Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

A DISTRIBUTED NETWORK ARCHITECTURE FOR VIDEO - ON - DEMAND

by
Ge Dai

The objective of this thesis is to design a distributed network architecture that provides video - on - demand services to public subscribers. This architecture is proposed as an alternative to a centralized video service system. The latter system is currently being developed by Oracle Corporation and NCube Corporation.

A simulator is developed to compare the performance of both the distributed and centralized video server architectures. Moreover, an estimate of the cost of both systems is derived using current price data.

It is shown that the distributed video server architecture offers a better cost / performance trade-off than the centralized system. In addition, the distributed system can be scaled up in an incremental fashion to increase the system capacity and throughput.

Finally, the distributed system is a more robust system: in the presence of component failure, it can be configured to isolate or bypass failed components. Thus, it allows for graceful performance degradation, which is difficult to achieve in a centralized system.

A DISTRIBUTED NETWORK ARCHITECTURE FOR VIDEO - ON - DEMAND

by
Ge Dai

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering

Department of Electrical and Computer Engineering

October 1994

Blank Page

APPROVAL PAGE

A DISTRIBUTED NETWORK ARCHITECTURE FOR VIDEO - ON - DEMAND

Ge Dai

[illegible]

Dr. Michael Palis, Thesis Advisor — Date
Associate Professor of Electrical and Computer Engineering
New Jersey Institute of Technology

[illegible]

Dr. Anthony Robbi, Committee Member — Date
Associate Professor of Electrical and Computer Engineering
New Jersey Institute of Technology

[illegible]

Dr. Kenneth S. Sohn, Committee Member — Date
Professor, Acting Chairman of Electrical and Computer Engineering
New Jersey Institute of Technology

BIOGRAPHICAL SKETCH

Author: Ge Dai

Degree: Master of Science in Electrical Engineering

Date: October 1994

Undergraduate and Graduate Education:

- Master of Science in Electrical Engineering,
  New Jersey Institute of Technology
  Newark, New Jersey, USA, 1994

- Bachelor of Science in Electrical Engineering,
  Shanghai University of Science and Technology,
  Shanghai, P. R. China, 1992

Major: Electrical Engineering

To my parents who
devote their lives
to a better China

ACKNOWLEDGMENT

The author would like to express his sincere gratitude to his advisor, Professor Michael Palis, for his guidance, support, kindness, encouragement and friendship throughout the process of producing this thesis.

Thanks to Professor Sohn, Professor Robbi and Professor Manikopoulos for serving as members of the thesis committee.

Thanks to all my friends, who have been a source of love and indispensable support.

TABLE OF CONTENTS

Chapter **Page**

1 INTRODUCTION 1

1.1 Telephony 1

1.2 Video Server 3

1.3 Thesis Objectives 4

1.4 Thesis Contribution 4

1.5 Outline of the Rest of the Thesis 5

2 ANALYSIS AND DESIGN OF THE USER SITE 6

2,1 Overview 6

2.2 Video Functions 6

2.2.1 The VCR 6

2.2.2 Introduction to the MPEG Video Standard 7

2.3 System Design at The User Site 9

2.3.1 Brief Description 9

2.3.2 Design of Video Terminal 10

2.4 Reservation Policy, Service Points and Sessions 14

2.5 Workload Model 15

3 A CENTRALIZED VIDEO SERVER BASED ON A HYPERCUBE PARALLEL COMPUTER 18

3.1 Structure of a Hypercube Parallel Computer 18

3.2 Design of a Hypercube Based Video Server 19

3.3 Video Stream Allocation on the Hypercube 20

3.4 File Server Architecture 25

TABLE OF CONTENTS
(Continued)

Chapter **Page**

4 DESIGN OF DISTRIBUTED VIDEO SERVER 27

4.1 Study of Distributed Architecture 27

4.2 Network Architecture 28

4.3 Node Architecture 30

4.3.1 Assistant Node 31

4.3.2 Front Node 37

4.4 Performance Evaluation -- Static Scheduling 41

5 CONCLUSIONS 49

5.1 Performance Comparison 49

5.2 Cost Estimation 50

5.2.1 Distributed System 50

5.2.2 Centralized System 51

5.2.3 Other Distributed System 51

5.3 Conclusion 52

5.4 Recommendations for Further Study 52

APPENDIX A - Source Code List for Simulator on the Centralized System 54

APPENDIX B - Source Code List for Simulator on the Distributed System. 60

REFERENCES 73

LIST OF TABLES

Table **Page**

2.1 Video Functions 7

4.1 Distribution Table of Peak Time Sessions 45

4.2 Table of Mapping 47

LIST OF FIGURES

Figure	Page
1.1 Bandwidth of a Conventional Telephone Line	1
1.2 Model of Local Telephone Network	2
2.1 Video Terminal System at Customer's Site	9
2.2 System Diagram of Video Terminal System	13
2.3 Basic Poisson Distribution Function	16
2.4 Joint Poisson Distribution Function	16
3.1 Structure of HyperCube Parallel Computer.	18
3.2 Simplest Model of Video Server System.	19
3.3 Basic Model of Hypercube System	19
3.4 Data Link Topology of Hypercube System	20
3.5 Data Stream on the File Server -- Hypercube Unit	21
3.6 Performance of 4-PE Hypercube Unit using Round-Robin Allocation	23
3.7 Performance of 4-PE Hypercube Unit using Addressed Allocation	24
3.8 Overall Performance of a Hypercube Unit.	25
3.9 Structure of File Server	26
4.1 An Architecture of Distributed System	28
4.2 Video Server Network Architecture	28
4.3 One Piece of the Distributed Network	29
4.4 Node Architecture	30
4.5 Data Flow Through the System	32
4.6 Relationship of Duplication and Sessions for an Assistant Node.	33
4.7 Time Slices for Hard Drive Access	36
4.8 Relationship of Duplication and Sessions for a Front Node	37

LIST OF FIGURES
(Continued)

Figure **Page**

4.9 Data Access from a CD-ROM 38

4.10 Video Storage on Each Node 42

4.11 Reservation Table of the 5 Videos 43

4.12 Performance of the Distributed System 48

CHAPTER 1

INTRODUCTION

1.1 Telephony

The telephone has withstood the test of time since Bell invented it and helped usher in the telecommunication era. But till now the concept of the telephone is just the same as what we had before: talking with each other through a pair of telephone lines. Now we have some modern equipment, such as cellular phones, wireless telephones, and even digital technology such as FAX, modem, etc.

The basic function of the telephone service is to transmit 3 KHz voice signals through a telephone line. Scientists have sought ways to extend this bandwidth. They found that besides the lower bandwidth, a conventional telephone line also has a middle bandwidth of about 6 MHz (see Figure 1.1).

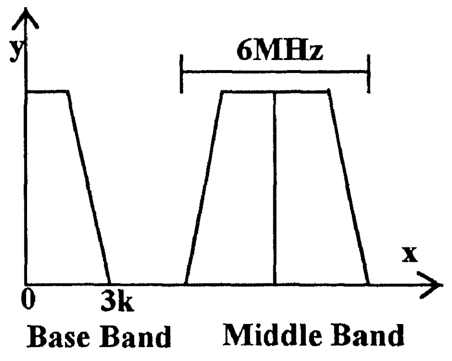


Figure 1.1 Bandwidth of a Conventional Telephone Line

Within this bandwidth, a 6MHz video signal can be transmitted. With advanced data compression methods one can do much more: transmit Full Motion Video within 1.5 Mbps using MPEG I standard or 3 ~ 10 MHz on HDTV using MPEG II. In addition, modern modulation technology now allows about 4 VHS signals to be transmitted on a conventional video channel.

If we could truly use telephone lines to transmit video signals, what would happen to our world? Without changing the current lines, we will have our telephone, video, and other remote service available through a telephone line! We will have interactive classes at home instead of at school! We will open a new page in history.

There is little difference between a "telephone line video" and a "cable TV": In cable TV, a cable company provides a cable and a list of programs with a time schedule for you to choose. You have to follow the schedule instead of making your own schedule. You can not perform real time video functions, such as Fast Forward, Rewind, Pause, etc. But with telephone line video service, you can choose your own favorite movies, make up your own schedule, do video functions, and even communicate with others, such as to your professor, in real time. One of the major advantages of telephone line video compared with cable service is its lower cost, which is very important to both companies and customers.

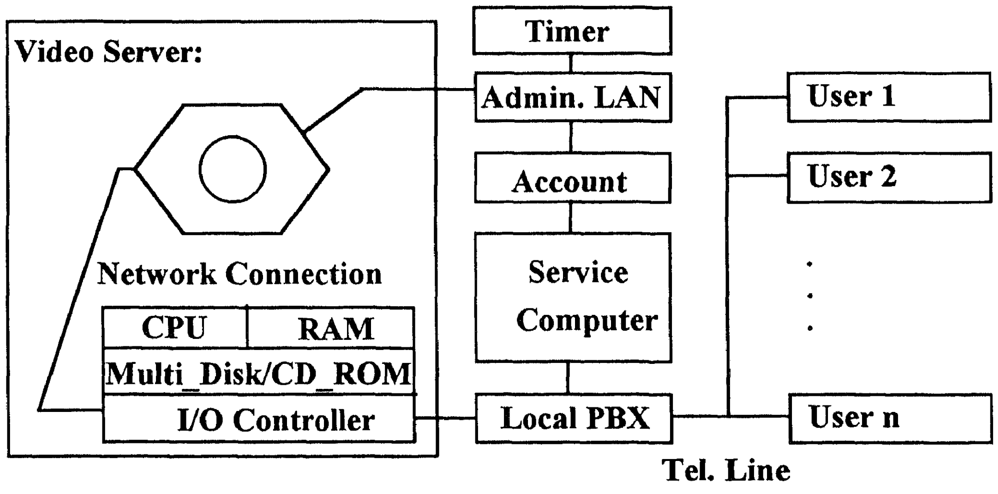


Figure 1.2 Model of local telephone network

1.2 Video Server Architecture

In order to efficiently use the bandwidth available from the telephone line, a "Video Server" seems necessary. Because of the distributed architecture of the telephone network, it is logical to set up the sever locally with a local telephone exchange. The architecture of a local telephone exchange center is shown in Figure 1.2.

