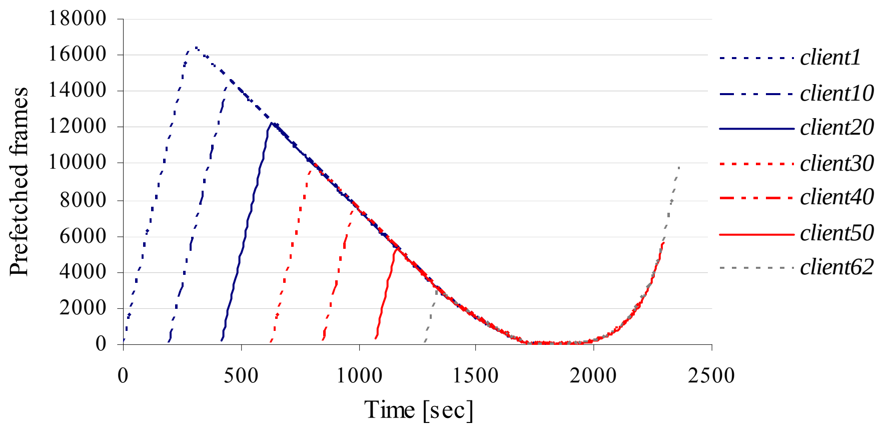


Figure 4.31. Frames loss when the no. of clients is increased to 63

Again, although the algorithm can prohibit the starvation, this experiment has been conducted with starvation occurrence in order to analyze it and make sure that it does not occur in the real implementation.

The first 15 clients did not suffer from the starvation since they have accumulated pre-fetched reserve before the server has become saturated. The rest of the clients, especially those who came later, have had frames loss and suffered from jittering.

To give an idea about the amount of the frames loss, the following steps calculates the percentage of the jitter that has occurred in this set of experiments including all clients:

***Futurama file*:**
Total slot = 2387 Total slots which have witnessed jitter = 176 Total number of clients = 63 The average number of jittering slot per client = 176/63 ≈ 2.8 The average percent of jitter per client = 2.8/fileLength = 2.8/1402 * 100 = 0,2%

Another point that can be deduced from the above experiments is that, it is not necessary that all clients have to have large buffers space and IO bandwidth. It is sufficient that the first clients and the last ones have high machine specifications because, the first clients can utilize the server resources since the server does not yet have so many clients, and the latest clients can also utilize the server resources since the earlier client exit the system rendering their resources to the rest of the clients.

With regard to the CPU and the memory resource, all the experiments have shown that these resources don’t represent a bottleneck and can’t be a major obstacle in providing VoD services. Figure 4.32 and figure 4.33 shows the required memory and CPU respectively based on parameters taken from experiment set 5. The reason for this selection refers to the fact that these parameters (57 clients, IO=2, MaxCV=40...etc.) did not cause starvation, with small and big files, at the client side yet the server reached a point where all the CVS have been assigned.

Figure 4.32 shows the real size of the memory which is required by the VoD server when all the clients are present. Notice that, this size does not include the memory required by the application. The algorithm assigns a memory buffer of 32Kbytes and 2 CVs for each scheduled client. Thereafter, a block of video data equal to 32Kbytes is retrieved from the disk and temporarily stored in the memory buffer from which the delivery thread transmits the video data to the client.

As figure 4.32 shows, the maximum amount of memory needed is around 640KB for transmitting the video data. In fact, this amount is used for serving 20, rather than 57, clients at the same time. The reason for this is that, the algorithm can assign, as a maximum, 40 CVs however, when all the clients are present the algorithm assigns 2 CVs for each of the next 20 scheduled clients instead of assigning 1 CV for each of the next 40 clients in order to reduce the overhead generated from reordering and shuffling the lists of the requests in the server. Had the algorithm has assigned 1 CV for each client the amount of the memory would be doubled since each client will be assigned a 32KByte memory buffer. In fact, this is possible when the clients' buffers get full and thus the clients can't receive more than 1 CV in subsequent slots. Therefore, more clients will be scheduled and more memory buffers will be assigned to these clients. This situation can raise the memory needed up to 1.28 MB.

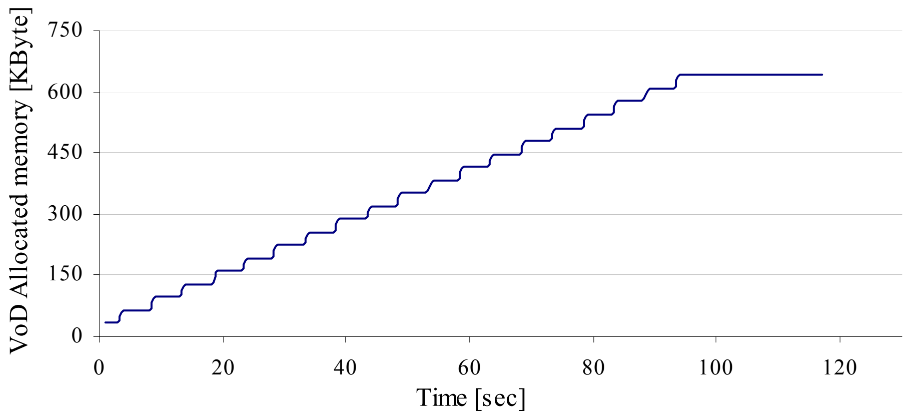


Figure 4.32. The memory needed when all the CVs are assigned

With regard to the CPU, figure 4.33 shows the maximum CPU computing time needed for the process of retrieving 32KB of video data from the disk and transmitting the last fragment of it over the network. Again, the figure does not include the computing time required by the application.

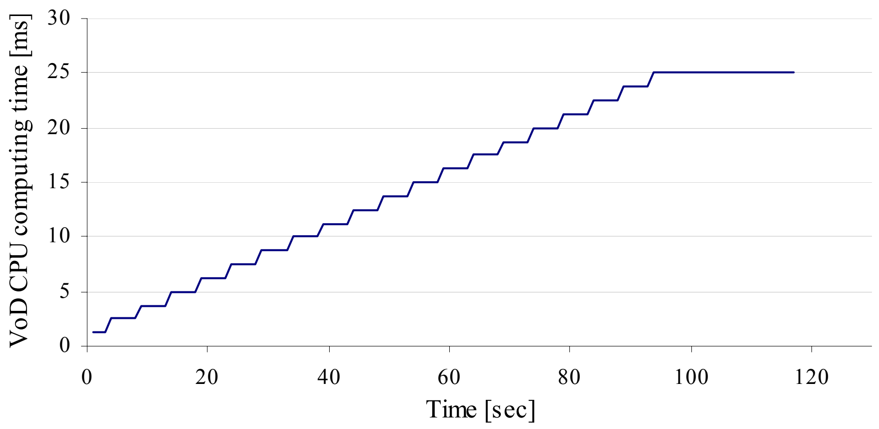


Figure 4.33. The CPU computing time when all the CVs are assigned

Different from the memory, the needed CPU computing time during a slot is calculated as if 40 clients were served instead of 20. The reason is due to the fact that, although 20 clients are scheduled, all the CVs (40) are assigned during this slot. Thus, the total amount of the retrieved and transmitted data is the same in both cases. Thus, the calculation is doubled for each client. In other word, the assigned CPU resource for each client is doubled.

Concerning the required bandwidth, figure 4.34 shows only the bandwidth required by the VoD without calculating the packets headers. It shows the maximum bandwidth needed when all the CVs are assigned.

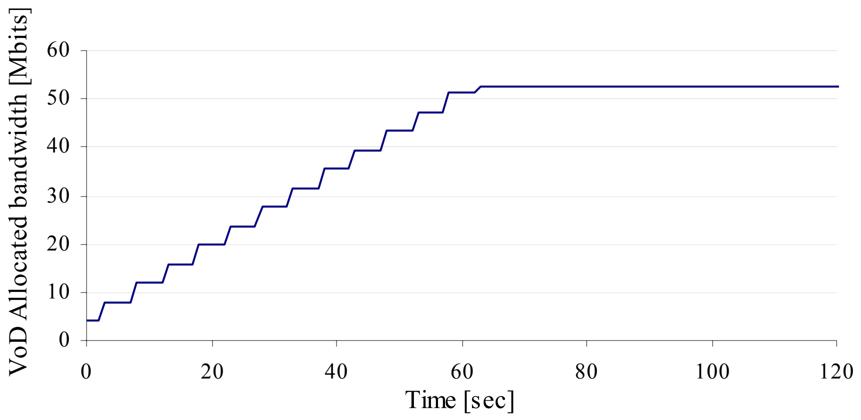


Figure 4.34. The required bandwidth when all the CVs are assigned

The disk storage guarantees a very low respond time when the block of the data is between 32KB and 64KB. Figure 4.35 [YAN03] shows the result of retrieving blocks of video data of different sizes. The delay for retrieving 32KB or 64KB is between 2 to 3 milliseconds. This delay is acceptable by the CB_MDA algorithm and does not affect the scheduling process of the video streams.

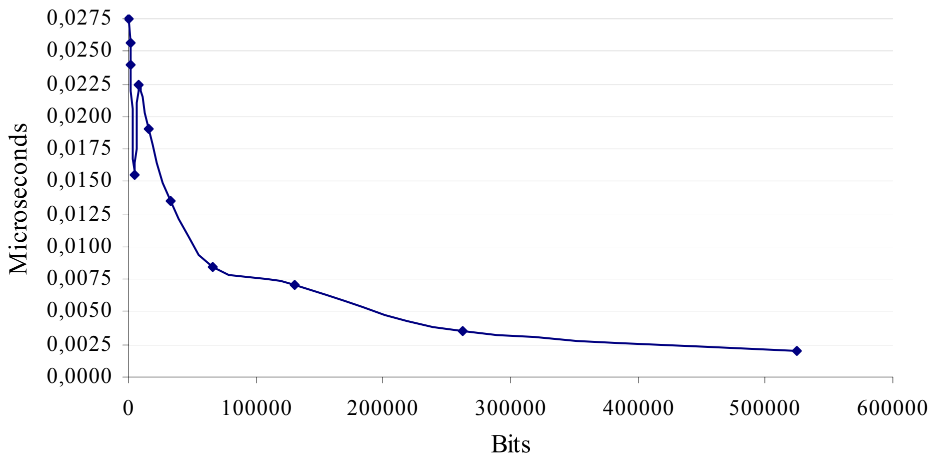


Figure 4.35. The response time for retrieving data block of different sizes

With regard to the quality of service (QoS), we have defined two ways for measuring the QoS. The first one refers to the quality of the video reproduction and the second one refers to the client investment in his machine.

The client gets the best QoS in term of video reproduction when the average CVs assigned to him during the service is >= 1 and, during the service, the client buffer did not register any starvation during any subsequent slot. In other words, if the client gets a CV=1 during all the slots then this indicates that, he has been assigned the necessary disk bandwidth, network bandwidth, memory, and CPU so that, he does not suffer from the starvation and he gets the answer in a short period of time.

On the other side, the QoS can be measured based on the client's machine capacity. That is, if the client can support 2 CV but he gets an average of 1 CV then, he gets half of the quality of service that he was expecting. To see this clearly, the buffers of client *i+30* of experiment set 1 (figure 4.14) and client *i+30* of experiment set 2 (figure 4.19) show that, no starvation has been registered in both cases however, the Quality of Service in the former case is better than that of the later case.

For example, the client of experiments set 1 started the service at time 145 and finished at time 355. Thus, the service time is 220. Knowing that the video length is 314, the client has finished 94 second before the time that would be needed had the server assigned only 1 CV during the whole service. As a result, the QoS from the server point of view, for this client, is the total video sent plus the saved time (314+94) divided by the file length (314). Thus, the result is equal to approximately 1.15. However, from the client point of view, the QoS would be the time of the video received (314) plus the time he has gained (104) divided by what the client's machine can offer (2CVs). Thus, the QoS is (314+104)/618 ≈ (0.58). This means that only 58% of the client resources have been utilized.

On the other hand, and using the same calculations with the same file size (314), client i+30 of experiment set 2 has a lower QoS, yet he did not suffer starvation. He started the service at time 145 and finished at time 445. Thus, the service time lasted 300 seconds. As a result, the client has gained 14 seconds only. Likewise, the server has finished servicing this client 14 seconds ahead of the time however it incorporated more clients in the service. Consequently, the QoS from the server point of view is equal to (314+14)/314 ≈ 1.05 plus the extra clients who have been incorporated in the service while the QoS from the client point of view is equal to (314+14)/628 ≈ 0.55. Figure 4.36) and figure 4.37 show the QoS from the server and the client point of view respectively for three sets of experiments with 40, 57, and 59 clients. Also, the figures show the threshold line of the QoS.

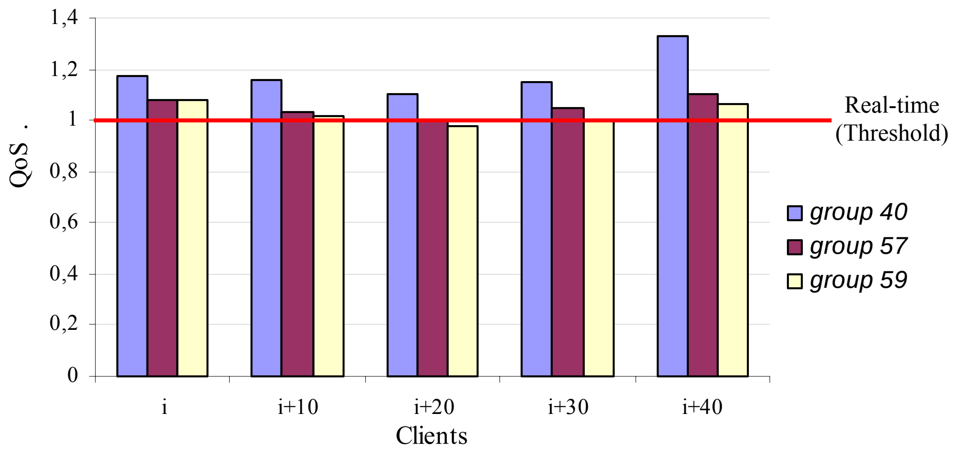


Figure 4.36. The QoS from the server point of view

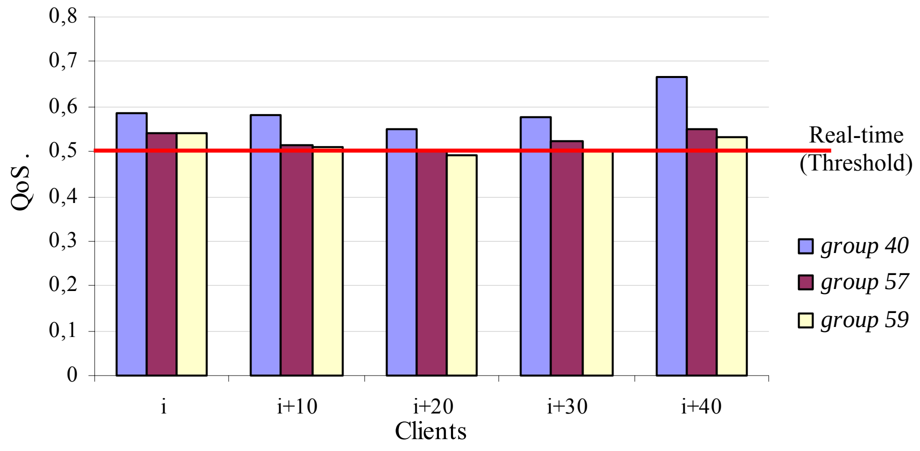


Figure 4.37. The QoS from the client point of view

To satisfy the clients who have higher machines capacities than the rest of the clients, it would be more efficient to the client as well as the server to schedule them, if possible, during the periods when the server has sufficient resources. Mathematically, the server must accept more clients. However, the collision and the headers overhead restrain the server from doing that.