A Service Computer is used as a supervisor to manage the resource of the Video Server and balance the network load. It maintain a database of customer accounts, each containing such information as User ID, reservation requests and billing information. The Service Computer receives incoming video requests, accesses user accounts, sets up video sessions, and issue commands to the Video Server.

The Video Server provides the main video service. It maintains a mass storage sub-system that holds several hundreds of full-motion video streams representing movies, sports and musical courts, documentaries, etc. In response to commands issued by the Service Computer, it delivers the requested video streams to the telephone network for transmission to the customer sites.

The architecture of the Video Server can either be centralized or distributed. In a centralized system, the mass storage sub-system (or file server) is controlled by a central computer. Because full-motion video consumes enormous bandwidth, high-speed links (e.g. FDDI) are needed to connect the file server to the central computer. Furthermore, because handreds of video streams have to be processed simultaneously in real-time, the central computer should consist of multiple processes to partition the workload into manageable pieces. Recently NCube Corporation and Oracle Corporation announced a joint venture to develop and market a centralized system based on the NCube massively parallel machine that will provide video-on-demand and other interactive services[8].

Alternatively, the Video Server can be designed as a distributed system. In this design, the file server and processors are distributed across a network of nodes. Each node consists of a standard microprocessor-based PC or workstation and a local storage sub-system for holding a portion of the videos that are supplied by the system. Because both storage and processing are distributed and because processing of different video streams requires little interaction between the nodes, the bandwidth requirements of the network links are significantly less than that of the file server - central computer link in a centralized system. On the other hand, the distributed nature of the system poses the important problem of network load balancing, i.e., how to appropriately allocate the video streams to the individual nodes such that the workloads of the nodes are more or less equal.

1.3 Thesis Objectives

The main objective of this thesis is to compare the centralized and distributed approaches to Video Server design and to determine whether or not the distributed approach is a viable alternative from a cost / performance viewpoint. To achieve these objectives, a detailed study of the user model, workload parameters, and the system requirements of full-motion video are needed. Finally, a simulation of the two systems must be carried out in order to accurately predict their performance.

1.4 Thesis Contribution

- A detailed analysis of the customer, including viewing characteristics, types and frequency of use of video function (e.g., PLAY, STOP, PAUSE, etc.), video request frequencies and preferred viewing times.
- Design a Video Terminal
- Design a Centralized Video Server based on a Hypercube parallel machine
- Design a Distributed Network Architecture for Video-On-Demand
- Simulation of both Centralized and Distributed System based on a workload model.
- Cost Estimation of both systems.

The thesis demonstrates the distributed video server architecture offers a better cost / performance trade-off than the centralized system. In addition, the distributed system can be scaled up in an incremental fashion to increase the system capacity and throughput.

1.5 Outline of the Rest of the Thesis

The rest of the thesis is organized as follows: Chapter 2 is an analysis of the user model. We put the same model into two systems so that we can compare the result. Chapter 3 and Chapter 4 will give us a model of a centralized system and a distributed system respectively, including hardware design. Finally, Chapter 5 will compare the results and make the conclusion.

CHAPTER 2

ANALYSIS AND DESIGN OF THE USER SITE

2.1 Overview

In designing the system, we should always keep in mind that our ultimate goal is to provide the best service to customers.

Consider how a customer rents videos from a local video rental shop. The procedure begins with the customer coming into the shop. He or she looks through the topics listed in the shop, chooses one or several of his or her favorite videos, borrows them and then brings them to home. At some proper time, the customer views the videos. The procedure ends with the customer returning the videos and paying for them in full.

At home, the customer views the rented videos on a VCR. In order to determine the kind of services a customer would require while viewing the videos, a study of the VCR is therefore necessary.

In the next subsection, we briefly discuss the VCR and the MPEG video standard to determine the video functions that must be provided to the user, as well as their system requirement. In subsection 2.3, we use this information to design a "Video Terminal" which acts as an interface between the user and the remote Video Server.

2.2 Video Functions

2.2.1 The VCR

To watch a video, one needs to do some basic operations, namely the standard functions provided by an ordinary VCR (without recording). These functions are shown in Table 2.1 below:

Table 2.1 Video Functions

1.	PLAY	Play the video in normal speed.
2.	STOP	Stop the video.
3.	PAUSE	Pause the video, the screen will display the last video frame.
4.	F.FWD / F.REW	Fast Forward / Rewind: x times faster than normal speed. Screen is black.
5.	FWD / REV	Forward / Review the video: y times faster than normal speed. Screen displays.
6.	P.FWD / P.REV	Picture Forward / Review: go forward / backward a frame.

Notice that this description is limited to the video operations only; most of the commercial VCR are also equipped with a TV receiver, but that is not an issue here.

2.2.2 Introduction to the MPEG Video Standard

MPEG stands for Motion Picture Experts Group which is responsible for audio, video, and combined synchronization of information in digital image sequence coding.[1] There are two MPEG standards. The standard of MPEG I defines a bit stream for compressed video and audio optimized to fit into a bandwidth of 1.5 Mbps, and MPEG II, defines a bit stream for video and audio coded at around 3 to 10 Mbps.

The basic scheme of MPEG is to predict motion from frame to frame in the temporal direction, and then to use DCT's (discrete cosine transforms) to organize the redundancy in the spatial directions. The DCT's are done on 8×8 blocks, and the motion prediction is done in the luminance channel on 16×16 blocks. In other words, given the 16×16 block in the current frame that you are trying to code, you look for a close match to that block in a previous or future frame There are backward prediction modes where later frames are sent first to allow interpolation between frames. The DCT coefficients are "quantized", which means that we divide them by some value to truncate less significant bits. Ideally, many of the coefficients will then end up being zero. The quantization can change for every "macro block". The DCT coefficients, motion vectors, and quantization parameters are then Huffman coded using fixed tables. The DCT coefficients have a special Huffman table that is "two-dimensional" in that one code specifies a run-length of zeros and the non-zero value that ended the run. Also, the motion vectors and the DC DCT components are DPCM coded.

There are three types of coded frames, "I", "P" and "B". "I" frame, or intra frame, is simply frame coded as a still image, not using any past history. "P" frame, or predicted frame, is predicted from the most recently reconstructed I or P frame. Each macro block in a P frame can either come with vector and difference DCT coefficients for a close match in the last I or P, or it can just be "intra" coded in the case that there was no good match. Lastly, "B" frame, or bi-directional frame, is predicted from the closest two I or P frames, one in the past and one in the future. You search for matching blocks in those frames, and to see which works best, try three different things: using the forward vector, the backward vector, and you try averaging the two blocks from the future and past frames, and subtracting that from the block being coded. If none of those work well, you can intra code the block. The sequence of decoded frames usually goes like: IBBPBBPB BPBBIBBPBBPB .... where there are 12 frames from I to I. This is based on a random access requirement that you need a starting point at least once every 0.4 second or so. When you decode, you have to first decode the I, then decode the P, keep both of them in memory, and then decode the two B's in between. You display the I while decoding the P, display the B's as they are being decoded, and then display the P as you're decoding the next P, and so on.

The centralized and distributed Video Servers we describe in this thesis are designed to handle MPEG I, which is about 352 × 240 × 20 bit color.

2.3 System Design at the User Site

2.3.1 Brief Description

As we have introduced before, MPEG streams are compressed in order to save bandwidth. Therefore, at the customer site, there should be a Video Terminal to perform decoding. To deal with a MPEG stream involves digital signal processing, where a digital processor is necessary. There are two kinds of equipment that can be used to decode a MPEG stream, one is a special designed TV set, and the other is a personal computer with some additional peripherals. Since personal computers are already widely used, it is reasonable to foresee that in the future a video terminal will most likely be embedded in a personal computer. Now, Intel chips can reach as fast as 112 MIPS[2] with no difficulty, an achievement which makes it possible to do video processing (not decoding, but management) in real time. The diagram in Figure 2.1 shows such a system at a Video Terminal system at a customer's site.

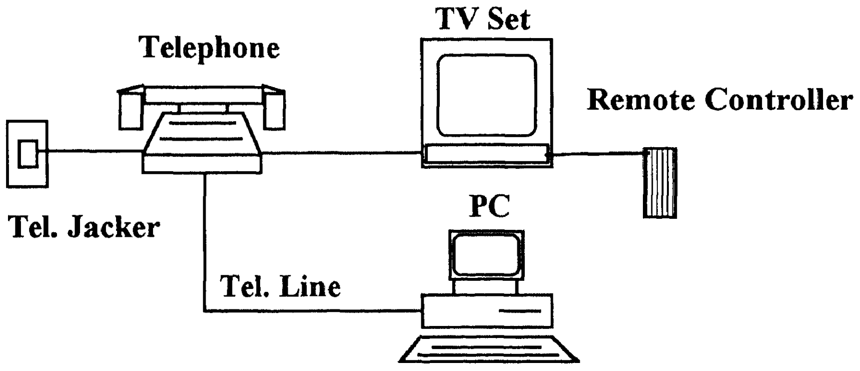


Figure 2.1 Video Terminal System at Customer's Site

The system is divided into three parts. First, there is the communication part, an interface between the telephone line and the digital computer. The simplest form could be any modem, which works at 1.5 Mbps on MPEG I or 4 Mbps on MPEG II. Second, there is the decoding part, a digital processor with some amount of memory. Third, there is a decoder co-processor and a video display device, which should have enough bandwidth to hold 30 frames/sec data stream.

2.3.2. Design of Video Terminal

We now discuss the design of the Video Terminal. Although the Video Server handle only MPEG I video streams, we will design the Video Terminal to handle MPEG II video streams, which has frames of dimension $720 \times 486 \times 20$.

We first look into the details of the video data and associated operations.

***Stop*:**

When the user selects STOP, the Video Terminal will stop immediately, mark the current frame, and send STOP command and other control information to the Video Server, which will stop transmitting, do some bookkeeping, and then enter a waiting session until receiving the next PLAY command sent by user. In the meantime, the Video Terminal will enter a waiting state.

When, for instance, a PLAY command is issued by the user, the Video Server will resume transmitting and the Video Terminal will begin decoding and displaying the new frames.

Play:

The PLAY mode always begins with the current I frame: each frame is packed with an error recovery code which is able to recover some small errors and detect larger ones. When some unrecoverable error occurs, one way to rescue it is to redisplay the last frame while receiving a new one. If two or more errors occur in a sequence, the Video Terminal will interrupt the PLAY mode automatically, go to a check mode to investigate the channel, and inform the user of the emergency.

Pause:

This is basically similar to the STOP mode, except that the Video Terminal will also display the last frame on the screen . Resuming the PLAY mode is the same as the PLAY mode going from the STOP mode.

Fast Forward, Forward, and Picture Forward:

We can operate the "Forward" function from three modes to achieve different speeds and features. If we operate from a STOP mode, the function is Fast Forward (F.FWD). In this case, the Video Terminal will only display TIME_STAMP information provided by the Video Server. One easy way to implement this function is to retrieve information from I frames, which makes the actual speed 12 times as fast as the normal one.

If we operate from a PLAY mode, the function is Forward (FWD). In this case, the Video Terminal will display a sequence of fast changing pictures. Because the video frames have to be retrieved from the Video Server, it is not possible to accomplish FWD at a rate equal to F.FWD. Instead, the Video Server skips B frames and sends only I and P frames, so as to achieve a speed 3 times as fast as the normal speed.

If we operate from a PAUSE mode, the function is Picture Forward (P.FWD.) This function achieves the "slow motion" effect by advancing the frames at a slower rate. Obviously, this can be achieved by performing the PLAY function at say, 1/3 the normal speed.

Fast_Rewind, Rewind and Picture Rewind:

The "Rewind" Function, like the Forward function, can be operated from different modes.

From the STOP mode, this function is Fast Rewind (F.RWD). In this case, the screen is black (no picture is displayed). Consequently, this can be handled just like F.FWD, where the Video Server sends only TIME_STAMP information to the Video Terminal.

From the PLAY mode, this function is Rewind (RWD). Recall that the video stream can be viewed as a sequence of "packets", where each "packet" is itself a fixed sequence of frames of the form IBBPBBPBBPBBPBB. That is, a packet always starts with an I frame. During Rewind, the Video Server transmits the packets in reverse temporal order, i.e., in the reverse order of their appearance in the PLAY mode. Unlike F.RWD (where the screen is black), RWD should display the video stream on the screen. To do this, the Video Terminal requires a storage buffer that can hold the frames decoded from the packet. (The packet itself -- which consists of undecoded frames -- requires just a small data buffer). For MPEG II, the dimension of a standard video frame is 720 × 486 × 20 (875KB). Thus, a decoded packet (which consists of 15 frames) will require at least 12MB of storage, which is unreasonably large. We can cut the memory requirement to between 5-8 MB by requiring the Remote System to skip the B frames and only transmit the I and P frames (just like the FWD mode). But even 8 MB of video memory is too much. To cut more, we must use another method. It is reasonable to assume that when rewinding, the viewer cares much more about the content of the picture than the quality. Hence, we can reduce the 20-bit color to 12 bits, or even 8 bits. This way, we can save 2/3 of the memory. Therefore, if we use a standard memory of 4 MB, 1 MB can be reserved for the program and the packet buffer and 3 MB can be used for buffering the decoded frames. More precisely, we can store 3MB/(720 × 486 × 8) = 8 frames, which is enough for one packet consisting only of I and P frames.

In PAUSE mode, we would like to have a speed 1/3 slower than the normal speed. (Picture Rewind or P.RWD). But we should provide the whole detail of the video stream. As discussed earlier, a single video frame (MPEG II) requires 875 KB so that 3 MB of video buffer can hold only 3 frames. However, since the undecoded packet is stored in a buffer, the Video Terminal can display the first 3 decoded frames at 1/3 the speed, while decoding another 3 frames.

The remaining problem is how fast the processor be? An HP750 workstation can achieve 10 - 15 frames per second[3], which is within a factor of two of real-time performance. But with conventional Intel chips, such as Pentium which is widely used in personal computers, it is hard to perform real time decoding, since they are 4 times slower than an HP750. Therefore, a fast co-processor is needed to handle MPEG II video streams (at least twice as fast as an HP750). Fortunately, MPEG I chips are now available and it is reasonable to expect that MPEG II chips will soon be available.

To display the video, we also need a fast video monitor. To provide 30 frames/sec, we will need 30 × 850 KB = 25.5 MB/sec of throughput, which is about 51 MHz of bandwidth. A conventional super VGA usually has over 50 MHz of bandwidth, so it is suitable to use as a video monitor.

The system diagram of the video system is shown in Figure 2.2.

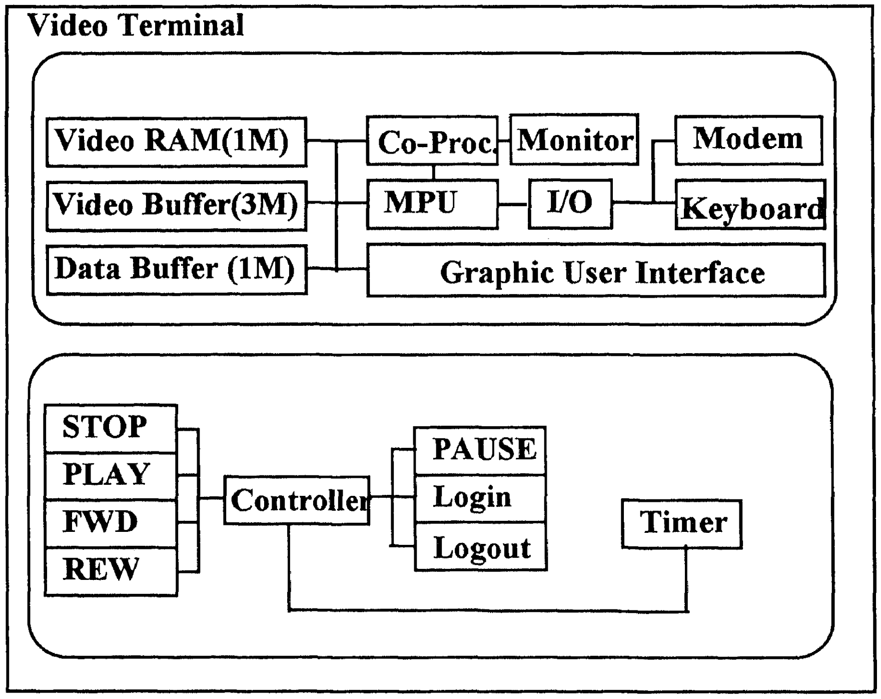


Figure 2.2 System Diagram of Video Terminal System

2.4 Reservation Policy, Service Points and Sessions

In general, requests for video services can arrive at arbitrary times. Serving these requests as soon as they arrive requires a sophisticated dynamic scheduling algorithm which incurs a significant overhead on system resources. Moreover, even an optimal dynamic schedule will inevitably fail when it is flooded with thousands of calls requesting immediate service all at the same time. The total resources required by these requests may well exceed the system capacity, these risking a system crash. Consequently, it is advantageous to encourage the customers to reserve their videos well ahead of their preferred viewing time.

Through a study of the local video store, we found that the peak period for video viewing is during the evening: typically between 6 p.m. and 2 a.m. To encourage reservation ahead of this peak period, a dynamic price list can be used. For example, beyond 8 hours from the peak period, the charge might be \$1/hour of video services; within 4 to 8 hours, the charge might be \$1.25/hour; within 1 to 4 hours, \$1.50/hour; and within 1 hour or less, \$2/hour. In addition, "top ranked" videos might be charged less than less requested videos. Of course, we do not intend to make a detailed price list here. Our expectation is that, by applying a reservation policy, 90% of the customers will make reservation before the peak period, while the remaining 10% will prefer to request and receive service during the peak period.

Even with an advanced reservation, a customer request may specify an arbitrary viewing time. For example, one request might want to view Video 1 at 7:03 p.m. and another request might want to view the same video at 7:04 p.m. This requires sending two video streams which only differ temporally; i.e., one stream is a time-shifted version of the other. In contrast, suppose we impose the restriction that viewing times should begin at fixed time intervals, e.g., every 5-minute interval past the hour. Then the two requests can both be served at 7:05 p.m. by sending duplicates of the same video stream to both customer sites.

A session uses a certain amount of system resources, such as network bandwidth, storage, CPU processing time, modem, etc. The total resources consumed by a session is called a Service Point. Note that the number of sessions is not necessary equal to the number of Service Points: Because if two or more customers request the same video at the same time, their sessions can share one service point This is called *duplication.* Obviously, duplication can increase the capacity of the system.

The video servers described in this thesis are designed to start video sessions at every 5-minute interval past every hour. With this restriction, a customer will wait 5 minutes in the worst case, and 2.5 minutes on average, past his preferred viewing time before the actual video session begins. This, we believe, is a reasonable waiting time. On the other hand, adopting this restriction significantly enhances the probability of duplication (especially for top ranked videos) and hence increases the total number of simultaneous video sessions that can be handled by the system.

2.5 Workload Model

The performance of the video servers developed in this thesis is analyzed in terms of a workload model which we now describe. We assume that the average length of a video session is 2 hours. Each video has an associated request frequency, which measures how often it is requested compared with other videos. We assume that the total number videos is N = 500 and that the video request frequencies follow a Gaussian Distribution with variation $\sigma = N/10 = 50$. Based on these assumptions, 99% of the requests will be for 258 of the videos. Similarly, 90% of the requests will be for 166 of the videos.