As a conclusion to all above sets of experiments, several assertions can be stated as follows:

- ***Assertion 1***: The algorithm can send work ahead video data to appropriate client's buffers creating pre fetched reserve.
- ***Assertion 2***: The algorithm can adapt dynamically to the changes that take place in the system such as the arrival of new clients and the exit of clients being serviced.
- ***Assertion 3***: The algorithm can adapt to the heterogeneous clients who have different capacities in terms of buffers space and IO bandwidth.
- ***Assertion 4***: Although the starvation has been permitted in the previous experiments, the algorithm is able to detect the clients who might affect the quality of the service of others being in service and block them.
- ***Assertion 5***: As a result of the first assertion, the server can finish some of the clients before the time that would be needed if the service were in real time for all the clients. Thus, the server can finish serving some clients before the peak time when more clients are becoming interested in connecting to the server.
- ***Assertion 6***: Although the algorithm, so far, was using unicast channels, it has shown that it was able to manage and cope with more clients than what the resources would serve. Therefore, it is expected that multicast streaming would increase the number of the clients that can be served and reduce the required resource that would be needed for serving the same number of clients.
- ***Assertion 7***: Sine the increase and decrease of the memory and the CPU resources bind to the way the algorithm schedules the clients, these resources don't represent bottleneck in the VoD service even though the algorithm assigns all the available CVs. Therefore, they will not be discussed in the following experiments.

4.2.5 Stage Four: CB_MDA and Multicast & Unicast Transmission

The next set of experiments refers to the transmission of video data using a combination of multicast and unicast channels. As it has been explained in chapter three, a request for a none-active video stream (file) is served by a multicast channel. A subsequent request to the same stream is checked to see whether it can be allowed to join the on going multicast stream or to initiate a new multicast stream for this request.

If a client (or request) is managed to join a multicast stream then the algorithm allows this client to patch (buffer) the video data form the multicast channel and, at the same time, it assigns a short unicast channel to him in order to transmit the video prefix. Therefore, the algorithm must decide when to allow the client to join an on going stream and when to create a new multicast stream.

Chapter three has referred to studies which have used a patched value between 8 to 15 minutes in order to have a balanced number of multicast and unicast channels. However, in this stage, the experiments will use a patching value of 15% of the file rather than 15 minutes. This value has been deduced from preliminary experiments in order to decide an adequate patching value. Further discussion will be presented in stage 5 where the experiments show that having the patch as a percent of the video file will give a better performance.

Another issue tested in this stage is, the ability of the algorithm to reduce the usage time of the unicast channels in order to allow more users to join the ongoing multicast streams. The experiments which have done in stage three have shown that it was possible to send work-ahead (pre-fetched reserve) data so that the communication channels can be released earlier. Thus, the algorithm tries to send work-ahead over the unicast channels in order to reduce their usage time.

In this stage, two sets of experiments have been conducted in order to test the following issues:

1. Testing whether the number of the clients that can be serviced at the same time is greater than MaxCV when only 15% (a recommended patching value) of the buffer size is available.
2. Under the same condition, testing how many clients can be serviced when multicast is implemented. It is assumed that the client can receive from multiple channels. If for example the IO is equal to 2, then, it is divided as 1 for the multicast and one for the unicast. Otherwise, the algorithm handles it dynamically and according to the changes in the system.

Different file sizes have been used in these experiments. However, and for the sake of consistency, the parameters of the small file "Ontario" and a bigger one "Futurama" have been used so that the results can be compared with those of the previous experiments which used the same files.

Experiments Set 7:

For the Ontario file, the buffer size used is 50 while the buffer size of the Futurama is 215. These values represent approximately 15% of their files length respectively. The following table presents a summary of experiments set 7:

Experiments Set 7							
Objective	The objective is to test if the algorithm can still serve more clients than MaxCV when the buffer size is represented by the patch value which is 15 % of the file length.						
Parameters	No. of Clients	MaxCv	IO BW	Buffer size	Arrival delay	File size	Play time
	52 53	40	2 cv	15% of files length	1.59% of files play time	50MB	314
	52 53	40	2 cv	15% of files length	1.59% of files play time	220MB	1402

Figure 4.38 shows the amount of the pre-fetched reserve from the Ontario and Futurama video files when the clients have finished the service. Interestingly, this figure shows that both curves have the same tendency. Also, the same tendency can be shown for a sample of clients from Ontario and Futurama streams in figure 4.39 and figure 4.40 respectively. None of the clients in both experiments has registered frames loss.

Thereafter, another test has been conducted after adding one more client for the Ontario and Futurama files experiments. The objective was to saturate the server and measure the percentage of frames loss with each file. The number of clients has become 53. In both files a starvation has occurred and this starvation is reflected in figures 4.41 and 4.42 for client *i+30*.

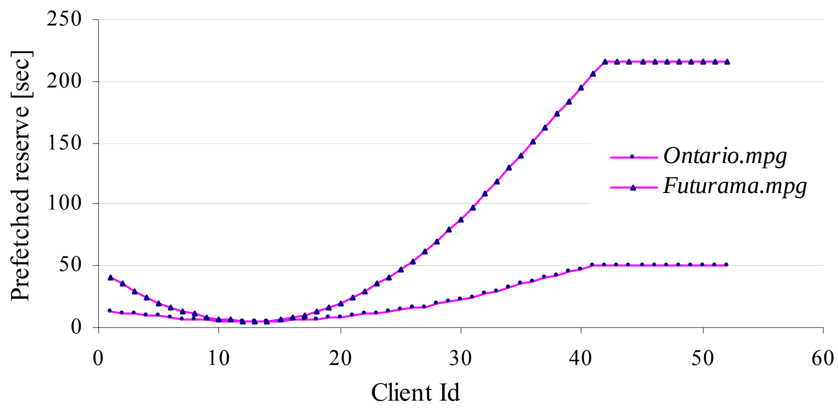


Figure 4.38. The PRV of 52 clients for each file

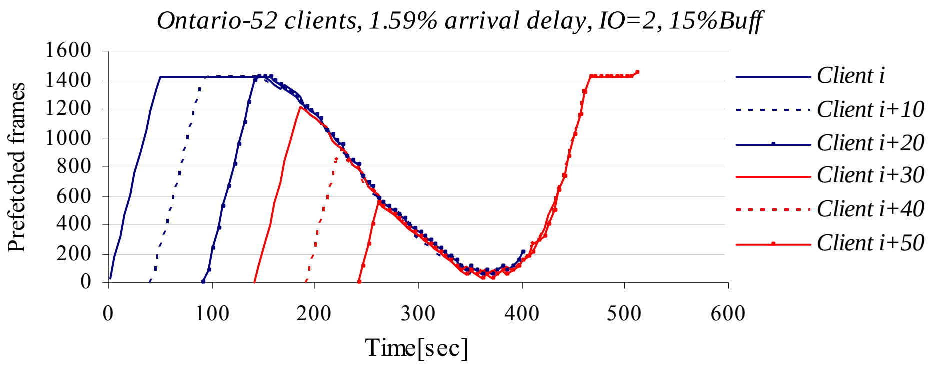


Figure 4.39. PRV with buffer size of only 15% of small file size

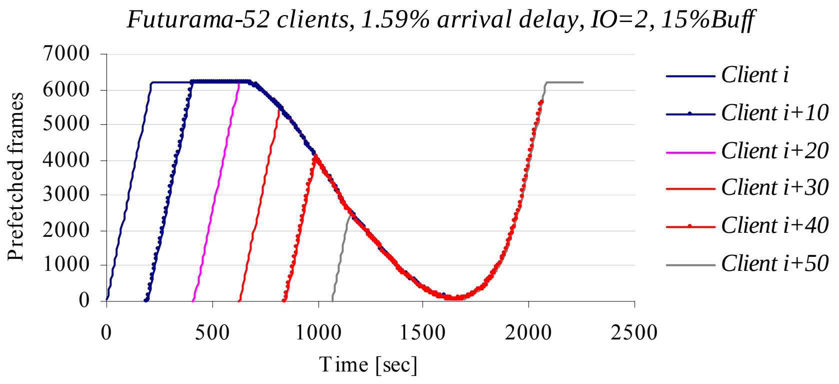


Figure 4.40. PRV with buffer size of only 15% of big file size

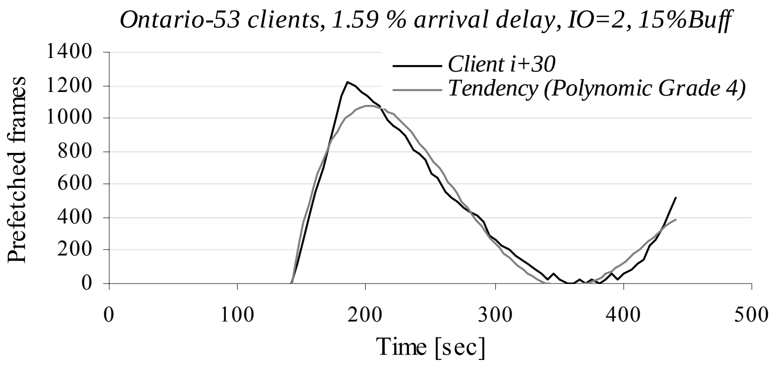


Figure 4.41. The PRV of client i+30 requesting small file

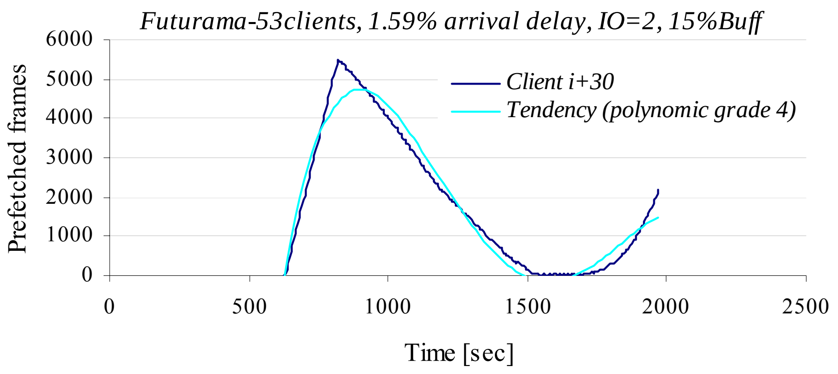


Figure 4.42. The PRV of client i+30 requesting big file

The percentage of the jitter is calculated as follows:

Ontario file:
Total slot = 522 Total slots which have witnessed jitter = 22 Total number of clients = 53 The avg. number of jittering slot per client = 22/53 ≈ 0.42 The average percent of jitter per client = 0.42/fileLength = 0.42/314 * 100 = 0,13%

The calculations for Ontario file show that the average suffer, which has been purposely permitted, is less than 1 second (≈0.42) for each client. This value represents 0,13% of the file length.

With respect to Futurama file, the same calculations have been done as follows:

Futurama file:
Total slot = 2331 Total slots which have witnessed jitter = 144 Total number of clients = 53 The avg. number of jittering slot per client = 144/53 ≈ 2.7 The average percent of jitter per client = 2.7/fileLength =2.7/1402* 100 ≈ 0,19%

The final results show some kind of estimation of the percentage of the jitter that could happen when the number of the clients is increased. If cheating is allowed, then the server can send less number of frames causing minus yet acceptable quality of service. That is, the users' eyes can't recognize the reduction in the play rate of the frames.

4.2.6 Stage Five: CB_MDA and Pre-fetching from Unicast Channels

Although it was possible to send work-ahead (pre-fetched reserve) video data over unicast channels in order to utilize the server resources and increase its performance, the unicast transmission is not recommendable since the system runs out of the resources very quickly. For example, if the clients continue to arrive with a very short period of time between their arrivals then, the server will be able to serve only as many client as many unicast it can supports (Maxcv). Consequently, very high bandwidth will be required in order to cope with the increase number of the clients.

To solve this problem, and as it has been explained in chapter three, multicast transmission presents a solution with which several clients who request the same video file within a short period of time can receive the video data from the same stream. However, to incorporate the later arrivals in the multicast streams, short unicast channels are used for transmitting the video prefix. The same technique of pre-fetching, which has been tested in the previous experiments, can also be applied to the short unicast channels in order to reduce their usage time.

In this stage, the experiments concentrate on showing that the combination of multicast and unicast transmission along with patching and pre-fetching techniques, which have been mentioned in chapter three, can reduce the required systems resources, especially the bandwidth, and increase the system performance in term of the number of clients served. Also, the finishing time of servicing the clients is reduced in comparison with the time that would be needed if the clients were served by unicast only and at a real-time rate.

Experiments Set 8:

The objective of the following experiments is to show the required bandwidth when the unicast is used and when the combination of multicast and unicast are used without and with pre-fetching. To maintain the consistency, the first set of experiments has been done with small files such as Ontario.mpg (Figure 4.43). The parameters are stated as follows: Maximum credit values that can be assigned (Maxcv) = 40, Video file length = 314, The average arrival time of the clients is 1.59% of the video play time (5 sec), The maximum video data that the client can buffer (Max Buffer) = 15%, The patching value is 15 percent of the file size, Maximum number of clients that have arrived (Max clients) = 61, The clients I/O bandwidth (Clients' IO) is 3, In case of buffering (pre-fetching) transmission, unicast is assigned 2CVs and the multicast is assigned only 1CV since not all the clients can receive from the multicast at 2CVs rate. The following table presents a summary of experiments set 8:

Experiments Set 8							
Objective	Testing the required bandwidth when only unicast transmission is used with that when the multicast and unicast are used implementing and not implementing pre-fetching.						
Parameters	No. of Clients	MaxCv	IO BW	Buffer size	Arrival delay	File size	Play time
	52 53	40	2 cv	15% of files length	1.59% of files play time	50MB	314
	52 53	40	2cv	15% of files length	1.59% of files play time	220MB	1402

	Mul. &	Uni	Mul. &	Uni.	Mul. &	Uni.	Mul &	Uni
Pref.	No	No	No	Yes	Yes	No	Yes	Yes

Figure 4.43 shows three curves which represents the required CVs when 61 clients are served by the unicast transmission only (the doted line), by multicast and unicast transmission without pre-fetching video data (the gray line), and by the multicast and unicast but with pre-fetching (the continued line).

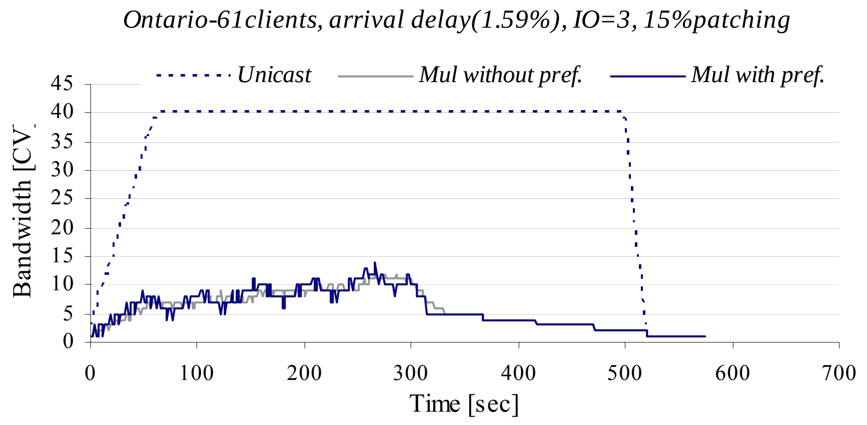


Figure 4.43. BW required of different techniques (Ontario file)

The graph shows that, with unicast transmission (the doted line), all the CVs have been consumed after a short period of time (at time 70 sec.). At this time (70 sec.), the algorithm registered 14 clients being in the service with 40 CVs assigned to them. With multicast and unicast implementing pre-fetching, the algorithm registered 14 clients yet with only 8 CVs.

Finally, with multicast and unicast implementing patching but without pre-fetching, the algorithm registered 14 clients and 7 CVs which is less than that of pre-fetching curve. This is true due to the fact that the algorithm utilizes the bandwidth availability by assigning more CVs to the short unicast channels in order to transmit the video prefix with pre-fetching to the later arrivals who join the multicast.

However, another look at some points of the graph, specifically at time 80, it can be seen clearly that the CV of the pre-fetching curve has drastically decreased to (3 CVs) while the non-pre-fetching curve has remained the same (7 CVs). The reason is that, with pre-fetching, the unicast channels finish transmitting the video prefix earlier, so these channels are released and become available for future reuse.

The next figures 4.44 and 4.45 shows the same experiments but with larger video files (Futurama and StarWar respectively) in order to show how the same clients (61 clients) are incorporated very quickly in the service when multicast is implemented reducing the required bandwidth.