We assume that viewing times (i.e., the times at which video sessions start) occur at every 5-minute interval past every hour over an eight-hour peak period (6 p.m. to 2 a.m.). The Poisson Distribution can be used to model the number of video sessions started at a specific viewing time. The basic Poisson Distribution Function is :

$$P(k,\alpha) = \frac{\alpha^k}{k!} \cdot e^{-\alpha}$$

As illustrated in Figure 2.3, this simple function is not realistic.

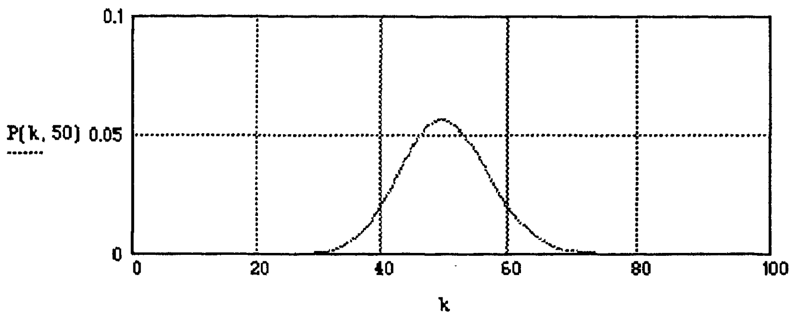


Figure 2.3 Basic Poisson Distribution Function

A more realistic model would be a joint Poisson Distribution Function with Time Decaying. The function is:

$$i = 1 \ldots 4 \qquad T(k) = \sum_i \frac{P(k, 13 \cdot i + 2)}{1 + 0.01 \cdot i + 0.04 \cdot i^2}$$

Figure 2.4 illustrates this function:

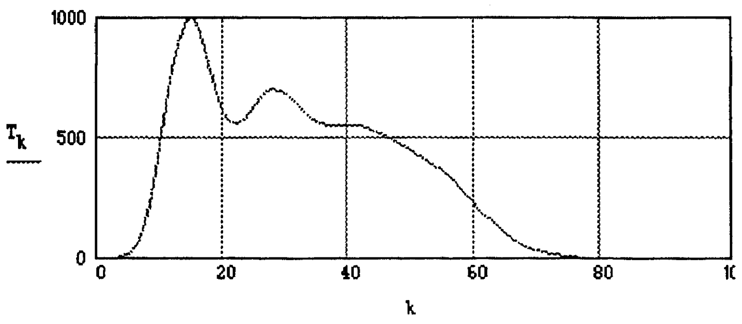


Figure 2.4 Joint Poisson Distribution Function

The horizontal axis, k, denotes the number of 5-minute intervals past 6:00 p.m., the start of the peak period. For example, $k = 0$ represents 6:00 p.m., $k = 1$ represents 6:05 p.m., etc. The vertical axis, $T(k)$, denotes the number of video sessions to be started at the viewing time k. Henceforth, Figure 2.4 will be referred to as the reservation table.

We assume that the video service system serves a mid-size city with 10,000 families. Remember that this is a paid service, so we expect that roughly 25%, or 2500 families, will request videos for viewing during the eight-hour peak period. If we further assume that the average family will view 1.25 videos, then the total number of video sessions started during the peak period will be $2500 \times 1.25 = 3200$. (That is, for the function depicted in Figure 2.4, $\sum T(k) = 3200$) Finally, we assume that from these 3200 video sessions, 90% come from reservations made before the peak period.

CHAPTER 3

A CENTRALIZED VIDEO SERVER BASED ON A HYPERCUBE PARALLEL COMPUTER

3.1 Structure of a Hypercube Parallel Computer

Oracle Corporation and NCube Corporation recently announced that they are co-developing a centralized video server based on the NCube parallel machine. The NCube parallel machine is a collection of identical processors connected by an interconnection network. The topology of the network is the hypercube, which connects $N = 2^n$ processors as follows: Let the processors be $P_0, P_1, \ldots, P_{N-1}$. Thus, each processor index is represented as an n-bit binary number. In the hypercube network, processor P_i is connected to processor P_j if and only if the binary representation of i and j differ in exactly on bit position. Figure 3.1 illustrates the hypercube network for N = 8.

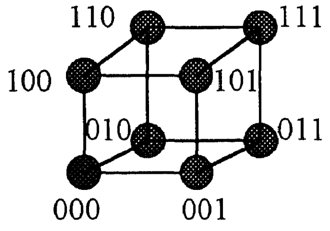


Figure 3.1 Structure of HyperCube Parallel Computer

In the NCube machine, each processor has its own local memory for storing programs and data. There is no globally shared memory -- processors communicate by passing messages over the network. The NCube is a MIMD architecture; the processors can execute different programs on different data.

In what follows, we describe the design of centralized Video Server based on a 64-node hypercube parallel computer. We use this model as the base line for the distributed Video Server discussed in the next chapter.

3.2 Design of a Hypercube Based Video Server

A Video Server, regardless of its architecture, must necessarily connect to a number of channels that link up with the customer Video Terminals (See Figure 3.2). Figure 3.3 illustrates one possible implementation of this Video Server using a Hypercube parallel machine.

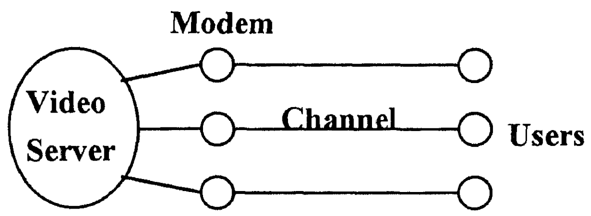


Figure 3.2 Simplest Model of Video Server System

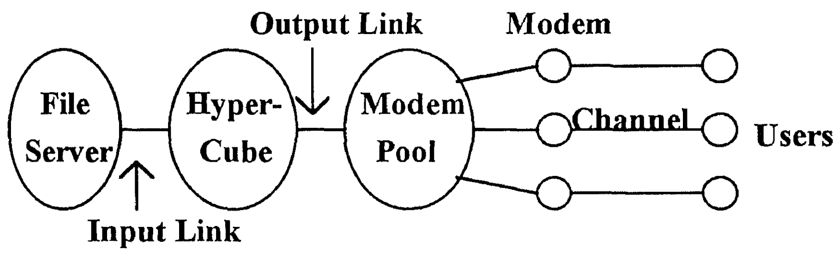
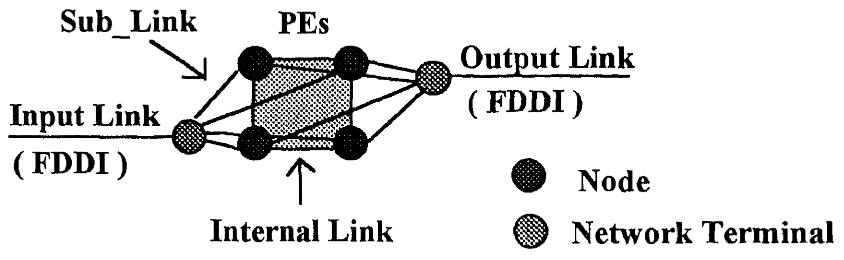
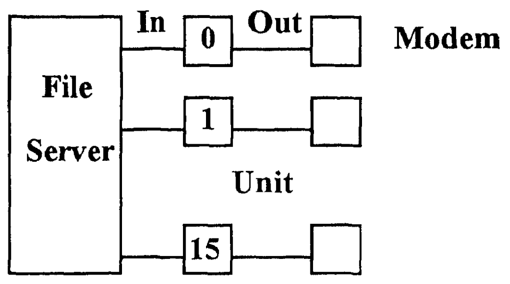


Figure 3.3 Basic Model of Hypercube System

The function of the File Server is to store all the videos. The function of the Hypercube machine is to retrieve video stream from the File Server in response to user requests, and dispatch them to the appropriate modems for transmission to the user sites. To achieve real-time response, the bandwidth of the input link that connects the Hypercube with the File Server should be as high as possible. On the other hand, it is not possible to connect the input link directly to a single Hypercube node since it is not fast enough to process the incoming data at a very high rate. However, note that 4 PEs are normally integrated into one circuit board; therefore, a single data link can be used to connect each board (henceforth called unit) with the File Server (see Figure 3.4). This way, we decrease the demand on each data link. As we shall see soon, the bandwidth required for each input / output link is still high; hence, FDDI should be used for each link.



(a) Internal Topology



(b) Global Connection

Figure 3.4 Data Link Topology of Hypercube System

3.3 Video Stream Allocation on the Hypercube

We now describe how video streams from the File Server are handled by the Hypercube system. We use the FDDI link connecting the File Server with a unit of the Hypercube as a packet switched network: a video stream is sent out as a sequence of packets, each containing video data, and some additional information, such as video ID, packet sequence number, etc. The packets comprising a single video stream need not arrive contiguously at the unit; packets from different video streams may be interspersed, as shown in Figure 3.5. There are, however, two constraints. First, the packets from a single video should arrive at the unit in the right sequence; i.e., for the same video stream, packet 1 should arrive earlier than packet 2, packet 2 should arrive earlier than packet 3, etc. The second constraint is that the packets from the same video should arrive at the unit at an average rate no less than 1.5 Mbps, so that the Video Server can output the video stream at the normal (MPEG I) speed. Packets arriving at a unit are properly sequenced before being sent out to the output links. Specifically, the packets comprising a single video stream must be collected and sent out as one contiguous sequence and in their correct temporal order. Within the Hypercube, a single incoming video stream may be duplicated as several identical outgoing video streams. This, for instance, is necessary if there are several users wishing to view the same video at the same time.

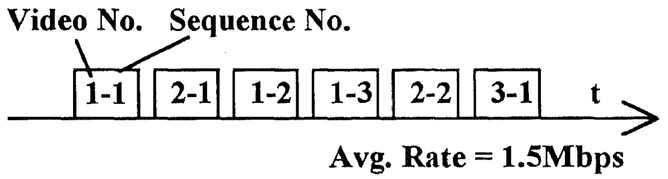


Figure 3.5 Data Stream on the File Server - Hypercube Unit

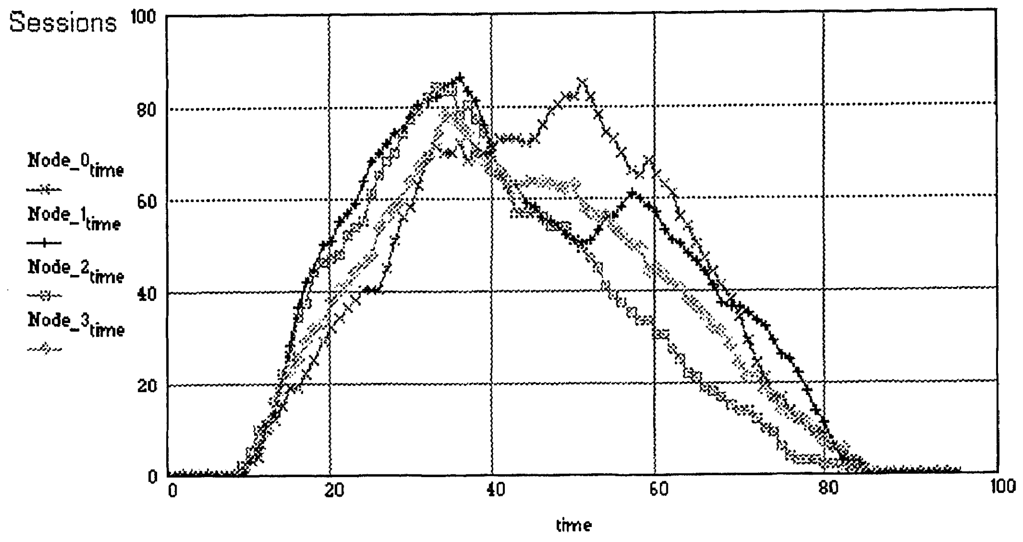
Video packets arriving at a unit should be distributed among the four PE's comprising the unit. Packet allocation is done by the Network Terminal of each unit (see Figure 3.4 (a)). We describe two packet allocation strategies. The first strategy is Round-Robin allocation. In this scheme, the Network Terminal allocates the packets to the PE's in Round-Robin fashion: the first packet is assigned to PE 0, the second packet to PE 1, the third packet to PE 2, etc. This strategy balances the workload of the PE's as much as possible. Moreover, it requires only a simple Network Terminal design. The drawback is that, in order to properly sequence the packets, the PE's need to query other PE's. This entails a considerable amount of message-passing among the 4 PE's.

The other strategy is called Addressed Allocation. In this scheme, each video stream is assigned to a unique PE. For example, if video 1 is assigned to PE 0, than all video 1 packets are sent to this PE. This strategy does not require inter-PE communication. However, the workload of the PE's can potentially be unbalanced, since the video-to-PE assignment is static. Moreover, the Network Terminal is more complicated since the assignment function should be known to the Network Terminal.

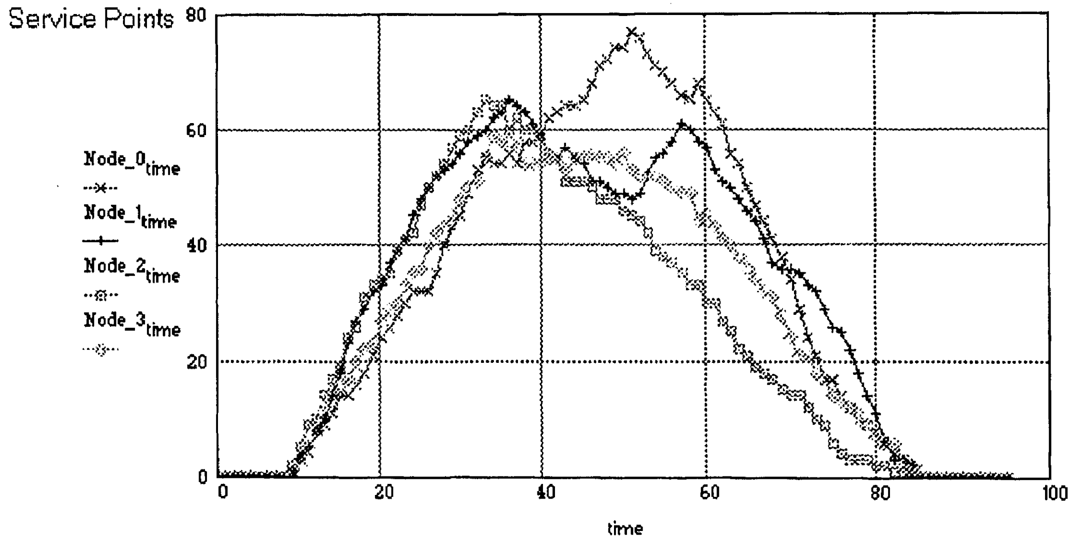
The above observations were validated by our experimental results. Using the workload model, we simulated one unit of the Hypercube using both packet allocation algorithms. As indicated by Figures. 3.6. and 3.7, Round-Robin Allocation is better than Addressed Allocation in that the loads of the 4 PE's are balanced. However, here we assumed that the bandwidth of the internal links that connect the PE's is unlimited. When the system load increases, the actual limited bandwidth of these links will become the bottleneck. Addressed Allocation, on the other hand, does not require a lot of inter-PE communication. However, as shown in Figure 3.7, there is a significant imbalance between PE's using Addressed Allocation.

We propose a hybrid strategy. During the busy period, we use Addressed Allocation for lower ranked videos and Round-Robin allocation for top ranked videos. When the system is not busy, we use Round-Robin allocation. The advantage of the hybrid strategy is that it adapts well to dynamic workload condition. The disadvantage is that the design of the Network Terminal is more complicated.

If we sum up the load of the 4 PE's, we obtain a graph reflecting the overall performance of the unit (see Figure 3.8). From the graph, we see that the maximum number of sessions is about 310 and the maximum number of service points is about 240. Thus the maximum bandwidth across the File Server - Hypercube Unit Link (See Figure 3.4) should be 240×1.5 Mbps = 360 Mbps, which can be handled by FDDI.