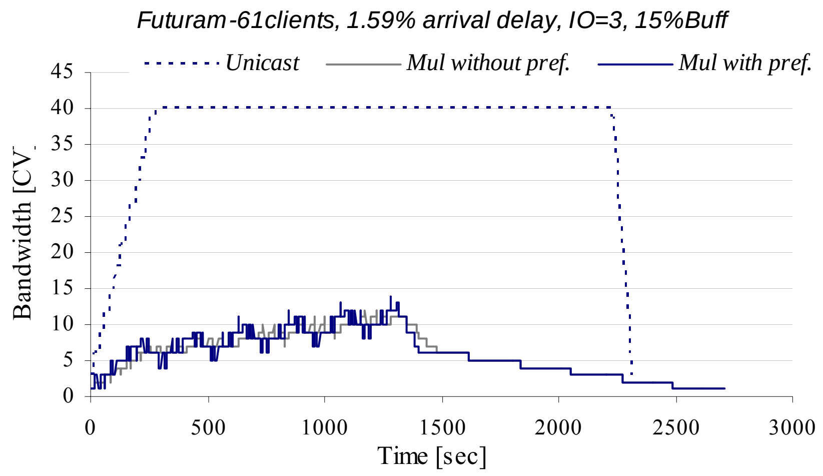


Figure 4.44. BW required of different techniques (Futurama file)

As a conclusion, although these experiments show that the unicast transmission has survived the starvation in the experiments of this stage, they show also that implementing unicast transmission is an inadequate solution for the VoD applications since it consumes high bandwidth.

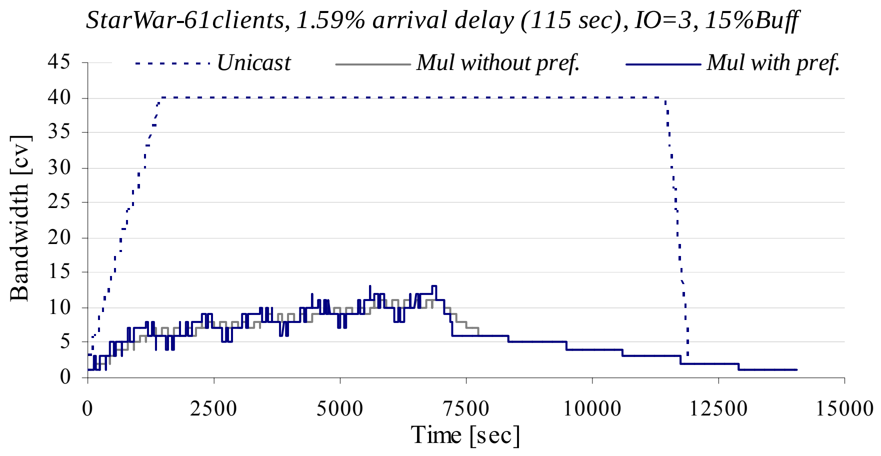


Figure 4.45. BW required of different techniques (StarWar file)

On the other hand, the multicast solution can drastically increase the system performance since the same number of clients who have been served by the unicast transmission can be served by multicast transmission with buffering and pre-fetching techniques yet with less number of CVs. Also, these figures show that, the combination of multicast and unicast with buffering and pre-fetching techniques have consumed only about 37% of the CVs. Thus more clients can join the service without reducing the quality of the service presented to the clients being in the service.

The above experiments have proven that multicast transmission can reduce drastically the required bandwidth. In addition, none of the above figures show that the bandwidth has been saturated. Therefore, other experiments have been conducted in order to check the algorithm behavior when more clients are accepted.

Experiments Set 9

The objective of these experiments is to check the required bandwidth when more clients are incorporated in the service. This is done by reducing the arrival delay so that more clients can present in a shorter period of time.

The video prefix in these experiments has been transmitted with and without pre-fetching in order to see how pre-fetching saturate the bandwidth but at the same time might reduce the finishing time. The parameters and characteristics of these experiments are stated as follows: Maximum credit values that can be assigned (MaxCV) = 40, video files length =1402 and 7200 sec., The average arrival time of the clients = 22 and 11 second for Futurama file, 57 (which is 50% less than that of the previous experiment),10, and 5 seconds for StarWar file, the maximum video data that the client can buffer (Max Buffer) = 15% of the video size, The patching value is 15% of the video size, Maximum number of clients that have arrived (Max clients) = 61, The clients I/O bandwidth (Clients' IO) is 3. The following table presents a summary of experiments set 9:

Experiments Set 9							
Objective	Testing the required bandwidth when more clients arrive and the transmission is implemented with pre-fetching						
Parameters	No. of Clients	MaxCv	IO BW	Buffer size	Arrival delay	File size	Play time
	61	40	3 cv	15%	1.59% 0.8%	220MB	1402
	61	40	3 cv	15%	1.59% 0.8%	1,13GB	7200
	105	40	3 cv	15%	5 sec	50MB 220MB 1,13GB	100 314 600 1402 1000 2000 3000 4000 7200

The short unicast channels are assigned 2CVs in order to create pre-fetching reserve, and the multicast channels are assigned only 1CV since not all the clients can offer 3CVs. If the client can offer only 2CVs then each channel is assigned 1 CV.

Figure 4.46 shows the result from Futurama file. It shows, on one side, a comparison between the transmission when the average delay arrival is 22 and 11 seconds and, on the other side, it shows the comparison between the transmissions when the unicast channels are creating pre-fetched reserve.

As the figure indicates, with arrival delay of 22 seconds no saturation has been witnessed, however the service has lasted longer. For example, the figure shows that, in the 22 second curve, the last arrival (client i+61) was able to receive the last fragment of the file around time 2500. However, when the average arrival delay has been reduced to 11 seconds, but without pre-fetching, the performance, in terms of the bandwidth and finishing time, has improved. For example, the figure shows that, in the 11 second curve with no pre-fetching, more bandwidth has been utilized but the last client received his video prefix at time 1865 resulting in a reduction in the service time. This means that, the algorithm was able to finish serving all the clients, when the delay was 11 seconds, before the time that would be needed if the arrival delay was 22 seconds.

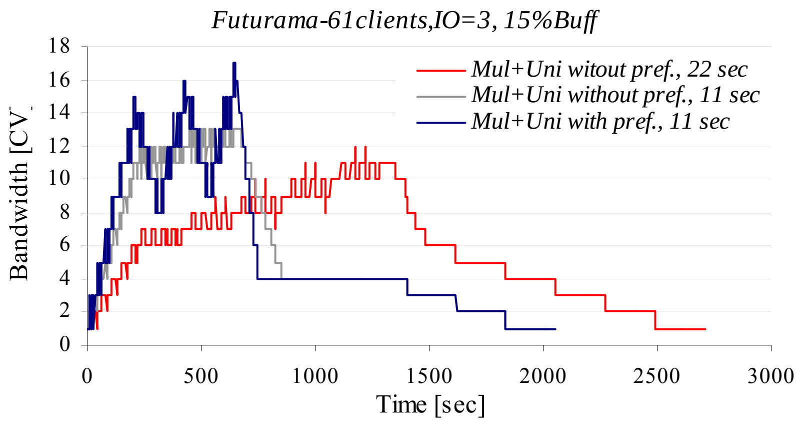


Figure 4.46. BW and time of different techniques (Futurama file)

Also, with respect to the curves of the 11 seconds with pre-fetching, the figure shows that, transmitting the video prefix at a faster rate permits utilize the available bandwidth and permits the server to reduce the usage time of the unicast channels causing the system to be able to reduce the finishing time and to face new arrivals. For example, figure 4.46 shows that the 11-second figure with pre-fetching is going up and down during the time 200 to 650. The reason for this behavior is that, the algorithm was utilizing the available bandwidth in order to transmit the video prefix at a faster rate resulting in releasing the short unicast channels earlier.

Figure 4.47 shows the same experiments presented in the previous figure but, with a bigger file (StarWar), in order to test the algorithm behavior. In these experiments the arrival delay has not only been reduced to 50%, but also to 90% (10 seconds). The figure shows the curves when the arrival delay is 57 seconds and 10 seconds. Also, it shows the curves of 10 seconds sending with and without pre-fetching video data.

With pre-fetching and at some point (time 620), the algorithm was able to utilize all the bandwidth so that the clients joined the multicast and got their video prefix faster. At time 971 the last client has already received the last fragment of his video prefix and at this time all the clients (61 clients) have already joined the video stream.

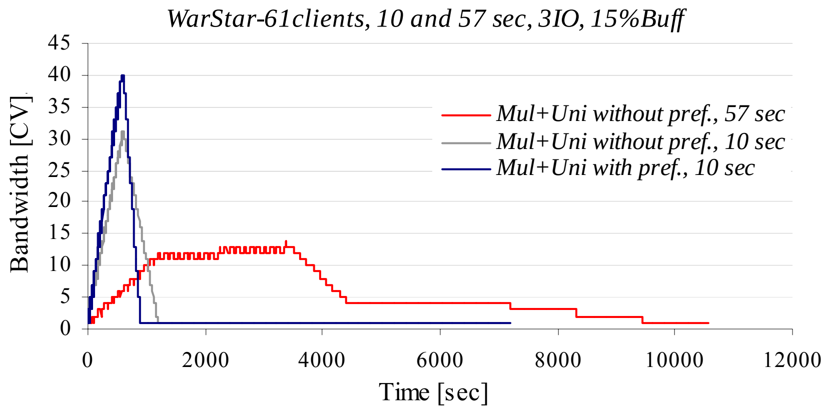


Figure 4.47. BW and time of different techniques (StarWar file)

As a conclusion, the algorithm has managed to incorporate all the later clients into the multicast stream by buffering the video data from the multicast stream and assigning unicast streams to these clients in order to receive the video prefix. Sending pre-fetched video data over the unicast channels has allowed the algorithm to utilize the available bandwidth and to reduce the service time of the video prefix resulting in releasing the unicast channels earlier.

This indicates that, the unicast channels are becoming a critical issue in the success of joining the multicast channels. Therefore, these unicast channels must be allocated and released as soon as possible. To solve this problem, the algorithm uses the unicast channels with the work-ahead (pre-fetching) policy which has been tested in the previous experiments. This technique has been explained with details in chapter three (3.4.4) and the next experiments are related to it.

Experiments set 10:

The objective of these experiments is to show how the algorithm assigns short unicast channels with a CV > 1 in order to reduce the service time needed for transmitting the video prefix. In these experiments, many different files with different sizes (100, 214, 600, 1024, 2000, 3000, 4000, 7200 sec) have been used. The buffer size is 15% of the files sizes. The patch value is also 15% of the files sizes. In the first set of experiments (figure 4.48) the IO value is 2CVs (1CV for the multicast channel and 1CV for the unicast channel).

In the second set of experiments (figure 4.49) the IO is 3CVs (1CV for the multicast and 2 CVs for the unicast channel). The following table presents a summary of experiments set 10:

Experiment set 10									
Objective	Testing the finishing time and the bandwidth when prefetching is applied to the short unicast channels and compare it with other variations.								
Parameters	No. of Clients	MaxCv	IO BW	Patch value	Arrival delay	File size	Play time		
	105	40	2 cv 3 cv	15%	5 sec	220MB 1,13GB 50MB 220MB 1,13GB	1402 7200 100 314 600 1402 1000 2000 3000 4000 7200		
		Mul. & Uni		Mul. & Uni.		Mul. & Uni.		Mul & Uni	
	Pref.	No	No	No	Yes	Yes	No	Yes	Yes

Figure 4.49 shows the required bandwidth and time for incorporating 105 clients. It also shows that the maximum number of unicast channels required with pre-fetching (16 unicast channels) is less than that of figure 4.48 (23 unicast channel). Also, the figure shows that the time required for transmitting the video prefix (approximately 670) is less than that of figure 4.48 (approximately 1005).

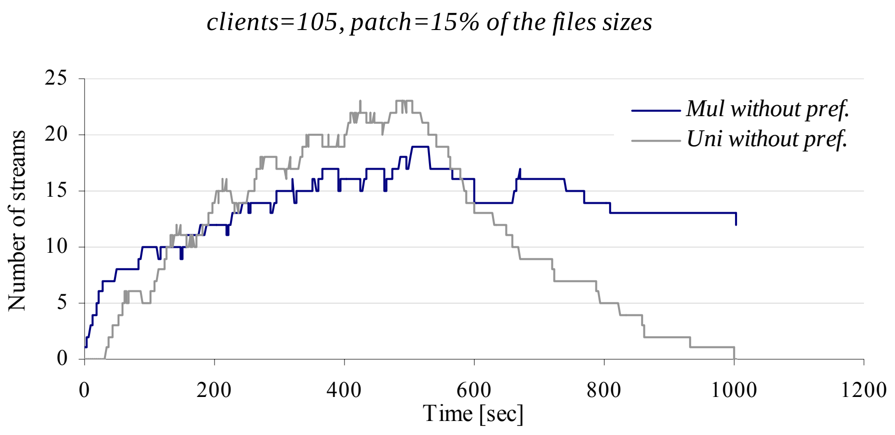


Figure 4.48. Multicast and unicast without pre-fetching

With regard to creating pre-fetching from the multicast channels, figure 4.50 shows that this choice is not recommendable since the maximum number of multicast and unicast channels is increased drastically. Figure 4.50 shows that at time (around 500 sec) the number of multicast streams reached 27 and the number of unicast streams reached 27 (a total of 54 active streams). The reason is that, when the multicast stream sends at a higher rate then the size of the video prefix that the unicast channel must transmit gets longer. As a result, the unicast spends more time for transmitting the video prefix. Furthermore, a subsequent joining client will even make the next unicast channel last longer. Therefore, in the final implementation, the priority for doing pre-fetching is given to the short unicast channels.

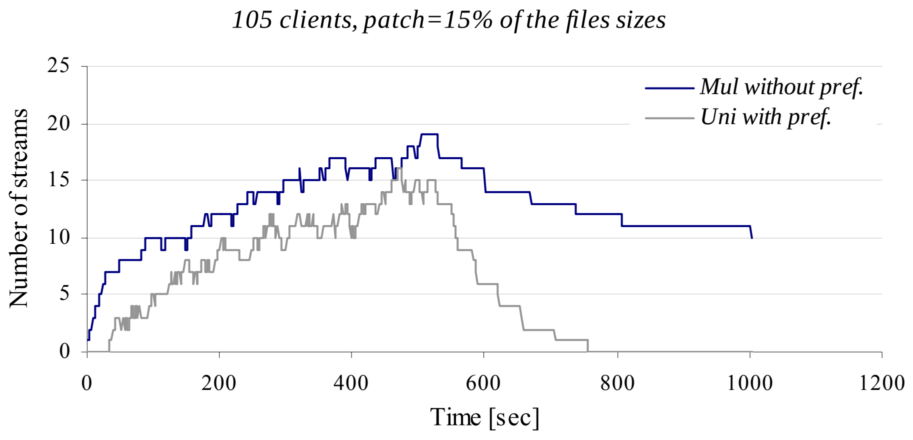


Figure 4.49. Multicast and unicast with pre-fetching

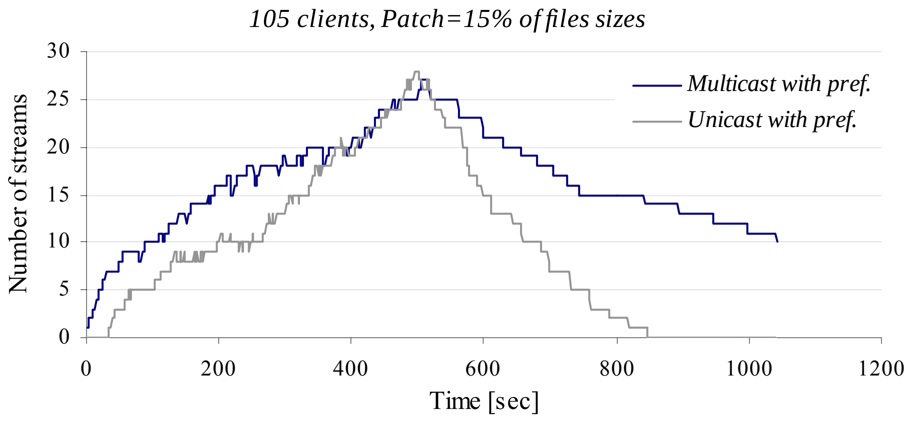


Figure 4.50. Pre-fetching with both, multicast and unicast

In general, the experiments have shown that the CB_MDA outperforms other delivery algorithm such as the batching and the classical patching, as well as the periodic broadcasting techniques. This performance is measured in terms of the response time, the number of request served by the server, and the implementation of VCR-Like commands. The remaining question is, when should the algorithm initiates a new multicast channel?