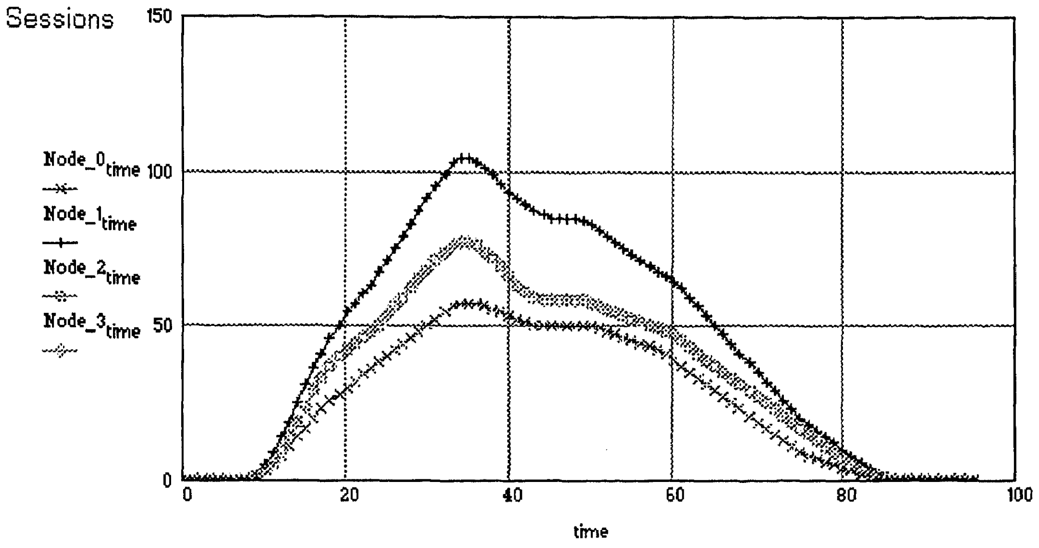


(a) Sessions

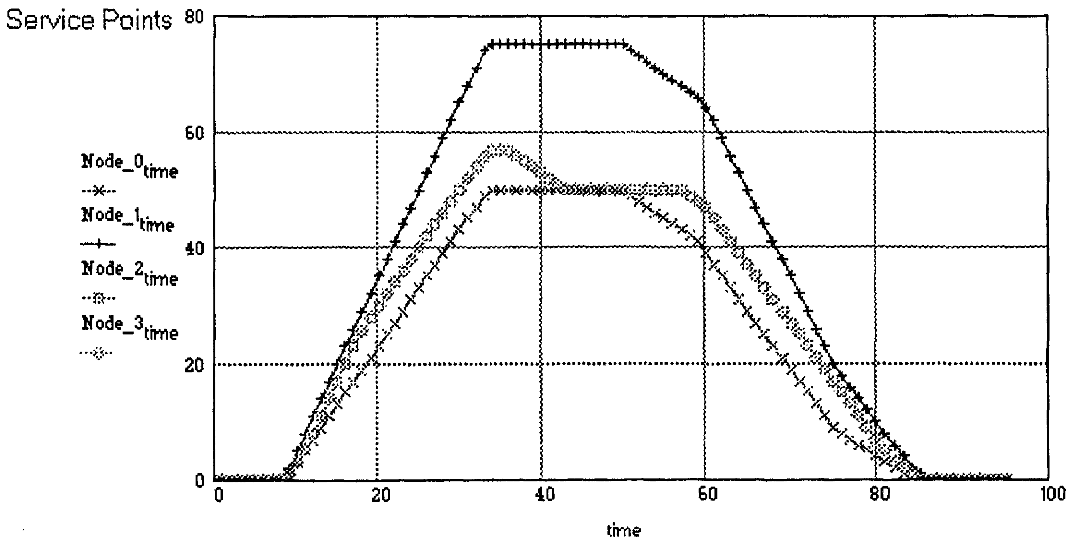


(b) Service Points

Figure 3.6 Performance of 4-PE Hypercube Unit using Round-Robin Allocation



(a) Sessions



(b) Service Points

Figure 3.7 Performance of 4-PE Hypercube Unit using Addressed Allocation

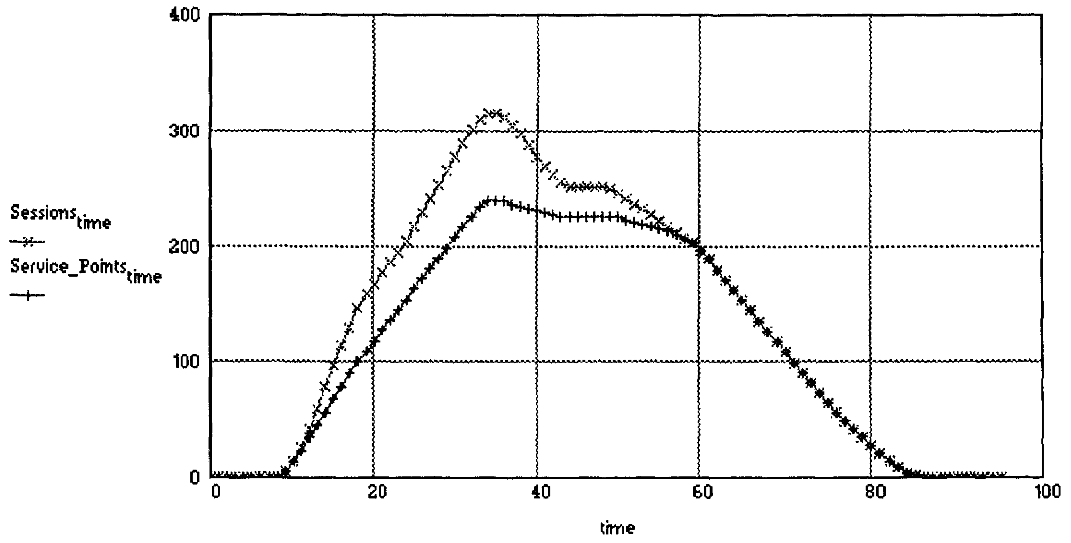


Figure 3.8 Overall Performance of a Hypercube Unit

Observe that there is a difference between the number of service points and the number of sessions the unit can handle. The difference is the duplication factor, which is about 310 / 240 = 1.3. Since the number of sessions represents the actual number of video streams sent via the output link to the modems, the bandwidth of the output link should be 1.3 times that of the input link (about 470 Mbps). Therefore, FDDI should also be used for the output link.

To summarize, a 64-node Hypercube system can provide a total of 240 × 16 = 3840 service points, or a total of 310 × 16 = 4960 sessions. This sufficiently covers the 3200 sessions required by the workload model described in Section 2.4.

3.4 File Server Architecture

We now discuss the architecture of the File Server. In order to reduce the price, all original videos should be stored permanently in CDs as it is the cheapest way. Of course we can use an optical storage, such as Optical JukeBox, but we predict that in the future all video media will go to CD which everyone can afford, and it is easy to change a video CD than overwriting on an Optical JukeBox.

The drawback of CD-ROM is that it is slow. The fastest CD-ROM currently available is 4× speed (which translates to an average transfer rate only 650KB/sec). Thus, we need to pre-load the video streams on faster media -- much as magnetic disks -- before service could begin. Moreover, the total storage capacity should be large enough to hold all, or most, of the videos.

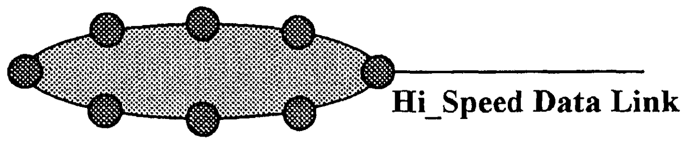


Figure 3.9 Structure of File Server

One possible organization of the File Server is shown in Figure 3.9. The CD-ROMs that hold the original videos are distributed across several nodes. Each node has RAID-type magnetic storage used for pre-loading the videos. The internal links are used to transfer video data from one node to the other. Observe that the File Server itself is a distributed system!

Further study shows that an important factor contributing to the total cost is the number of high speed links. Moreover, if we incorporate duplication into the File Server, then the bandwidth of the output links connected to the modems can be drastically reduced. Better yet, if we distribute the modems among the nodes, we can essentially discard the high speed link! Now we open the door to our distributed architecture.

CHAPTER 4

DESIGN OF A DISTRIBUTED VIDEO SERVER

4.1 Study of Distributed Architecture

Before designing the distributed server, we should first decide on the topology of the system. There are three typical topologies: STAR, BUS and RING. We prefer the RING structure because it is balanced: the longest distance between 2 nodes is less than a BUS system, and there is no central node as there is in STAR structure.

The drawback of the RING structure is that as the number of nodes increases, the effective bandwidth available to each node decreases. On the other hand, the total number of nodes should be large enough to store and provide several hundreds of video streams. To increase the number of nodes while not limiting the bandwidth available to each node, the architecture can be modified to that shown in Figure 4.1. The nodes along the RING structure (henceforth called Front Nodes) are connected by high speed FDDI links. In turn, each Front Node has an Assistant Node connected to it by a slower (and cheaper) sub-link. If more Assistant Nodes are placed on the sub-link, can the high-speed link be discarded? The answer is negative, because the bottleneck is the maximum throughput of the lower speed link. For example, Ethernet has a maximum bandwidth of 10 Mbps which can hold 10 / 1.5 = 6 video streams at most (1.5 Mbps is the MPEG I data rate). Therefore, the number of Assistant Nodes is limited.

Observe that the Assistant Nodes in each sub-net have no direct connection with other nodes in another sub-net. Consequently, communication between two Assistant Nodes in different sub-nets has to be done via the Front Nodes. This is not efficient. We want every node to have some connections with nodes from another sub-net.

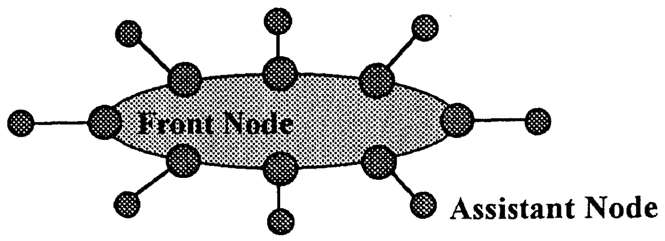


Figure 4.1 An Architecture of Distributed System

4.2 Network Architecture

The network architecture we propose is depicted in Figure 4.2:

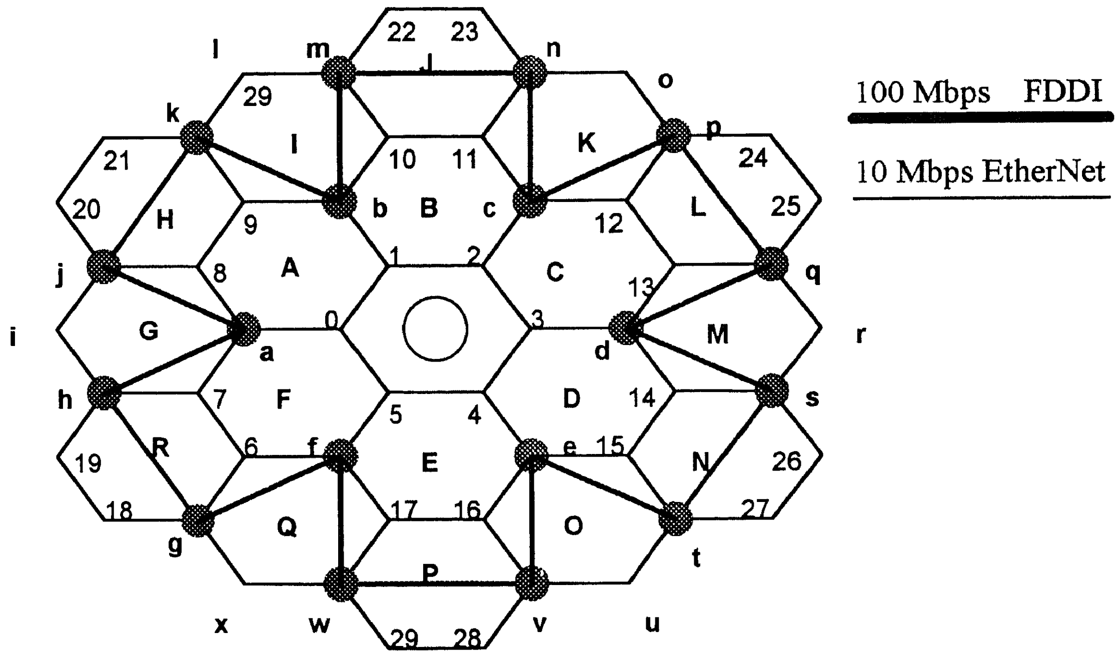


Figure 4.2 Video Server Network Architecture

It contains a high speed data highway, using 100 Mbps FDDI, and 18 local traffic links using 10 Mbps Ethernet. Each Front Node (indicated by the large black circle) has 3 Assistant Nodes. For example, Node *a* has Assistant Nodes *0*, *7*, and *8*. Note that there is some overlap: node *j* also has *8* as an Assistant Node. In this architecture, when one node gets “hot”, it can easily get help from other nodes around it, thus making network load balancing easier.

One can view the network architecture as a single "layer," and we can use several layers of this kind to build a larger system. To share information and data between each layer and to communicate with central service computer, the system needs an outside channel. Here we connect 6 Front Nodes, *a*, *b*, *c*, *d*, *e* and *f*, to the outside. The connection can use either an Ethernet when there are small number of layers, or a FDDI link when it is a large system. It is also flexible to expand and configure. That is one of the merits of using a distributed system.

As we shall see later, two layer of the network architecture are needed to provide the same performance as the 64-node Hypercube system described earlier.

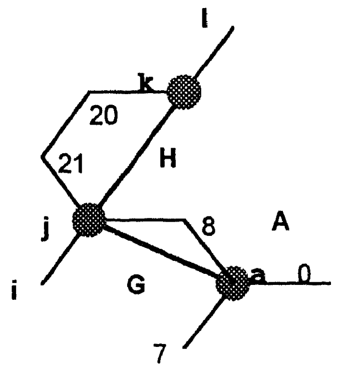


Figure 4.3 One Piece of the Distributed Network

The videos are stored in the individual processing nodes of the distributed network. Because only the Front Nodes are connected to the FDDI Data Highway, frequently requested videos are stored in these nodes in order to have fast response. Less frequently requested videos are stored in the Assistant Nodes. We now discuss how the videos are allocated to the processing nodes. One layer of the distributed network (see Figure 4.2) can be partitioned into 6 identical pieces, as illustrated in Figure 4.3. Henceforth, we refer to a single piece of the network as a Unit. Observe that a Unit consists of 3 Front Nodes and 7 Assistant Nodes.

Since two layers are used in the design and there are a total of 500 videos to be stored, each layer should hold 500 / 2 = 250 videos. With a total of 18 Front Nodes in each layer, each Front Node should store 250 / 18 = 14 videos.

As we have derived in Chapter 2, 166 out of the 500 videos will be requested 90% of the time; hence each layer will have 166 / 2 = 83 videos requested 90% of the time. This implies that each Unit has 83 / 6 = 14 of these videos, and each Front Node has 14 / 3 = 5 of these videos.

Consider now the videos requested 99% of the time, which is a total of 258. For this case, each layer should have 258 / 2 = 129 videos, and each Unit should have 129 / 6 = 22 videos. 14 of them are requested 90% of the time. Therefore, we can map the remaining 22 - 14 = 8 videos to the Assistant Nodes, where each node will have 8 / 7 = 2 videos.

4.3 Node Architecture

Figure 4.4 shows the organization of a node:

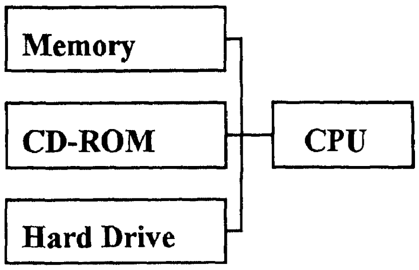


Figure 4.4 Node Architecture

The CD-ROM is used to hold permanent copies of the videos. To achieve a balanced load in the network and faster access to the video streams, frequently accessed videos are pre-loaded in the Hard Drive by the scheduling process (described later). The memory is used for buffering the data read from the local CD-ROM or HD and the data received from other network nodes.

In our design, two types of node processors are used. An Intel Pentium processor can reaches 112 MIPS and its PCI bus can achieve 132 MB/sec[4], and hence is suitable for a Front Node Processor. Since Assistant Nodes work only for the Front Nodes, they do not have to fully equipped. An Intel 486 DX2-66 is rated at 35 MIPS cost less than a Pentium processor, and is suitable for an Assistant Node.

4.3.1 Assistant Node

- Main Memory

The Intel 486 DX 2-66 processor can reach 35 MIPS, but we found that the bottleneck is not the speed of the processor, but the speed of memory. Access times of typical DRAM used in PCs is 70 nsec. Assuming a 32-bit data bus (e.g., VESA Local Bus), the maximum bus speed is:

$$\frac{4\ \text{Bytes}}{70\ \text{ns}} = 57\ \text{MB / sec}$$

Figure 4.5 depicts how video data flows through the system. Each distinct video stream is assigned a segment in main memory. The video stream is transferred to its segment on a block-by-block basis using DMA. Each block is first transferred into a channel buffer, then copied to the main memory segment. (The channel buffer is shared by several CD-ROMs and HD's.) The CPU then performs some bookkeeping on the memory segment copy (e.g., incorporates session information such as User ID, modem port, video status, etc.) then dispatches it to a modem buffer for transmission to the customer.

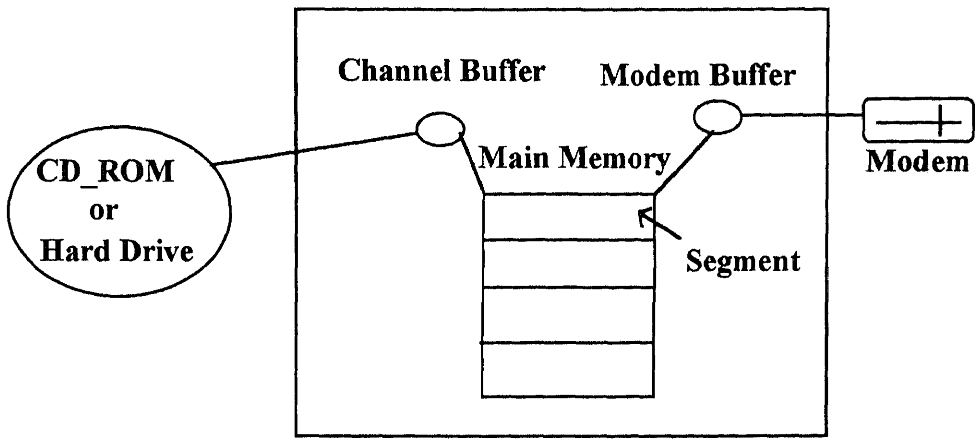


Figure 4.5 Data Flow through the system

We now estimate the number of separate video streams that can be processed simultaneously at the rate of 1.5 Mbps. This number is an upper bound on the number of service points then can be provided by the assistant node. To send out a video stream, we need the following memory accesses:

1. At least one read from CD-ROM or Hard Drive to channel buffers;
2. At least one read from channel buffer and one write to memory;
3. At least one read from memory and one write to modem buffer;

Each of 1,2, and 3 above can be achieved by DMA (Direct Memory Access).

4. Sending out a video stream incurs some overhead. This includes the time needed to set up the DMA and handle the interrupt at DMA completion; to perform bookkeeping on video data that is read into memory (e.g., determining the locations of the frames) and to access database information about each session, such as User ID, modem port, video status, etc. We make the pessimistic assumption that the overhead accounts for an additional two memory accesses per byte of video data.

Therefore a video stream requires a total of 7 memory access per byte of video data. This calculation assumes no duplication of the video stream. Let us now discuss how duplication is achieved: Assume two customers want to watch the same video at the same time; thus, two sessions should be run simultaneously. These two sessions can be connected to one channel buffer, which gets data from a HD or CD-ROM. For this case, the total number of memory accesses is 11: 1 read from CD-ROM or HD to channel buffer, 1 read from buffer and 1 write to memory, 2 reads from memory and 2 writes to the modem buffers and an additional 4 accesses due to overhead (two for each session). In general, assuming a duplication factor of x, a video stream requires 3 + 4x memory accesses. This implies that for a duplication factor of x, the average number of memory access per video stream is (3 + 4x) / x. Since the maximum bus speed is 57 MB/sec, the maximum number of video streams (or sessions) that can be sustained, assuming a duplication factor of x, is:

$$\frac{(57\ \text{MB}/\text{sec})/\text{access}}{\left(\frac{3+4x}{x}\right)\frac{\text{accesses}}{\text{video stream}}} \times \frac{8\ \text{bits}/\text{byte}}{1.5\ \text{Mbps}} = \frac{304x}{3+4x}\ \text{sessions.}$$

For example, with no duplication (x = 1), the maximum number of sessions is 304 / 7 ≈ 43. Note that this is also the number of service points. If x = 1.5, then the maximum number of sessions is 50. Figure 4.6 shows the graph of the number of sessions vs. the duplication factor.

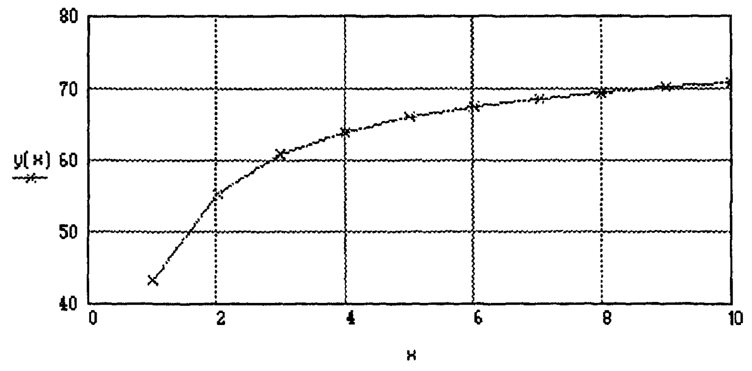


Figure 4.6 Relationship of Duplication and Sessions for an Assistant Node

Notice that at a maximum bus speed of 57 MB/sec, the memory can dynamically sustain a varying number of sessions (depending on the duplication factor).

Let us take the conservative view that the system can only handle a total of 38 sessions, which is 90% of 43, the number of service points. We can safely assume that the other 10% is for the operating system.

We now calculate the main memory capacity: Assuming a duplication factor of 1.5 (which is slightly better than the 1.3 duplication for the Hypercube system), the number of service points (i.e., different video streams) that need to be supplied by the HD or CD-ROM is 38 / 1.5 =25. We show later that the CPU can attend to each service point once every second. Hence, each memory segment should be large enough to hold one second of videos. Since the MPEG data rate is 1.5 Mbps, each service point requires a memory segment of 192 KB to hold one second of video. Therefore, the 38 service points require a total of 0.5 $\times$ 192 KB = 4.7 MB. To buffer I/O, CD-ROM needs 768 KB (4x Speed, 3 videos) and we need two buffers for the CD-ROM in order to perform Read/Write. The HD will need two 192 KB buffers which is 384 KB. We will also need some memory for network packets and bookkeeping. That makes a 1.3MB of buffer. The system program will occupy 2 MB of memory with data. Then total memory is about 8 MB.