Experiments set 11:

Another issue which has been mentioned in chapter three is the decision when to initiate a new multicast channel. To answer this question, different experiments have been conducted with different patching values. These values are 15, minutes, 10 minutes, 5 minutes, 2.5 minutes, 15% and 10% of the file length. The reason for choosing between percentages of files sizes and fixed amount of patching is to compare between them especially when the files sizes are small.

For example, if the video file length is less than 10 minutes then a patching of 10 does not include this file. With the percentage, however, these short files will have part of them patched. This way, the analysis of patching will be more logical. The following table presents a summary of experiments set 11:

Experiment set 11									
Objective	Find out an optimal value for patching								
Parameters	No. of Clients	MaxCv	IO BW	Patch value	Arrival delay	File Size		Play time	
	105	40	3	15%	5 sec	220MB		1402	
						1,13GB		7200	
								100	
						50MB		314	
								1402	
						220MB		1000	
								2000	
								3000	
								4000	
						1,13GB		7200	
		Mul. & Uni		Mul. & Uni.		Mul. & Uni.		Mul & Uni	
	Pref.	No	No	No	Yes	Yes	No	Yes	Yes
	Patch	15%		15%		15%		15%	
	Patch	10%		10%		10%		10%	
	Patch	5 Min		5 Min		5 Min		5 Min	
	Patch	2,5 Min.		2,5 Min.		2,5 Min.		2,5 Min.	

With regard to these experiments, they have been conducted without pre-fetching from both or either of the multicast and unicast channels, and with pre-fetching from both or either of the multicast and unicast channels. The next figures (4.51, 4.52, 4.53, and 4.54) show only the cases when the CB_MDA does not create pre-fetching from the unicast channels, however this technique is applied in the real implementation. Figure 4.51 shows some kind of balance in the number of unicast and multicast channels.

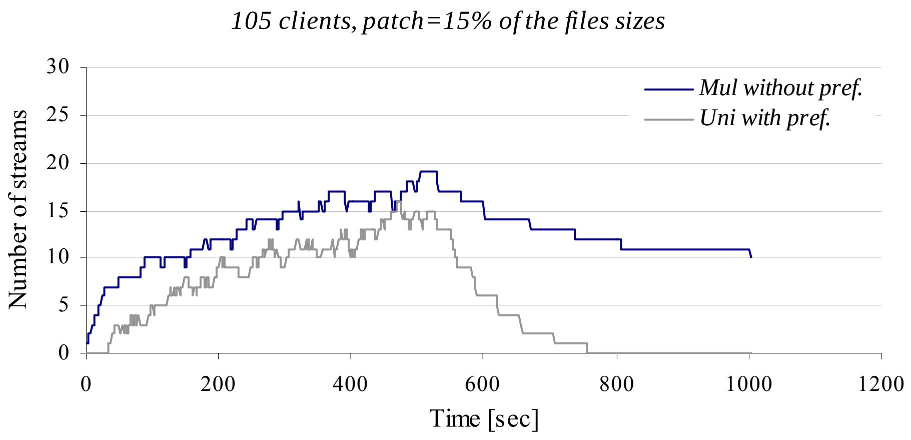


Figure 4.51. No. of streams when patching is 15% of the file length

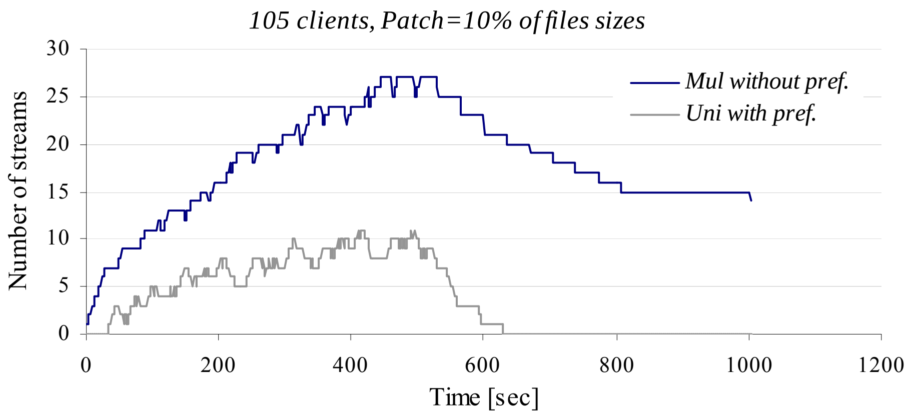


Figure 4.52. No. of streams when patching is 10% of the file length

Figure 4.52 shows that, when patching is equal to 10% of the files sizes then the multicast channels are increased however, the finishing time of the unicast channels is reduced. When patching is equal to 5 minutes (figure 4.53), the finishing time of the unicast channels is even less than that of the previous figures.

Finally, figure 4.54 shows that when the patching value is equal to 2,5 minutes the results are close to that of figure 4.51. As a result, we recommended a patching value of 15% of the files sizes.

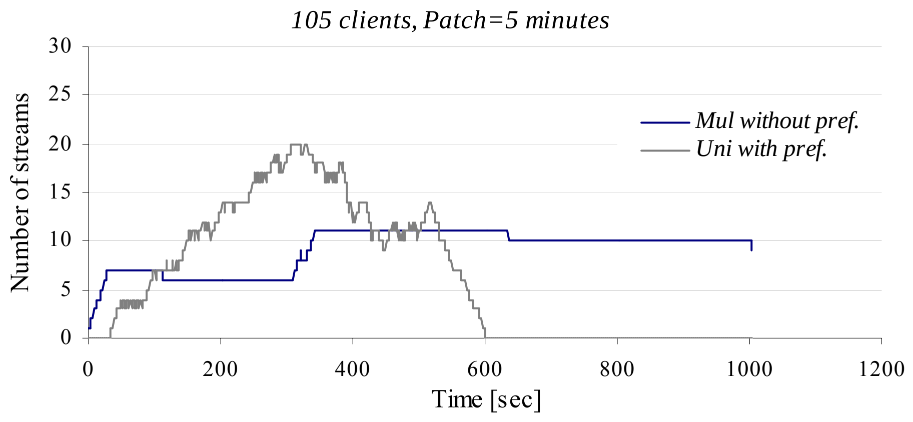


Figure 4.53. No. of streams when patching is 5 minutes

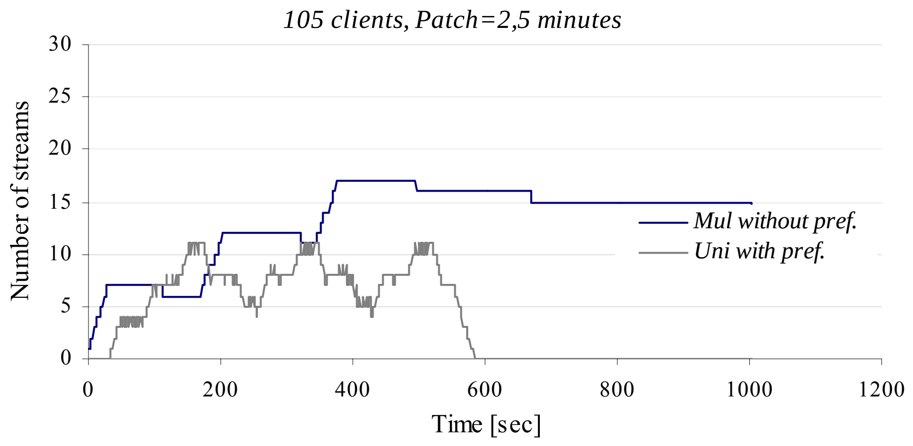


Figure 4.54. No. of streams when patching is 2,5 minutes

In general, the experiments which have been carried out in the fourth and fifth stages have proved the effectiveness of implementing the CB_MDA, which controls the flow of the video data from the server to the client by using a combination of multicast and unicast channels and by implementing patching and pre-fetching techniques. These results have driven us to make the following assertions:

- ***Assertion 1***: The algorithm manages to use combination of multicast and unicast channels in order to provide true video on demand.
- ***Assertion 2***: The algorithm permits the joining of new clients to appropriate multicast streams through unicast channels.

- ***Assertion 3***: The algorithm reduces the usage time of the unicast channels which are needed for serving short requests such as joining the multicast channels.
- ***Assertion 4***: As a result to the above assertions, more requests are served in comparison with the experiments of stage three when the same parameters and machines specifications are applied.
- ***Assertion 5***: The algorithm has achieved balanced number of multicast and unicast channels when 10 minutes of patching is applied. However, the channels balance has no effect on increasing or decreasing the number of requests served by the server. The ability of the server to adapt to the changes that take place in the system is more effective, and this is done by the algorithm.

As a conclusion to chapter four, the experiments have proved that there are so many factors affect the performance of the VoD server such as the server capacity, the client capacity, the scheduling algorithm and the transmission type. In the UAB labs, we were able to mount a stable version of the server and run a set of clients with real movies, and we have shown how multicasting, patching, and pre-fetching techniques function at the server side, and how the clients have connected to the server, synchronized with it in order to receive from one or multiple channels, and were able to open multiple video windows with the same and with different movies.

Chapter 5
Conclusions

5.1 Conclusions

The major parts which have been investigated in this work are the admission control subsystem (ACM), the resource management subsystem (RMM), and the credit based media delivery algorithm (CB_MDA).

The ACM works as a connecting point between the system's modules. It acquires the clients' requests, creates threads for negotiating with clients, and requests the necessary resources from the RMM. Then, it decides, based on the information received from the RMM, whether a request can be attended to, and, given that a request cannot be attended to because of the lack of resources in the system, it can check with the CB_MDA to see if the request can be scheduled soon.

To facilitate the acquisition of the requests and the negotiation with the clients, the ACM creates a thread for each client for this purpose. This thread sends a list of the available video files to the client and informs the client about the port from which he will receive the video data.

With regard to the requests of the VCR commands, the ACM forwards these requests to the CB_MDA in order to take an action based on the command type (Stop, Pause, Play, Jump forward, jump backward).

The RMM has been designed and implemented to provide the necessary resources for a specific request. Also, it makes sure that the acceptance of a new client does not affect the performance requirements of other videos already being in service, and, that adequate resources are available throughout the entire path from the video server to the client displaying device.

To achieve these objectives, the RMM translates the request into QoS parameters, requests the necessary resources and reserves them. This process has been achieved by using four brokers (threads) which are: *CPU broker, Memory broker, Network broker,* and *Disk broker*.

The analysis of these brokers is centered on four policies: Maximum Policy (MP), Adaptive Maximum Policy (AMP), Average Policy (AP), and Adaptive Average Policy (AAP). Each broker works separately and reports to the RMM about its status based on one of these policies. If all of them report positive confirmations to the RMM, the petition is then accepted and registered in the admitted users list.

The CB_MDA has been designed and implemented to support a scalable video delivery from the server to the clients. The main objective of the CB_MDA is to regulate and control the video data streaming to the appropriate clients, to block the client who might affect the quality of service provided to the ones being in service, to achieve an acceptable response time, and to handle the requests related to the VCR functions.

The algorithm (CB_MDA) gets informed about the arrival of new requests from the ACM, and about the resources availability from the RMM. The idea of the CB_MDA is to better utilize the system resources. That is, instead of reserving the resources for each request by the RMM, the RMM makes these resources available to the algorithm which will be in charge of assigning these resources to the appropriate requests.

In its implementation, the CB_MDA is capable of serving groups of clients through multicast and unicast streaming. The algorithm uses meaning of patching so that, the later arrivals who join the multicast streams can buffer the data from the arriving point meanwhile, the algorithm assigns unicast channels in order to transmit the video prefix (the initial part of the video).

In addition, the algorithm takes into considerations that the clients can be heterogeneous. That is, some clients have better machines specifications, in terms of bandwidth and buffer space, than others. Therefore, it uses the periods of time when the server has availability in its resources and assigns these resources to the appropriate clients. Thus, these clients might get finished before the time that would be needed if the client had low capacity machines.

The client, on the other side, is synchronizing with the server so that it can receive from multicast channel or from multiple channels (Multicast and unicast) in order to receive the video prefix and buffer the video data received from the multicast channel.

5.2 Contributions

This thesis makes contributions related to the design of the server VPS and the scheduling algorithm CB_MDA:

The VPS Contributions:

- The design and implementation of a multithreaded video on demand proxy server (VPS) that can work as a building block for constructing distributed video on demand (DVoD) architecture.
- The design permits the scalability of the VPS without significant changes due to the fact that the VPS architecture is composed of standalone modules which are ACM, RMM, and CB_MDA.
- The design allows the separation of the ACM from the VPS in order to make a replication of the other modules on a different machine so that the service can be distributed. In this case, the ACM still work as a central point for all replicated servers.
- The ACM achieves an acceptable short response time by implementing multi-threaded mechanism for negotiating and attending to the clients.

The CB_MDA Contributions:

- The experiments show that the Credit_Based Media Delivery Algorithm (CB_MDA) is able to provide true video on demand service, guarantee on time delivery, and handle the VCR-Like functions (play, pause, jump forward, jump backward, and stop).
- The CB_MDA guarantees the QoS of the clients being in service by blocking all the requests which might cause starvation in any of these clients.
- The CB_MDA algorithm does not require reserving dedicated resources for serving the VCR-functions allowing more requests to be handled.
- Also, the CB_MDA algorithm does not require dedicating specific number of multicast or unicast channels giving more flexibility in handling all kind of video type (popular and unpopular).

- The (CB_MDA) implements patching and pre-fetching techniques in order to increase the performance and supply instantaneous VCR commands as well as increase the efficiency of the proxy server. It can also adapt to the system changes dynamically switching users from one mode to another (multicast to unicast).
- The CB_MDA reduces the usage time of the unicast channels required for transmitting the video prefix allowing the release of the unicast channels earlier.
- Like the classical patching algorithm the CB_MDA achieves a similar short response time since it assigns a unicast channel for transmitting the video prefix in order to start the reproduction immediately.

Experimental Study:

- In addition to the server capacity, the experiments have shown that the client's buffer and IO bandwidth play an important role in increasing the server performance. For example, if the client can provide a moderate buffer size equal to 25% of the file size and an IO equal to 2 CVs (multiple channels), then the system performance, in terms of the number of clients that can be served, is increased by, approximately, 50%.
- Any increase in the client's buffer space should be accompanied by an increase in the client's IO bandwidth and vise versa.
- Using the same parameters, when the algorithm did not use combination of multicast and unicast channel, and without implementing patching and pre-fetching, the CB_MDA has served up to 60 clients before registering starvation. With the use of multicast and unicast channels and patching and pre-fetching techniques, the algorithm served up to 105 clients, yet only about 40% of the resources have been used.
- The algorithm can mange to have balanced number of multicast and unicast channels so that the system does not run out of unicast channels by creating excessive multicast channels, and does reduce the system performance by creating more unicast channels.
- As a result of this investigation, we have created real video on demand server that implement the above techniques and allow the users to view real video data. In addition, we have created a real experimental platform that can be used to conduct more experiments and to change the parameters

values as needed. This is would be important for future modification and enhancements.

The Client

- The client architecture provides interoperability so that other players can be used such as Xine.
- Also, the client provides a GUI for communicating and negotiating with the server. The GUI is consistent with other well-known players that can be found in the market.

5.3 Future Work

This section discusses future research areas that can further expand the capabilities of the VPS:

1. Users' behavior: the user behavior plays a great role in determining the way the video stream must be transmitted. For example, if the user can be convinced to wait few minutes before he can start the video reproduction, then this will allow the algorithm to collect the requests related to the same video and serve them at once. This requires that the algorithm should work as a hybrid algorithm were patching, pre-fetching and batching techniques are implemented.
2. Resources calculation: the video metadata shows that the frames sizes vary from moment to moment. Therefore, it will be beneficial to create, ahead of the time, a curve line for the resources needed for each video file and then schedule the streams in way where the points of high resources of one video can coincide with those points of low resources of another video.
3. Machines specifications: this thesis has shown that there were moments when high machines specifications can improve the server performance. An example of an environment where the machines specifications vary from one place to another is the schools and universities' departments. Therefore, if the video server is to work in such environments then it would be better, if possible, to schedule those who have low machines specifications in the middle since, when the system gets saturated it will sends at the lowest possible rate. That is, the clients gets only what they see without pre-fetching.

4. Storage disk: the disk broker implemented by our group requires more modification so that the issue of caching and mirroring can be implemented.
5. Client extension: the client must be able to synchronize with the server especially in the case of multicasting and patching. The group of Xine, a multimedia player, has shown a great interest in incorporating multicasting and unicast reception of the video data at the same time in their player. Therefore, creating a plug-in is needed for Xine player.

5.4 Publications and Activities Related with this Dissertation

5.4.1 Publications Related to the Dissertation

Latest and preliminary versions of the overall work have been published in proceedings of several national and international conferences:

- B. Qazzaz, R. Suppi, F. Cores, A. Ripoll, P. Hernandez, E. Luque, "Providing Interactive Video on Demand Services in Distributed Architecture", Proceedings of 29th EUROMICRO Conference, pp. 215-222, Belek-Antalya, Turkey, September 2003.
    - This paper presents the architecture of the VoD server as a building block in the DVoD architecture.
    - Also, it shows the working principle of the multimedia delivery algorithm CB_MDA.
    - The algorithm uses a combination of multicast and unicast channels and implements patching and pre-fetching techniques in order to provide interactive video on demand and reduce the usage time of the unicast channels by sending work-ahead video data.
- B. Qazzaz, R. Suppi, F. Cores, A. Ripoll, X. Yang, P. Hernandez, E. Luque. "Admission Control Policies For Video On Demand Brokers", IEEE ICME: International Multimedia & Expo Proceedings, pp. 529-532, Baltimore-Maryland, USA, July 2003.
    - This paper presents four admission control policies that can be implemented in order to reserve the necessary resources so that the contracted QoS can't be jeopardized.
    - Also, it presents four possible brokers where the admission policies are implemented. The resources brokers are CPU, Memory, Network, and Disk.

- B. Qazzaz, R. Suppi, E. Luque, “Video on Demand: Design and Implementation of a Scalable Server”, II Congreso Internacional Sociedad de la Información y del Conocimiento (CISIC), pp. 311-321, Madrid, Spain, Mayo 2003.
    - This paper combines the system brokers and the media delivery algorithm for providing true video on demand.
    - It provides dynamic adaptation to the changes that might occur in the system and presents the server as an educational tool at the classroom.
- R. Suppi, B. Qazzaz, E. Luque. "Design and Development of a Multithreaded Video on Demand Server for LAN", *SCI/ISAS 5th World Multi-conference on Systemics Cybernetics and Informatics*, Vol. XII. pp. 380-388, Orlando, Florida USA, July 2001.
    - This work presents the first version of the video architecture.
    - It provides interactive video on demand by dedicating a unicast channel for each user.
- E. Luque, A. Ripoll, R. Suppi, P. Hernández, T. Diez, J.A. Marco, F. Cores, B. Qazzaz, R. del Castillo, “Diseño e implementación de un sistema Interactivo de vides bajo Demanda”, Septiembre 2000.
    - It presents a general overview of the VoD system.
    - It presents the main issues related to VoD system with the focus on the QoS.
- F. Cores, A. Ripoll, B. Qazzaz, R. Suppi, X. Yang, P. Hernandez, E. Luque, “Exploiting Traffic Balancing and Multicast Efficiency in Distributed Video-on-Demand Architectures”, 9th International Euro-Par Conference, pp. 859-869, Klagenfurt, Austria, August 2003.
    - Our contribution to this work is to implement our architecture as a proxy server.
    - Provide interactive video on demand by implementing the CB_MDA algorithm.
    - This work has been selected as on of the best papers presented to the Euro-par conference.
- F. Cores, A. Ripoll, X. Yang, B. Qazzaz, R. Suppi, P. Hernandez, E. Luque, “Improving Bandwidth Efficiency in Distributed Video-on-Demand Architectures”, Parallel Processing Letters, Vol. 13, No. 4, pp 589-600, December 2003.

  - This paper came as a result of the previous presented in the 9th Euro-par conference. The previous work has been selected as one of the best contribution to the conference. Therefore, further improvement has been made in order to improve the bandwidth efficiency.

5.4.2. Technical Demo

This work has been accepted as a technical demo at the International Conference on Information Technology: Research and Education (ITRE03).

- B. Qazzaz, R. Suppi, E. Luque, "Video on Demand Proxy: Design and Implementation", ITRE2003 International Conference on Information Technology: Research and Education (ITRE), New Jersey, USA, August 2003.
  - This demo shows the transmission of the video data from the disk to the client.
  - It shows clearly how the algorithm adapts to the characteristics of the clients' machines specification.

5.4.3 Other Publications

During my work on my Ph.D. dissertation, and as a partial time, I have worked with a European Project called Forest Management and Monitoring System FOREMMS. In FOREMMS, I have developed an application for the Hand-held devices which work with GPS systems. The application has been deployed and used by FOREMMS partner from Finland, Italy and Poland. In addition, part of the design and the implementation has been presented and accepted at the EnviroInfo2000 conference:

- B. Qazzaz, B. Abdalhaq, D. Tamajon, D. I. Rexachs, E. Luque, "A System for Data Collection of Environmental Information", Proceedings of the 16th International Conference Informatics for Environmental Protection, pp. 428-431, Vienna, Austria, September 2002.

5.4.4. Work Related with Undergraduate Final Projects

After the creation of the first version of the work in the year of 2000, many students have become interested in the subject and shown their willingness to work in this subject. Therefore, we facilitated to them the access to what we were doing, and supervised and directed their work until they have finished and presented it as a final project and as a requirement for their graduation.

Some of the final projects which have been produced from our work while it was under investigation are the following:

- Mathias Beck, “Usage of Multicasting for Video-On-Demand Servers”, Department of Computer Science, Technical University of Clausthal, Clausthal-Zellerfeld, Germany, January 2004.
- Jens Lichtenberg, “Multicast Based Client for Video on Demand Services”, Department of Computer Science, Technical University of Clausthal, Clausthal-Zellerfeld, Germany, January 2004.
- Jordi Valls, “Visualizador de MPEG para Sistema de Video bajo Demanda”, Escola Universitaria d’Informàtica, Sabadell, Spain, Septiembre 2002.

Appendix A
Media Streaming Algorithms Summary

A.1. Introduction

The performance of any video on demand (VoD) system depends on the way the media data is streamed and delivered from the server to the client. Several algorithms have been put forward to serve this purpose. These algorithms were changing with the time as a result of the advancement technology in the computers, networks, and storage technologies.

In the past, the computer systems did not have large storage nor, they were able to have high Input/output bandwidth. Likewise, the early networks were not as fast as the today’s ones. As a result, and with this continuous progress in the computer technology, the computer applications such as VoD applications are becoming consuming more than what they were consuming before in terms of bandwidth and memory.

Also, the users have become more willing to receive the best quality of service (QoS). Therefore, there was always a need for providing video streaming mechanisms which can make better utilization of the system resource and provide an acceptable quality of service to the users.

During the last two decades, several algorithms dedicated to scheduling the media data delivery and streaming over the net work have been presented. These algorithms have considered unicast and multicast scheduling using different policies such as, deadline scheduling algorithms, batching, piggybacking, chaining, tapping, catching, patching, pre-fetching, Pyramid broadcasting, permutation-based pyramid broadcasting, skyscraper broadcasting, harmonic broadcasting and their variants. In general, there are two schemes for allocating server channels for media data delivery called user-centered and data-centered. These algorithms are explained in the following subsections.

A.1.1. User-Centered Approach

A conventional VoD system assumes the *user-centered* scheduling scheme [AGG96b] [VIS96] in which a user eventually acquires some dedicated bandwidth. The consumption rate of a video object is equal to the amount of bandwidth necessary to view it continuously. When a client makes a request to the server, the server sends the requested object to the client via a dedicated channel. This scheme incurs high system costs, especially in terms of server storage-I/O and network bandwidths. To maximally utilize these channels, researchers have proposed efficient scheduling techniques [DAN96d] [FRE95] [HUA97b] [KEE95] [OYA95] [OZD94] [OZD95] [WAN00]. These techniques are said to be "user-centered," because channels are allocated to users, not data or objects. These algorithms simplify the implementation, but dedicating a stream to each viewer will quickly exhaust the server resources and the network bandwidth.

A.1.2. Data-Centered Approach

Data-centered scheduling dedicates channels to video objects, instead of users. It allows users to share a server stream by batching and using the multicast facility of modern communication networks. Also, it has the potential for dramatically reducing the network and server bandwidth requirements. The data-centered multicast VoD service can be either *client-initiated* or *server-initiated* [SHE95][GAO99].

- In the client-initiated service, channels are allocated among the users and the service is initiated by clients, so it is also known as a *scheduled* or *client-pull* service.
- In the server-initiated service, the server channels are dedicated to individual video objects, so it is also called a *periodic broadcast* or *server-push* service. Popular videos are broadcast periodically in this scheme, and a new request dynamically joins, with a small delay, the stream that is being broadcast.

- Another option would be would be combining the above two schemes. This combination is called hybrid batching. In practice, it is efficient to use *hybrid batching* that combines the above two schemes.

The Credit Based Media Delivery Algorithm (CB_MDA), which has been presented in chapter three, is a simple yet a robust algorithm and it is data-centered and belonged client-initiated scheme (client-pull). The main objective of the CB_MDA is providing TVoD service, short response time and scalability.

Scalability is defined as the ability of the system to scale well as the number of the clients is increased. That is, for the server it does not make any (or much) difference whether 1 or 100 clients are connected to the server. To achieve these objectives, the CB_MDA uses multicast and unicast channels for media data transmission.

The unicast channels are used as short ones to transmit the video prefix to a later arrival client while the client is merged to an appropriate multicast stream and buffering the media data from it. Also, the unicast channels are used to serve interactive service such as the case when the client invokes any of the VCR-like functions.

During the delivery process, the algorithm takes in consideration the server's resources and the client's buffer space and I/O bandwidth. That is, it tries to explore and discover the possibility of sending video data over the unicast channels at a rate of the consuming rate so that, the busy (usage) time of the unicast channel is reduced. This way, the system will have more available unicast channels for serving other requests.

The delivery of video data to the client involves three steps: (1) reading block of video data from the disk (storage node) to a buffer, (2) transmission of the block from the storage node to the network node, (3) the transmission of the block from the network node to the consumer's desktop.

A video server has to supply data blocks of the movie at regular periods to the client. If data is not transferred at regular intervals, the client may experience glitches in the delivery of the movie. To ensure glitch-free service, the video server has to guarantee finishing the three steps of service in a fixed amount of time. Guaranteeing delay bounds in step (1) can be addressed by appropriate disk scheduling [YAN03]. The problem of ensuring delay bounds in service steps (2) and (3) has been addressed in chapter three and four having in mind that the storage node can be also a network node.

As a continuation, in the following sections some of the most famous algorithms are explained.

A.2. Deadline scheduling

Deadline scheduling [LIU73][JEF91] are shown to be optimal however, both of these studies assume that the task completion times are known in advance, but this is not realistic from the point of view of the user. Another deadline-based scheduling algorithm is presented in [ROT97] where the algorithm imposes a time delay, between the arrivals of users, which makes the response time high.

A slot-based algorithm presented in [RED99] addresses the block transmission from the storage node to the network node over interconnected network but does not consider the block transmission from the network node to the client. Reddy [RED95] has proposed a movie-scheduling algorithm in a multiprocessor video server. His solution minimizes the contention for links over the interconnection network. He assumes the homogeneous set of streams that contains the same playback rate streams. In fact clients are likely to request heterogeneous streams of which playback rates are different from each other.

A.3. Batching

Another approach called batching [DAN94] has been proposed to schedule video requests at the cost of undesirable latency to respond to client requests. The idea was to queue users who request the same video data then, to assign a single multicast channel instead of multiple independent unicast channels in order to reduce resource requirements.

Further studies [LI96] and [SHA97] have been realized to refine the batching policies by incorporating knowledge of the next stream-completion time to reduce the server capacity required to achieve similar throughput and viewer turn-away probability.

The problem with batching is that, it causes undesirable latency and does not support interactive VCR-like control since multiple users share a multicast channel.

The equally-spaced batching mechanism has a fixed-maximum service latency and supports NVoD interactivity, but its usually large service latency may cause some clients to renege.

A.4. piggybacking

A set of well-known approaches have been presented to improving VoD systems efficiency, such as piggybacking [GOL95] [GOL96] [AGG96a] where display rates are dynamically adjusted so as to bring different streams to the same file position, at which time the streams can be merged. The idea is similar to that of batching [DAN94], but the grouping is done dynamically and while the displays are in progress. However, the slow merging rate limits the system's potential (e.g. two streams with 3 minutes apart take 30 minutes to merge). Also, it requires that the server stores (or computes in real time) extra encoding of the media data that is delivered most often. Note that, the reduction in the I/O demand is not quite as high as in the case of batching, since some time must pass before the streams can merge; hence, the tradeoff (between these two techniques) is between latency for starting the service of a request and the amount of I/O bandwidth saved.

A.5. Chaining

Chaining [SHEU97] is also a generalized dynamic multicast technique to reduce the demand on the network-I/O bandwidth by caching data in the client's local storage to facilitate future multicasts. Thus, data are actually pipelined through the client stations residing at the nodes of the respective chaining tree, and the server serves a "chain" of client stations using only a single data stream. The advantage of chaining is that not every request has to receive its data directly from the server. A large amount of video also becomes available from clients located throughout the network. This scheme scales well because each client station using the service also contributes its resources to the community. Hence, the larger the chaining trees, the more effective the application can utilize the aggregate bandwidth.

A.6. Stream Tapping

The authors of [CAR97] present *stream tapping* that allows a client to greedily "tap" data from any stream on the VoD server containing video data he can use. This is accomplished through the use of a small buffer on the client side and requires less than 20% of the disk bandwidth used by conventional systems for popular videos.

A.7. patching

To eliminate the service latency, *patching* was introduced in [HUA98]. The objective of patching is to substantially improve the number of requests each channel can serve per time unit, thereby sufficiently reducing the per-customer system cost.

In the patching scheme, channels are often used to patch the missing portion of a service or deliver a patching stream, rather than multicasting the video in its entirety. Given that there is an existing multicast video, when to schedule another multicast for the same video is crucial.

The time period after a multicast, during which patching must be used, is called the *patching window* [HUA99].Two simple approaches to setting the patching window are discussed in [HUA98]. The first one uses the length of the video as the patching window. That is, no multicast is initiated as long as there is an in-progress multicast session for the video. This approach is called the *greedy patching* because it tries to exploit an in-progress multicast as much as possible. However, an over-greedy can actually reduce data sharing [HUA98].

The second approach, called the *grace patching*, uses a patching stream for the new client only if it has enough buffer space to absorb the skew. Hence, under grace patching, the patching window is determined by the client buffer size. Considering such factors as video length, client buffer size, and request rate, the authors of [CAI99] generalized patching by determining the optimal patching window for each video.

An improved form of patching, called as the *transition patching* [CAI99], uses either a patching stream or a transition stream and improves performance without requiring any extra download bandwidth at the client site. Other optimal patching schemes were described in [EAG99] [SEN99].

In patching, a client might have to download data on both regular multicast and patching channels simultaneously. To implement patching, a client station needs three threads: two data loaders to download data from the two channels, and a video player to play back the video.

The *controlled CIWP* (Client-Initiated-With-Prefetching) [GAO99] is another multicast technique similar to patching and tapping for near instantaneous VoD service. The novelty of the controlled CIWP is that it uses a threshold to control the frequency of multicasting a complete video stream. It uses simple FCFS channel scheduling so that a client can be informed immediately of when its request will begin service.

A.8. Famous Broadcasting Protocols

Broadcasting protocols share the common objective of reducing the total bandwidth required. They repeatedly broadcast the same video over several channels in such a way that a customer may have to wait for few minutes (e.g. 5 minutes) before he can start watching the video. In [PAR99], the author states two factors that make broadcasting protocols accepted:

1) The saving is considerable since about 40% of the demand is for small number of popular movies (10 to 20) [CLA93] [DAN94] [DAN96].

2) any reduction in the cost of distributing popular videos through the use of more efficient broadcasting protocols will thus have a direct impact on the overall cost of VOD and, ultimately, on its success on the market place.

A.8.1. Equally-spaced interval Broadcasting

One of earlier periodic broadcast schemes was the *Equally-spaced interval Broadcasting* (EB) [DAN94]. Since it broadcasts a given video at equally-spaced intervals, the service latency can only be improved linearly with the increase of the server bandwidth. The author of [BAN94] also proposed the staggered VoD which broadcasts multiple copies of the same video at staggered times.

Pyramid broadcasting [VIS96], permutation-based pyramid broadcasting [AGG96b], skyscraper broadcasting [HUA97], harmonic broadcasting [JUH97] and its variants [PAR98], all, share the same goal and a similar organization. They divide each video into segments that are simultaneously broadcast on separate data streams. Segments protocols can be subdivided into two groups: 1) Pyramid Broadcasting, Permutation-based Pyramid Broadcasting and Skyscraper Broadcasting protocols. 2) Harmonic Broadcasting and its variants.

A.8.2. Pyramid Broadcasting

To significantly reduce the service latency, *Pyramid Broadcasting* (PB) was introduced in [VIS95]. In PB, each video file is partitioned into the segments of geometrically-increasing sizes, and the server capacity is evenly divided into K logical channels. The i-th channel is used to broadcast the i-th segments of all videos sequentially. Since the first segments are very small, they can be broadcast more frequently through the first channel. This ensures a smaller waiting time for every video.

A drawback of this scheme is that a large buffer — which usually corresponds to more than 70% of the video—must be used at the receiving end, requiring disks for buffering. Furthermore, since a very high transmission rate is used for each video segment, an extremely high bandwidth is required to write data to the disk as quickly as it receives the video. To address these issues, the authors of [AGG96b] proposed a technique called *Permutation-based Pyramid Broadcasting* (PPB).

A.8.3. Permutation-based Pyramid Broadcasting

PPB is similar to PB except that each channel multiplexes its own segments (instead of transmitting them sequentially), and a new stream is started once every short period. This strategy allows PPB to reduce both disk space and I/O bandwidth requirements at the receivers. However, the required disk size is still large due to the exponential nature of the data fragmentation scheme.

The sizes of successive segments increase exponentially, thus causing the size of the last segment to be very large (typically more than 50% of the video). Since the buffer sizes are determined by the largest segment, using the same data fragmentation scheme proposed for Pyramid Broadcasting limits the savings achievable by PPB. In PPB, a client needs to tune in different logical sub-channels to collect its data for a given data fragment if the maximum savings in disk space is desirable.

A.8.4. Skyscraper Broadcasting

To reduce the disk costs in the client side, the authors of [HUA97] introduced *Skyscraper Broadcasting* (SB), which uses a new data fragmentation technique and proposes a different broadcasting strategy. In SB, *K* channels are assigned to each of the *N* most popular objects. Each of these *K* channels transports a specific segment of the video at the playback rate. The progression of relative segments size on the channel is bounded by the width parameter *W*, in order to limit the storage capacity required at the client end. SB allows for simple and efficient implementation, and can achieve low service latency, while using only 20% of the buffer space required by PPB.

A.8.5. Greedy Disk-conserving Broadcasting

The authors of [GAO98] provided a framework for broadcasting schemes, and designed a family of schemes for broadcasting popular videos, called the *Greedy Disk-conserving Broadcasting* (GDB). They systematically analyze the resource requirements, i.e., the number of server broadcast channels, the client storage space, and the client I/O bandwidth required by GDB.

GDB exhibits a tradeoff between any two of the three resources, and outperforms SB in the sense of reducing resource requirements. The *Dynamic Skyscraper Broadcasting* (DSB) in [EAG98] dynamically schedules the objects that are broadcast on the skyscraper channels to provide all clients with a precise time at which their requested objects will be broadcast, or an upper bound on that time if the delay is small and reaps the cost/performance benefits of the skyscraper broadcasting.

A.9. Harmonic Schemes

The above broadcasting schemes generally assume that the client I/O bandwidths are limited to download data from only two channels. If the client can download data from more than two channels, there are methods available that can efficiently reduce the service latency with less broadcasting channels. For example, a broadcasting scheme based on the concept of harmonic series is proposed in [JUH97] [JUH98]; the scheme doesn't require that the bandwidth be assigned to a video equal to a multiple of a channel's bandwidth.

For a movie of length of D minutes, if we want to reduce the viewer's waiting time to $D=N$ minutes, we only need to allocate $H(N)$ video channels to broadcast the movie periodically, where $H(N)$ is the harmonic number of N, i.e., $H(N) = 1+ 1/2 +…+ 1/N$. The *staircase* scheme in [JUH97b] can reduce the storage and disk transfer-rate requirement at the client end. However, both the staircase and harmonic schemes cannot serve bufferless users.

In [JUH97c], a scheme called *Fast Broadcasting* (FB) is proposed, which can further reduce the waiting time and the buffer requirement. Using FB, if a STB does not have any buffer, its user can still view a movie insofar as a longer waiting time is acceptable. The authors of [TSE00] proposed two enhancements to FB, showing how to dynamically change the number of channels assigned to the video and seamlessly perform this transition, and presenting a greedy scheme to assign a set of channels to a set of videos such that the average viewers' waiting time is minimal.

A.10. Asynchronous multicasting

Asynchronous multicasting [WOO96], [KAL96] intended to improve the system efficiency by allowing the user to join a multicast group and store some video data in local buffer for later use. This is achieved by breaking up the video into segments and sending out these segments at a rate greater than the consumption rate of the video.

All practical scheduling policies are guided by three primary objectives: minimize the reneging probability, minimize average waiting time, and be fair. It was shown in [DAN94][DAN96][GAO99] and [GAO99b] that a hybrid of the above two techniques offered the best performance. For example, the *Catching* proposed in [GAO99b] is a combination of periodic broadcast and client-initiated prefix retrieval of popular videos. There are many hybrid schemes to improve the overall performance of multicast VoD. The selective catching in [GAO99b] further improves the overall performance by combining catching and controlled multicast to account for diverse user access patterns. Because most demands are on a few very popular movies, more channels are assigned to popular videos. However, it is necessary (and important) to support unpopular videos. We assume that scheduled multicasts are used to handle less popular videos, while the server-initiated scheme is used for popular videos. In this approach, a fraction of server channels are reserved and pre-allocated for periodically-broadcasting popular videos. The remaining channels are used to serve the rest of the videos using some scheduled multicasts. This hybrid of server initiated and client-initiated schemes achieves better overall performance.

Appendix B
Client Design for Playing Video Data

B.1. Introduction

The real evaluation to the VoD service is done at the client side where, the video data is received and played at the client's displaying device. It is the client who decides the required quality of service (QoS), it is him who decides when to connect to the server and when to disconnect, it is him who chooses what to watch and when, and it is him who evaluates the service and expresses his satisfaction or dissatisfaction of the service. Therefore, we have taken in our considerations the design of the VoD client in order to provide a tool for visualizing the video and to make an evaluation for the VoD server.

For a successful delivery of video data as well as a successful display of the video contents without interruptions, the client has to synchronize well with the server so that the displayed video is not experiencing jittering. This implicates that, if the server supports multiple channels then, the client should be able to receive from multiple channels. Also, if the server is supporting multicasting then, the client should be able to receive from multicast channels.

These facts and others, such as interacting with server and invoking VCR-functions, have required the need for studying and analyzing the client functionality and communication with server so that the design can be adequate for playing video data. For this reason, we have put a set of objectives and functional requirements in order to achieve a successful design and implementation of the client architecture. These objectives and requirements give the user the following abilities:

- Establishing communication between the server and the client
- Selecting the required video file to be viewed.
- Setting the parameters of the quality of service.
- Negotiating with server by sending requests and receiving answers.
- Setting specific configurations and saving them for future use
- Visualizing the video content. The video content can be reproduced from a stream or a local file.
- Playing the video stream without the need to store the video data in the disk before playing it.
- Visualizing more than one video simultaneously.
- Invoking VCR-like controls such as, stop, pause, jump forward, and jump backward.

These objectives and the analysis of the VoD client have shown that the client design can be made based on three separated modules. The first module is the graphical user interface (GUI) where the user can introduce his requests in order to connect to the server, to brows the video files list, to select between multiple screens, and to visualize the video contents. The second module is the communication module which works as a communicating point between the server and the client's modules. The third module is the video-player module which decodes the video data and plays it, and it provides a way for invoking VCR-like functions. The design and the implementation of each one of these modules, and the objective of each one of them are discussed in the following sections.

B.2. Client Architecture

The client architecture is based on a single end-client and multiple playback video and, it consists of three modules which facilitate messages exchange between the server and the client, video reception and video playback. These modules are; the interface module, the communication module and the player module.

Figure B.1 shows the implementation of the architecture. The idea of this architecture is to allow the user from the interface module to control multiple video players.

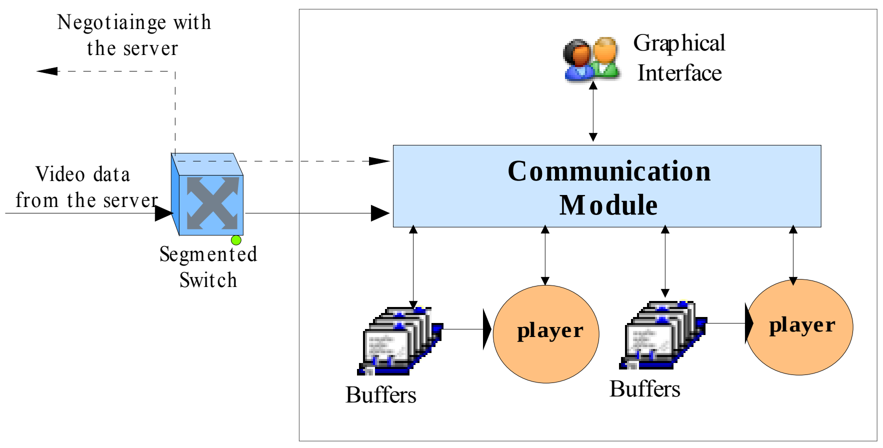


Figure B.1. **Single Client multiple players design**

The figure shows a single interface and multiple video surfaces for playing the video. The interface connects to the server through the communication module and creates a thread for each video in order to visualize it. In its turn, the communication module receives the video data and forwards it the appropriate thread.

B.2.1. Interface module

This module allows the user to connect to the server and to interact as well as negotiate with it by sending requests and receiving answers. The client is designed in a graphical mode so that it facilitates to the user the interaction with the application and the server.

As we said before, this module is functioning independently from the other modules. It generates all the signals and messages which lead to the invocation and displaying the video contents at the displaying device and, it demonstrate the necessary windows where the user can set his configuration and modify them as needed. In addition it allows the user to connect and to start the video reproduction.

B.2.2. Video-Player Module

The video-player module is launched by the interface module when the user requests the visualization of a movie. At the start up, the module configures the video surface which will be used to play the movie according to the graphic card and screen configurations. Once the video data is arrived to the player buffers, the player module call the necessary function in order to it decodes the video data and demonstrates it on the video surface.

As it can be seen from the above figure B.1 and the above discussion, there is a complete separation of functionality between the communication module and the player module. This separation allows interoperability between our client and any other widely used player such as Xine [WPG07].

B.2.3. Communication Module

This module is launched by the interface module when the application starts. It is responsible for all communications carried out between the client and the server as well as between the internal modules. It defines two types of communications:

- Control of messages: it controls messages that could be generated by the users and the messages that could be generated by the operating system as a result of an error. The communication module translates this information into messages that can be understood by the server. In the same way, it controls the answering messages from the server and conveys them to the corresponding module.
- Data transmission: It is responsible for receiving the video data packets and forwards them to the player. Since the client is designed with objective of reproducing multiple video simultaneously, it decides to where the data should be delivered.
- Establishing of communication. This module is able to establish
  - Unicast session.
  - Multicast session.
  - Combination of both at the same time.
  - Communication with more than one client at the same time.

2.2.4. Internal Client/server implementation

The communication between the clients modules are done in a client/server mode. These communications can be unidirectional or bidirectional. For example, the communication between the GUI and the player module is done as a unidirectional while the communication between the GUI and the communication module is done in a bidirectional manner.

The GUI carry out a function requested by the user, the function is conveyed to the player module and then, the player module reproduce and display the requested data at the graphic surface. The graphic surface is shared between the GUI module and the Player module.

The communication module does not initiate any command by itself. It only offers service of communication between the client and server. The client invokes the requests, the requests are conveyed to the server, the server sends the answer in form of messages or video data and then, the communication module forward the received answer to the appropriate module.

Since the application has client/server architecture between its modules, it is necessary to define an internal communication protocol which permits the modules to communicate between each other independently. Also, the internal communication between the modules and the communication with the server are also independent.

It would be easier to have both implementations done with the same protocol. However, this would make the client server dependent. That is, the client can't use other servers which use different protocols.

B.3. Interaction with the Application

At the start up, the interface module reads the network configuration from a file (e.g. the video packet size and the port used to communicate with the video server). Thereafter, it executes the communication module and passes to it the configuration data which will be needed for future connection with the server.

Thereafter, it appears on the screen a window from which the user can click or hit a button in order to connect to the server. Figure B.2 shows a general view of this window which includes three parts:

- Tools bar which is located at the top of the window and, it includes the main controls which are needed to connect to the server and initiate the video session. In addition, it has the volume control which controls the sound level.
- The Video surface which is located in the middle where the video data will be displayed.
- The VCR-Like controls which is located at the bottom of the window. These controls will be disabled until the arrival and the display of the video data.

To understand the functionality of the several controls of the player, figure B.2 shows the tool bar which appears at the top of the player GUI (figure B.3). The figures show clearly a mapping between the tool bar controls and its equivalents in a typical Microsoft window.

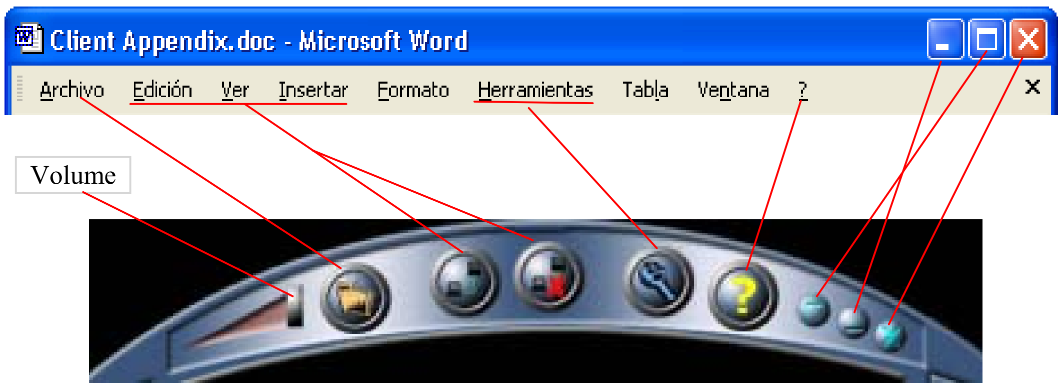


Figure B.2. Functionality of the tool bar controls

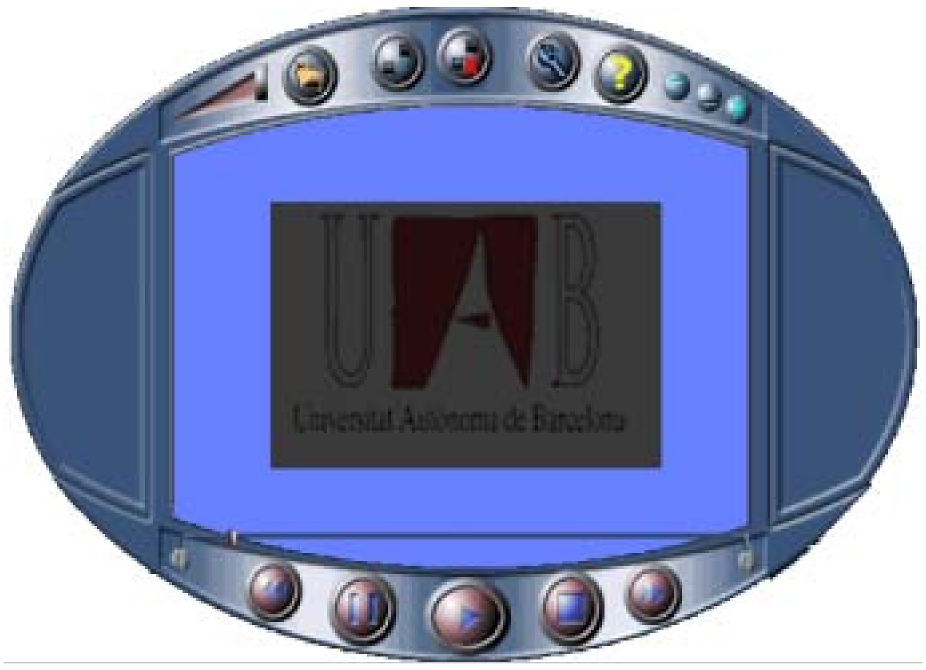


Figure B.3. Client Player Interface

As the tool bar indicates, it has several controls. These controls are (from the left to the right) sound volume, File, Connect, Disconnect, tools, help. In addition, the tool bar has three buttons which are used to maximize the window, to hide the window and to close the application. In general, these controls which appear in figure B.3 will be activated when the application is launched. Thereafter, these controls are used to carry out the available operations available by these controls.

The VCR controls appear at the bottom part of figure B.3, and they will remain disabled until a video is displayed at the video surface. These controls are classical ones and can be found in any player so, they share the same functionality. Therefore, only the tool bar controls are explained. It includes FILE, CONNECT, DESCONNECT, TOOL, and HELP.

FILE: The GUI allows the user to select a local file providing him a dialogue box with directories tree from which a file can be selected (Figure B.4).

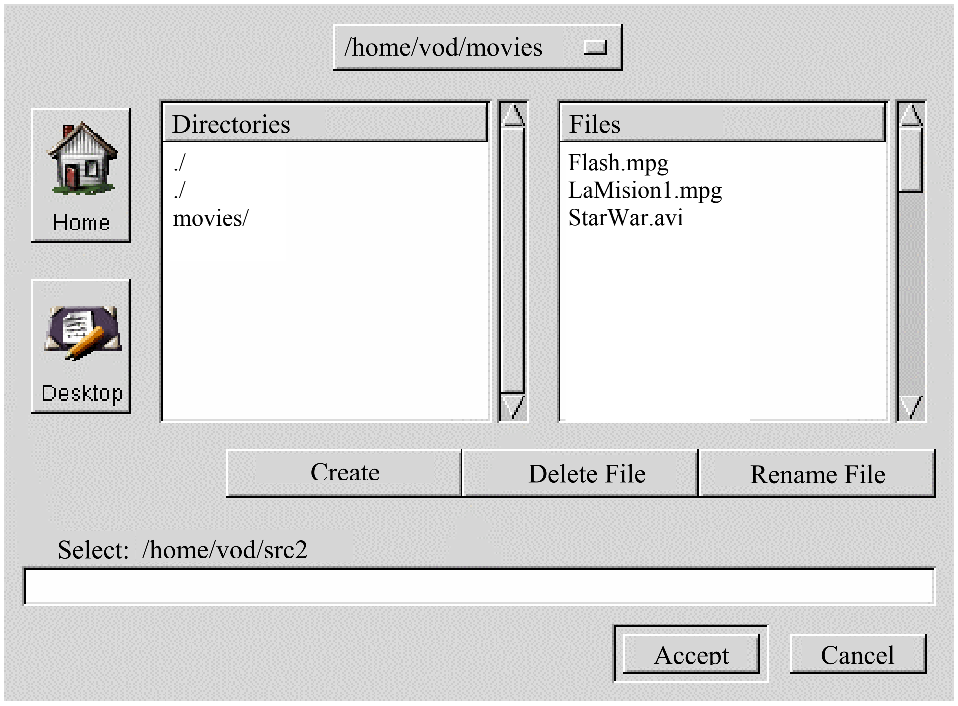
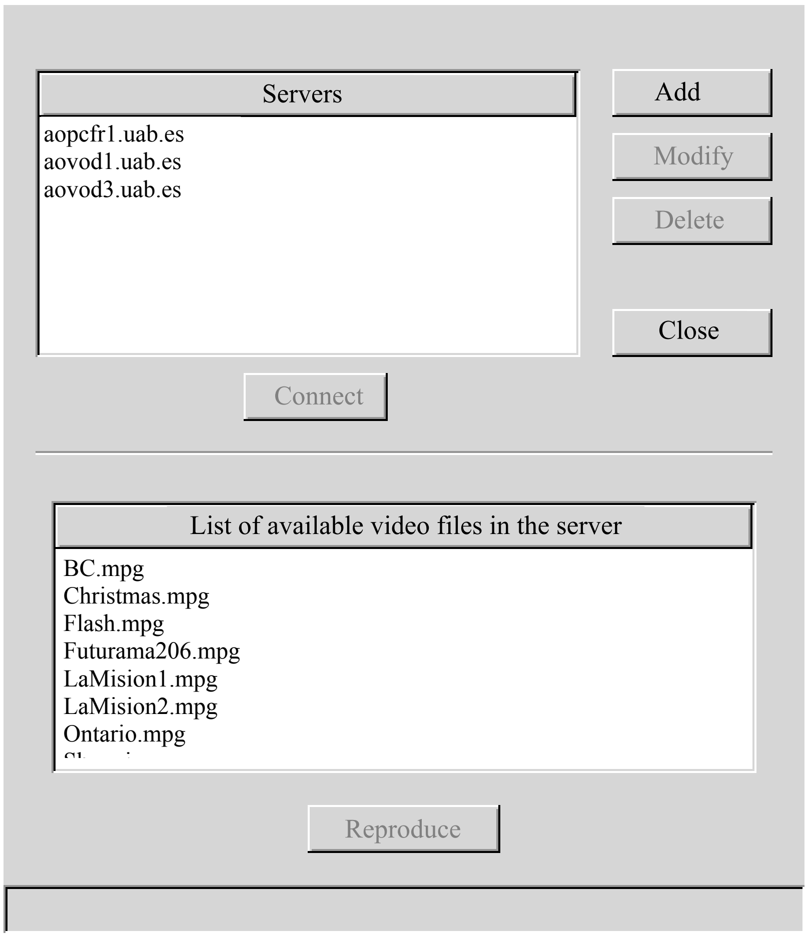


Figure B.4. Browsing local files

CONNECT: To establish a connection, the GUI provides a window from which the connection can be invoked. This window allows the user to select a server from a set of servers, add a new server, modify information of a server and connect (Figure B.5). Once the Connect button is clicked, a list of the available video files appears in the list box.



aopcfr1.uab.es
aovod1.uab.es
aovod3.uab.es

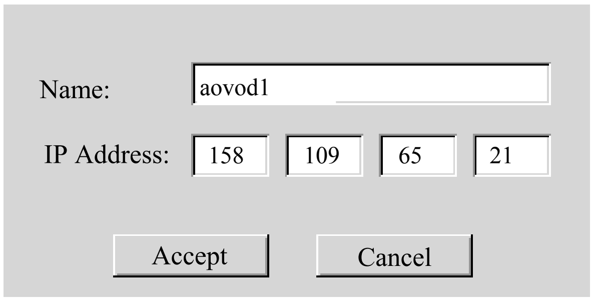
BC.mpg
Christmas.mpg
Flash.mpg
Futurama206.mpg
LaMision1.mpg
LaMision2.mpg
Ontario.mpg

Figure B.5. Start up window

If the user decides to modify a server then, it appears at the top another small window where, the user can change the name of the server and the IP address (figure B.5).

The main services that the GUI is presenting are summarized as follows:

- After the start up of the application, this module offers a window with a set of options. The user can select a predefined server or add a new one. In addition, the user can modify the information of a server and he can also delete a server. In case of changing the server information or adding a new server, the module offers a window for this purpose (Figure B.6).
- After selecting a server, the modify button, the delete button, and the connect button are activated. Thus, by clicking the connect button, the user will be connecting to the server


Name: aovod1

IP Address: 158 109 65 21

Figure B.6. Adding or modifying a server

- It generates events in order to interact with the application. This can be done by hitting a key or clicking a mouse as it is the case in connecting to the server. The user can click on the connect button in order to connect to the server.
- It demonstrates information for the user about the server and the video. This information can be clickable or not clickable. For example, the user can connect to the server and request a list of available video files. The server responds by sending the list along with the necessary information about the video files and the server in general. Upon receiving the answer and the requested information, the video files are displayed in a list box from which the user can click on a desirable one. Other information such as the file size and format are displayed only and can't be accessed.
- It demonstrates also control messages such as errors that might occur during the presentation.

DISCONNECT: The client can disconnect at any time he wants by clicking on the disconnect button.

TOOLS: It shows a window where the user can put or change the configurations such as, the network protocol, the resolution of the video, the language, and other information. Not all options in this window are implemented. They have been left for future work.

HELP: It provides on-line help to the user about the application usage.

B.4. Libraries Used

For the implementation of the client, a set of tools have been used to design and create the GUI and to handle the video functions and the network functions. These libraries include GTK1.2, GTKSkin, and GTKSDL libraries for designing the

GUI, the SDL1.2 and SMPEG0.4.4 libraries for designing the video functions, and the sockets and FIFO for the transmission and inter-process communication.

B.5. More Enhancement

The first version of the player has been enhanced in order to meet the changes which took place in the server after implementing the techniques of patching and prefetching. In addition, the GUI has been modified (figure B.7) so that it can offer other facilities such as switching between single and multiple modes to see multiple video, getting information about the video, the buffers, the network load, and the server status. Further work needs to be done with the new version.

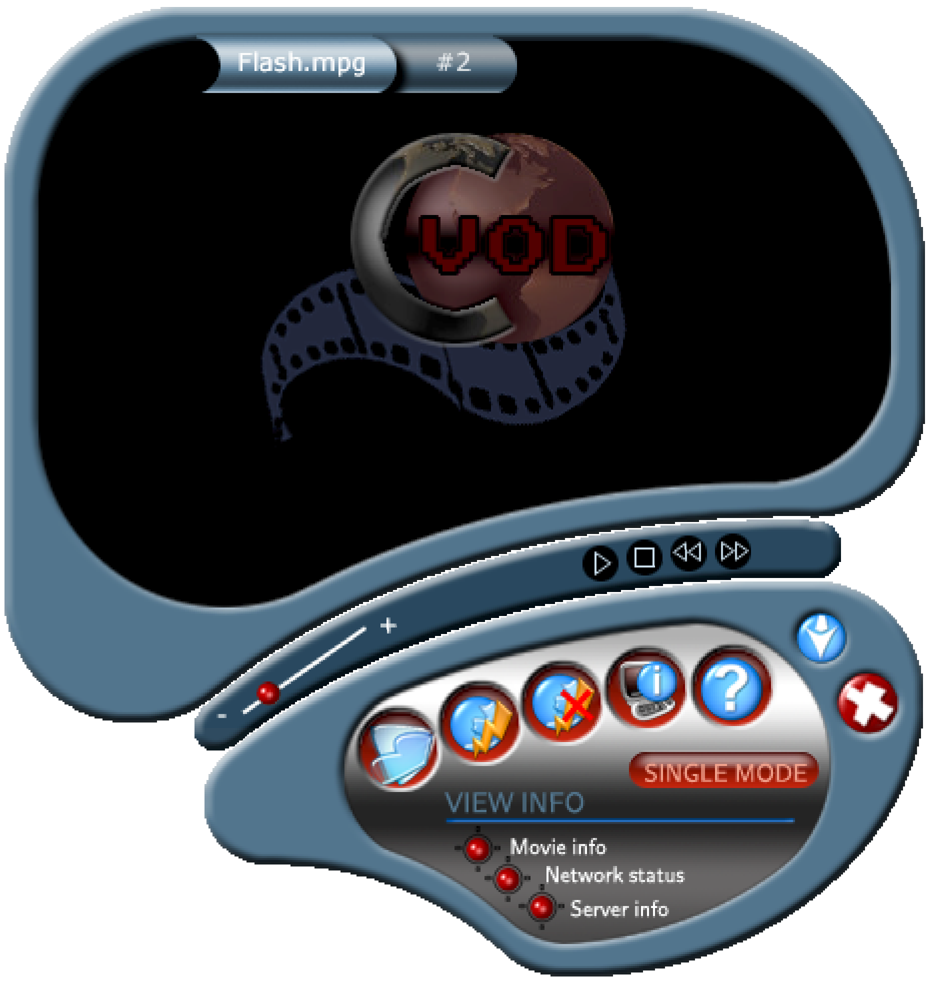


Figure B.7. The new version of the player GUI

References

[ABR97] E. L. Abram-Profeta, K. G. Shin, "*Scheduling video Programs in Near Video-on-Demand Systems*", In proc. Of ACM Multimedia '97, pp. 359-369, Seattle, USA, November 1997.

[AGG96a] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "*On Optimal Piggyback Merging Policies for Video-On-Demand Systems*", ACM SIGMETRICS '96, pp. 200-209, Philadelphia, PA, USA, May 1996.

[AGG96b] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "*A permutation-based pyramid broadcasting scheme for video-on-demands systems*", in proc. Of the IEEE Int'l Conf. on Multimedia Systems '96, pp. 118-126, Hiroshima, Japan, June 1996.

[BAN94] R. O. Banker, J. Huppertz, M. Hayashi, D. Lett, V. Godlewski, and M. Raley, "*Method of providing video-on-demand with VCR-like functions*", *U.S. Patent 5357276*, October 18 1994.

[BAR96] S. A. Barnett, and G. J. Anido, "*A cost comparison of distributed and centralized approaches to video-on-demand*", IEEE Journal on Selected Areas in Communications, vol. 14, pp. 1173-1183, August 1996.

[CAI99] Ying Cai and K.A. Hua, "*An efficient bandwidth-sharing technique for true video on demand systems*," Proc. ACM Multimedia'99, pp.211-214, Orlando, November 1999.

[CAR97] S. W. Carter and D. D. E. Long, "Improving video-on-demand server efficiency through stream tapping," *Proc. IEEE ICCCN'97*, Las Vegas, pp.200-207, September 1997.

[CHA99] S. H. G. Chan and F. Tobagi, "Caching schemes for distributed video services", *in Proc. of IEEE Int. Conference on Communications (ICC'99)*, Canada, pp. 994–1000. June 1999.

[CHA01] S. H. G. Chan and F. Tobagi, "Distributed Servers Architecture for Networked Video Services", in *IEEE/ACM Transactions on Networking*, vol. 9, No. 2, pp. 125-136 April 2001.

[CHO97] J. Cho, H. Shin, "Scheduling video streams in a large-scale video-on-demand server", Parallel Computing 23, 1997.

[CLA92] D. Clark, S. Shenker, L. Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism", In Proceedings of ACM SIGCOMM'92, pp. 14-26, Baltimore, MD, August 1992.

[CLA93] D. Clark, "Oracle Predicts Interactive Gear by Early 1994." The wall Street Journal, pp. B1, B14, November 10, 1993.

[COR02] F. Cores, A. Ripoll, E. Luque, "Double P-Tree: A Distributed Architecture for Large-Scale Video-on-Demand", *Euro-Par 2002*, LNCS 2400, pp. 816-825, August 2002.

[DAN94] A. dan, D. Sitaram, and P. Shahabuddin, "*Scheduling Policies for an On-Demand Video Server with Batching*", proc. of 2nd ACM Multimedia Conf. pp. 15-24, San Francisco, California, USA, October 1994.

[DAN96] A. Dan, D. Sitaram, and P. Shahabuddin, "Dynamic Batching Policies for an On-Demand Video Server," Multimedia systems, vol. 4, pp. 112-121, June 1996.

[DAN96d] A. Dan, Y. Heights, and D. Sitaram, "Generalized interval caching policy for mixed interactive and long video workloads," In *Proc. of SPIE's Conf. on Multimedia Computing and Networking*, pp.344-351, San Jose, CA, January 1996.

[DEN96] J. Dengler, C. Berhardt, E. Biersack, "Deterministic Admission Control Strategies in Video Severs with Variable Bit Rate Streams", *European Workshop IDMS96*, pp.245-264, March 1996.

[EAG98] D. L. Eager and M. K. Vernon, "*Dynamic skyscraper broadcasts for video-on-demand*," in Proc. 4th Int. Workshop on Multimedia Information Systems (MIS'98), pp. 18—32, Istanbul, Turkey, Sept. 1998.

[EAG99] D.L. Eager, M.K. Vernon, and J. Zahorjan, "Optimal and efficient merging schedules for video-on-demand servers," *Proc. ACM multimedia'99*, pp.199-202, Orlando, November 1999.

[FAB01] H. Fabmi, M. Latif, S. Sedigh_Ali, A. Ghafoor, P. Liu, L.H. Hsu, "Proxy servers for scalable interactive video support", *IEEE Computer*, vol. 34 Iss. 9, pp. 54 -60, September 2001.

[FER90] D. Ferrari, D.C. Verma, "A Scheme for Real-Time Channel Establishment in Wide Area Networks", IEEE J. on Selected Areas in Communications, Vol. 8, No. 3, pp. 368-379, April 1990.

[FRE95] C. S. Freedman and D. J. Dewitt, "The SPIFFI scalable video-on-demand system," *Proc. of ACM SIGMOD'95*, pp. 352-363, San Jose, CA, USA, May 1995

[GAO98] L. Gao, J. Kurose, and D. Towsley, "Efficient schemes for broadcasting popular videos," *Proceedings of NOSSDAV'98*, Cambridge, UK, July 1998

[GAO99] L. Gao and D. Towsley, "Supplying instantaneous video-on-demand services using controlled multicast," *Proc. of IEEE ICMCS'99*, pp. 117-121, Florence, Italy, June 1999.

[GAO99b] L. Gao, Z.-L. Zhang and D. Towsley, "Catching and selective catching: efficient latency reduction techniques for delivering continuous multimedia streams," *Proc. ACM multimedia'99*, pp.203-206, Orlando, Nov. 1999.

[GEM95] D.J. Gemmell, H. M. Vin, D. D. Kandlur, P. V. Rangan, L.A. Iwe, Multimedia Storage Servers: A Tutorial, IEEE Computer, pp. 40-49, May 1995.

[GOL95] L. Golubchik, John C.S. Lui and R.R. Muntz, "Reducing I/O Demand in Video-On_Demand Storage Servers," In Proc. of the ACM SIGMETRICS/Performance '95 Conference, pp. 25-36, May 1995.

[GOL96] L. Golubchik, J. C. S. Lui, and R. R. Muntz, "*Adaptive Piggybacking: A Novel Technique for Data Sharing in Video-On-Demand Storage Servers*", ACM Multimedia Syst., vol. 4(3), pp. 140-155, 1996.

[HOL97] S. Hollfelder, "*Admission Control for Multimedia Applications in Client-Pull Architectures*", International Workshop on Multimedia Info. Sys, pp. 25-27, Italy, September 1997.

[HUA02] Huadong Ma, Kang G. Shin, "*Multicast Video-on-Demand Services*", IEEE Computer, Vol. 35 Issue 2, February 2002.

[HUA97] K. A. Hua and S. Sheu, "*Skyscraper Broadcasting: a New Broadcasting Scheme for Metropolitan Video-On-Demand Systems*", Proc. ACM SIGCOMM '97, pp. 89-100, September 1997.

[HUA97b] K.A. Hua, S. Sheu, and J. Z. Wang, "*Earthworm: A network memory management technique for large-scale distributed multimedia applications*," Proc. of IEEE INFOCOM'97,pp. 990-997, Kobe, Japan, April 1997.

[HUA98] K.A. Hua, Y. Cai, and S. Sheu, "*Patching: A multicast technique for true video-on-demand services*," Proc. ACM Multimedia'98, pp.191-200, Bristol, U.K., September 1998.

[HUA99] Ying Cai, K.A. Hua and K. Vu, "*Optimizing patching performance*," Proc. of SPIE's Conference on Multimedia Computing and Networking (MMCN'99), pp.204-216, San Jose, January 1999.

[JEF91] K. Jeffery, D. F. Stanat and C. U. Martel, "*On Non-Preemptive Scheduling of Periodic and Sporadic Tasks*", Proc. Of Real-time Systems Symp., pp. 129-139, December 1991.

[JUH97] L. Juhn and L. Tseng, "Harmonic Broadcasting for Video-On-Demand Service," IEEE Transactions on Broadcasting 43, pp. 268-271, September 1997.

[JUH97b] L.-S. Juhn and L.-M. Tseng, "Staircase data broadcasting and receiving scheme for hot video service," *IEEE Transactions on Consumer Electronics*, 43(4):1110-1117, November1997.

[JUH97c] L.-S. Juhn and L.-M. Tseng, "Fast broadcasting for hot video access," *Proc. RTCSA'97*, pp.237-243, Oct.1997.

[JUH98] L.-S. Juhn and L.-M. Tseng, "Enhanced harmonic data broadcasting and receiving scheme for popular video service," *IEEE Transactions on Consumer Electronics*, 44(2):343-346, May 1998.

[KAL96] H. Kalva and B. Furht, "*Techniques for improving the capacity of video-on-demand systems*" in proc. 29th Annu. Hawaii Int. Conf. Systems Science, pp. 308-315, January 1996.

[KEE95] K. Keeton and R. H. Katz, "*Evaluating video layout strategies for a high-performance storage server*," Multimedia Systems, 3(2), pp. 43-52, May 1995.

[LI96] V. O. K. Li, W. Liao, X. Qui, and E. W. M. Wong, "*Performance Model of Interactive Video-On-Demand Systems*", IEEE J. Select. Areas Commun., vol. 14, no. 6, pp. 1099-1109, August 1996.

[LIT93] Thomas D.C. Little and D. Venkatesh, "*Probabilistic assignment of movies to storage devices in a video-on-demand*", Proc. Of International Workshop on Network and OS Support for Digital Audio and Video (NOSSDAV'93), pp. 213-224, Lancaster, UK, 1993.

[LIT94] Thomas D.C. Little and D. Venkatesh, "*Prospects for interactive video-on-demand*", IEEE Multimedia, Fall 1994.

[LIU73] C. L. Liu and J. W. Layland, "*Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment*", *J. Assoc. Comput. Mach*., pp. 46-61, 1973.

[MUN99] Mundur, P., Simon, R., Sood, A. Integrated admission control in hierarchical video-on-demand systems. IEEE Multimedia Systems. ICMCS '99. IEEE CS Press., pp. 220-225, 1999.

[NAH95] K. Nahrstedt, J. M. Smith, "The QoS Broker", IEEE Multimedia Magazine, Spring 1995.

[NAH98] K. Nahrstedt, H.-H. Chu, Srinivas Narayan, "QoS- Aware Resource Management for Distributed Multimedia Applications", *Journal on High-Speed Networking*, IOS Press, Vol. 8 , No.3-4, 1998, pp.227-255.

[NAH99] K. Nahrstedt "Quality of Service in Networked Multimedia Systems", Handbook of Multimedia Computing, CRC Press, pp.839-873, 1999.

[NER97] GT. Nerjes, P. Muth and G. Weikum, "Stochastic Performance Guarantees for Mixed Workloads in Multimedia Information Systems", *Proc. of the IEEE Int. WS on Research Issues in Data Eng*. (RIDE´97), 1997.

[OYA95] Y. Oyang *et al.*, "Design of multimedia storage systems for on-demand playback," *Proc. of Int'l Conf. on Data Engineering*, pp.457-465, Taipei, March 1995.

[OZD94] B. Ozden *et al.*, "A low-cost storage server for movie on demand databases," *Proc. of the 20th Int'l Conf. on VLDB*, pp. 594-605, Santiago, Chile, September 1994

[OZD95] B. Ozden *et al.*,"Demand paging for video-on-demand servers," *Proc. of IEEE ICMCS'95*, pp.264-272, Washington DC, May 1995

[PAD99] Padmavathi Mundur, Robert Simon, Arun Sood Integrated System Architecture for Video-on-Demand Applications, Internal Report, CS dept. George Mason University, 1999.

[PAR98] J. F. Pâris, S. W Carter, and D. D. E. Long, "Efficient Broadcasting Protocols for Video On Demand," in Proceedings of the 6th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '98), pp. 127-132, July 1998.

[PAR99] J. F. Pâris, S. W Carter, and D. E. Long, "A Hybrid Broadcasting Protocol for Video on Demand," In Proceedings of the 1999 Multimedia Computer and Networking Conference, pp. 317-326, San Jose, CA, January 1999.

[PER01] Perimele Consulting, "Digital Rights Management for Large Scale Video-on-Demand Networks", Oxford, London, August 2001.

[QAZ03a] B. Qazzaz, R. Suppi, F. Cores, A. Ripoll, X. Yang, P. Hernandez, E. Luque. "Admission Control Policies For Video On Demand Brokers", IEEE ICME Multimedia & Expo Proceedings, pp. 529-532, Baltimore-Myryland, USA, July 2003.

[QAZ03b] B. Qazzaz, R. Suppi, F. Cores, A. Ripoll, P. Hernandez, E. Luque, "Providing Interactive Video on Demand Services in Distributed Architecture", Proceedings of 29th EUROMICRO Conference, pp. 215-222, Belek-Antalya, Turkey, September 2003.

[RED99] A. L. N. Reddy, "*Real-Time Communication Scheduling in a Multicomputer Video Server*", Journal of Parallel and Distributed Computing 58, 425-445, 1999.

[RED95] A. L. Reddy, "*Scheduling and Data Ditribution in a Multiprocessor Video Server*", in Proc. Of International Conference on Multimedia Computing and Systems, pp. 256-263, Washington, DC, 1995.

[REI98] Martin Reisslein, Keith W. Ross, "*High-Performance Prefetching Protocols for VBR Prerecorded Video*", IEEE Network, Vol. 12, no. 6, pp. 46-55, November/December 1998.

[REI00] M. Reisslein, F. Hartanto, and K. W. Ross, "*Interactive video streaming with proxy servers*", Proc. of First International Workshop on Intelligent Multimedia Computing and Networking (IMMCN), February 2000

[ROT97] V. Rottmann, P. Berenbrink, and R. Lüling, "*A Simple Distributed Scheduling Policy for Parallel Interactive Continuous Media Servers*", Parallel Computing 23, pp. 1757-1776, 1997.

[SEN99] S. Sen *et al.*, "Optimal patching schemes for efficient multimedia streaming," *Proc. IEEE NOSSDAV'99*, Basking Ridge, NJ, June 1999.

[SHA97] H. Shachnai and P. S. Yu, "*Exploring Waiting Tolerance in Effective Batching for Video-On-Demand Scheduling*", in proc. 8th Conf. Computer Systems and Software Engineering, pp. 67-76, June 1997.

[SHE95] P.J. Shenoy, P. Goyal and H.M. Vin, "*Issues in Multimedia Server Design*", ACM Computing Surveys, vol. 27, no. 4, pp. 636-639, December 1995.

[SHE02] P. Shenoy, S. Hasan, P. Kulkarni, K. Ramamritham, "*Middleware versus Native OS Support: Architectural Considerations for Supporting Multimedia Applications*", Proceedings of Real-Time Applications and Systems (RTAS), 2002.

[SHEU97] S. Sheu, K.A. Hua and W. Tavanapong, "*Chaining: a generalized batching technique for video-on-demand systems*" *Proc. IEEE ICMCS'97*, pp.110-117, Ottawa, 1997.

[STE95] Ralf Steinmetz, Klara Nahrstedt, 1995, "Multimedia: Computing Communication and Application", New Jersey, Prentice Hall PTR.

[SUP01] R. Suppi, B. Qazzaz, E. Luque. "*Design and Development of a Multithreaded Video on Demand Server for LAN*", SCI/ISAS 5th

World Multi-conference on Systemics Cybernetics and Informatics, Vol. XII. pp. 380-388, Orlando-Florida, USA, July 2001

[TAN03] Andrew S. Tanenbaum, Computer Networks, Fourth edition 2003.

[TSE00] Y.-C. Tseng *et al.*, "*Data broadcasting and seamless channel transition for highly-demanded videos*" ProcIEEE INFOCOM'2000, pp.727-736, Tel-Alrabee, Palestine, March 2000.

[VEN96] Chitra Venkatramani, Design, Implementation and Evaluation of ETHER: A Real-Time Ethernet Protocol", Ph.D. Dissertation 1996.

[VEN97] N. Venkatasubramanian, K. Nahrstedt, "An Integrated Metric for Video QoS", *in proceeding of ACM Multimedia '97*, pp. 371-381, Seattle, USA, November 1997.

[VEN98] V. Venkataramani, "A Reservation Protocol for Multimedia Management System", 1998. http://cairo.cs.uiuc.edu

[TEW96] R. Tewari, R. Mukherjee, D. M. Vin, "Design and performance tradeoffs in clustered video servers", Proc. International Conference on Multimedia Computing and Systems, pp. 144-150, Los Alamitos, CA, 1996.

[VALL02] J. Valls, "Visualizador de MPEG para Sistema de Video bajo Demanda", Final Project, Escola Universitaria d'Informàtica, Sabadell, Spain, Septiembre 2002.

[VIN94] H. M. Vin, A. Goyal, A. Goyal, P. Goyal,"An Observation-Based Admission Algorithm for Multimedia Servers", *Proc. The First IEEE Int. Conf. on Multimedia Computing and systems (ICMCS´94)*, pp.234-243, Boston, 1994.

[VIS95] S. Viswanathan and T. Imielinski, "*Pyramid broadcasting for video on demand service*" *Proc. the SPIE Multimedia Computing and Networking Conference*, 2417, pp. 66-77, San Jose, CA, 1995

[VIS96] S. Viswanathan and T. Imielinski, "*Metropolitan area video-on-demand service using pyramid broadcasting*", ACM Multimedia Syst., vol. 4, no. 4, pp. 197-208, 1996.

[WAN00] B. Wang and J. C. Hou, "*Multicast routing and its QoS routing: problems, algorithms, and protocols*" *IEEE Network*, pp.22-36, January/February 2000

[WOO96] H. Woo and C. K. Kim, “*Multicast scheduling for VoD services*”, Multimedia Tools Application, vol. 2, no. 2, pp. 157-171, 1996.

[YAN03] X. Y. Yang, “Sistema de almacenamiento dedicado a video bajo demanda”, undergraduate Final Project, Departamento de Informática, CAOS, Universidad Autónoma de Barcelona, 2003.

Web PaGes

[WPG01] http://www.usvo.com/history/instatmdr.html

[WPG02] http://www.itvdictionary.com/vod.html

[WPG03] http://www.ipmonitor.com/index.asp

[WPG04] Http://www.cs.columbia.edu/~hgs/rtsp/proposed.txt

[WPG05] http://www.isi.edu/div7/rsvp/rsvp.html

[WPG06] http://bmrc.berkeley.edu

[WPG07] Http://www.xinehq.de

Abbreviations

AAP	Adaptive Average Policy
AC	Admission Control
ACM	Admission Control Module
AMP	Adaptive Maximum Policy
AP	Average Policy
BW	Bandwidth
CB_MDA	Credit Based Media Delivery Algorithm
CCV	Consumed Credits Value
CIWP	Client Initiated With Prefetching
CPU	Control Processing Unit
CV	Credit Value
DSB	Dynamic Skyscraper Broadcasting
DSL	Digital Subscriber Line
DVoD	Distributed Video on Demand
ES	Extra Streams
FB	Fast Broadcasting
FCFS	First Come First Served
FOREMMS	Forest Management and Monitoring System
FPS	Frames Per Second
GB	GigaByte
GDB	Greedy Disk-conserving Broadcasting
GTK	Gimp ToolKit
GTKSDL	Gimp ToolKit Simple DirectMedia Layer
GTKSkin	Gimp ToolKit Skin
GUI	Graphical User Interface
HDTV	High Definition Television

ICV	Initial Credit Value
IP	Internet Protocol
ISO	International Standard Organization
IVoD	Interactive Video on Demand
L	scheduling sLot.
LAN	Local Area Network
LSS	Local Storage Server
MB/s	MegaByte per second
Mb/s	MegaBit per second
MBuffer	Multicast Buffer
MP	Maximum Policy
MPEG	Motion Picture Experts Group
MTU	Maximum Transfer Unit
No-VoD	No video on Demand (Broadcast)
NTSC	National Television Standard Committee
NVoD	Near Video on Demand
OS	Operating Systems
PB	Pyramid Broadcasting
PC	Personal Computer
PPB	Permutation-based Pyramid Broadcasting
PPV	Pay-Per-View
PRV	Prefetched Reserve Value
QoS	Quality of Service
Q-VoD	Quasi Video on Demand
RMM	Resource Manager
RMM	Resource Manager Module
RSVP	ReSerVation Protocol

RTCP	Real Time Control Protocol
RTP	Real Time Protocol
RTSP	Real Time Session Protocol
RTT	Round Trip Time
S	video Stream
SB	Skyscraper Broadcasting
SDL	Simple DirectMedia Layer
SiET	Expected finishing Time of stream Si
SiF	maximum number of Frames in Si
SiP	Playing time length of stream Si
SMPEG	Simple direct media Motion Picture Experts Group
TCP/IP	Transmission Control Protocol/Internet Protocol
TCV	Total Credits Value
TVoD	True Video on Demand
UBuffer	Unicast Buffer
UDP/IP	User Datagram Protocol/Internet Protocol
VBR	Variable Bit Rate
VCR	Video Cassette Recorder
VHS	Video Home System
VoD	Video on Demand
VPS	Video Proxy Server