- Mass Storage

We use CD-ROMs to hold videos. Based on current CD storage capacities, a single 2-hour video requires 2 CD's. Now we can find 4x and 2x speed CD-ROM Drives in commercial products with 18 and 6 CD changers; hence, 9 or 3 videos can be stored in them. Since we only need to hold 2 videos, we can use a 4x speed CD-ROM w/6 CD changers for an Assistant Node. A 4x speed CD-ROM drive is fast enough to access 3 videos at normal speed. The reason we use an expensive 4x speed CD-ROM instead of a 2x speed CD-ROM is that the bandwidth of the Hard Drive is limited, as we will soon see.

Next we design the Hard Drive to pre-load video data. By using VESA Local Bus, a Hard Drive can reach an average of 1MB/sec of data transfer rate. This means that only 1 MB/sec $\times$ 8 bits/Byte / 1.5 Mbps = 5 service points maximum will be available from a Hard Drive. This is apparently not enough since 25 service points need to be supplied to the processor. From these 25 service points, the CD-ROM can provide 3 and other network nodes can provide 2. Then the remaining 20 should be provided by the Hard Drive. One IDE controller card can hold 2 Hard Drives; therefore we use 2 IDE controller cards, and 4 Hard Drives to produce the 20 service points.

Next, we consider the capacity of the Hard Drive. Since service points are allocated by the system at 5-minute intervals, the Hard Drive should have enough storage to hold 5 minutes of video per service point. Since a single Hard Drive can provide at most 5 service points, the total storage capacity is:

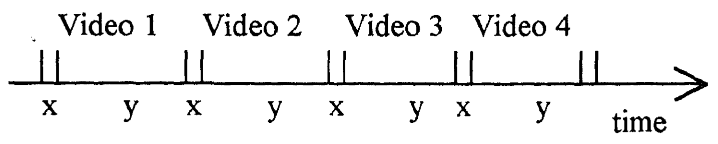
$$25 \text{ min.} \times 60 \text{ sec / min.} \times 1.5 \text{ Mbps / 8 bits / Byte} = 281.25 \text{ MB}$$

Therefore, we can use a standard 340 MB Hard Drive.

Finally, we consider the CPU time slice. This is a very important parameter, because the more the CPU slices per second, the faster the system can respond to our customers. But the CPU should match the speed of the Hard Drive.

Hard Drive:	Average Seek Time :	12 ms
	Data Transfer Rate :	1 MB/sec

Assume : CPU has n time slices; Hard Drive is allowed to access 5 times per slice;



x: Time used for seeking.

y: Time used for reading.

Figure 4.7 Time slices for Hard Drive access

Figure 4.7 shows the data access on a Hard Drive. Since a Hard Drive supplies 5 service points, in one second, the system should access the hard drive at least 5 times to retrieve 5 video data streams at an average rate of 1.5 Mbps. So 12 ms × 5 accesses × n slices of time is the total seek time (the sum of the x's in Figure 4.5). Therefore, the remaining time that can be used to read data is 1000 ms - total seek time. Since the data transfer rate should equal 1.5 Mbps, we have:

$$\frac{1000\text{ ms} - 12\text{ ms} \cdot 5\text{ accesses} \cdot n\text{ slices}}{1000\text{ ms} \cdot 5\text{ accesses}} \cdot 8\text{ Mbps} = 1.5\text{ Mbps}$$

where n = 1.042. Therefore, we only have 1 slice per second. This implies that the system can respond to a video command (e.g., PLAY, STOP) in about 1 second. With the assistance of the Video Terminal, this 1 second of response delay will be invisible in most of cases because we have designed 1MB of data buffer in the Video Terminal which can hold 5 seconds of video data.

Now that we have designed the Assistant Node with Intel 80486 DX 2-66, we summarize the features below:

- Intel 80486 - DX 2 - 66 (35 MIPS);
- 8 MB 70 ns DRAM;
- VESA / EISA 32 bits Local Bus;
- Four 340 MB Hard Drive, with Average Seek time 12 ms, and Data Transfer Rate 1MB/sec;

• One 4x Speed CD-ROM with 6 CD changers;

As we have calculated earlier, each Assistant Node can supply 25 service points total. At 1.5 duplication factor, the total number of sessions that can be provided is 38.

4.3.2. Front Node

This Front Node processor is different from the one we designed before. Here we use Pentium not only because of its high speed, but because of its PCI Bus. So the purpose for which we are going to design this type of Front Node is to provide a certain number of service points that can take the main load of the system and have additional resources to perform other important cloves such as network load balancing..

We have calculated that each Front Node has to store 14 videos. Two 4x speed CD-ROMs w/18 CD changers can hold 18 videos. This is enough to store the 14 videos; the remaining space could be used to store copies of videos from other Front Nodes to have some redundancy.

We use the same procedure as in the previous subsection. The Pentium has a 64-bit data bus; assuming a 70 nsec memory access time, we can derive that it can reach 114 MB/sec. With 7 accesses per video, the maximum number of service points is 84. As before, we assume that 10% of the service points is used by the operating system; thus we will have 76 service points dedicated for video service.

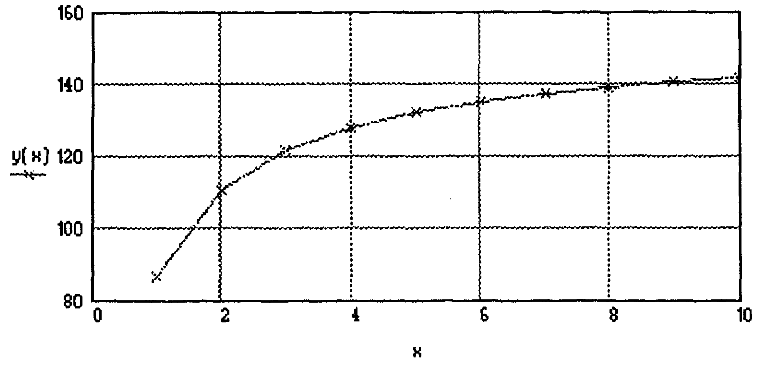


Figure 4.8 Relationship of Duplication and Sessions for a Front Node

Again by using duplication, we can diagram the relationship between duplication and the number of sessions (see Figure 4.8). We find that for the same duplication factor, the number of sessions is twice that for an Assistant Node.

We next determine the main storage capacity needed to deliver 76 service points. A 4× speed CD-ROM can supply 4 service points provided we buffer the video data on a Hard Drive. We list some of the parameters of a typical 4 x speed CD-ROM (specifically, a PIONEER DRM-604X):

- 612 KB/sec transfer rate;
- Average access time of 300 ms;
- 128 KB buffer;
- Disc exchange time of 5 seconds;
- Burst mode transfer speeds of 4.2 MB/sec;
- Up to 7 SCSI daisy chain available;

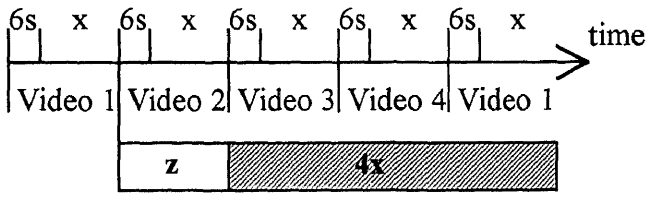


Figure 4.9 Data Access from a CD-ROM

To change a disc takes about 5 seconds. We assume the worst case that it takes 6 seconds to change a disc and seek data. Figure 4.9 depicts the data access from the CD-ROM (which supplies 4 video streams). In the figure, x minutes is spent reading one video stream before switching to another one. This x minutes actually represents 4x minutes of video data because the CD-ROM is 4x speed. One time slice consumes ($4x + 4 \times 6 / 60$) > 4x minutes. Hence, an additional $z = (4x + 4 \times 6 / 60 - 4x)$ minutes of video data have to pre-loaded on the Hard Drive. Suppose that the Hard Drive has 2y minutes of buffer reserved for this purpose (y minutes of buffer is used to accept incoming data, while the other y minutes of buffer is being output). Then, based on our assumption that each video lasts 2 hours, the amount of video data that would still have to be read from the CD-ROM is 120 - 2y. This implies that we need to access the CD-ROM (120 - 2y) / 4x times. Each time, we need to pre-load (4x + 4 × 6 / 60 - 4x) minutes of video on the Hard Drive. Therefore:

$$(4 \cdot x + 4 \cdot \frac{6}{60} - 4 \cdot x) \cdot \frac{120 - 2 \cdot y}{4 \cdot x} = y .$$

Assuming y = 4x, then x = 1.64 minutes and y = 6.56 minutes. Therefore, each CD-ROM drive requires 2y = 13.12 minutes of data per video pre-loaded on the Hard Drive. Since there are two CD-ROM drives and four videos per CD-ROM drive, the total pre-load capacity for the Hard Drive should be 13.12 × 2 × 8 = 106 minutes, which translates to

$$106 \text{ minutes} \times 60 \, \frac{\text{seconds}}{\text{minute}} \times \frac{1.5 \text{ Mbps}}{8 \text{ bits / Byte}} = 1192 \text{ MB}$$

Assuming that of the 76 service points, 8 are provided by the CD-ROM and 2 by other nodes of the network. Thus the Hard Drive should provide the remaining 76 - 8 - 2 = 66 service points. We can calculate that 66 service points represent of 3.7 GB of Hard Drive capacity by using the same formula used in the previous calculation. But since we can find only a 2.4GB Hard Drive in the market. We use two of them; the remaining space is sufficient for CD-ROM buffering and other usage.

Each Hard Drive need not provide more than 38 service points. Now we want to know if a Hard Drive can achieve that speed. The average seeking time for a Hard Drive is 8 ms, and the average data transfer rate for a PCI bus used in Pentium system is the same as SCSI bus, which can reach up to 20 MB/sec. But let us just assume that it can reach only 15 MB/sec.(if it can, the cost of the controller will be reduced) Then the total reading time is 1000 - 8x ms, and each access should read 1.5 Mbps of data, thus we can derive:

$$\frac{1000\text{ ms} - 8\text{ ms}\cdot\text{x access}}{1000\text{ ms}\cdot\text{x access}}\cdot 15\text{ MB/sec}\cdot 8\frac{\text{bits}}{\text{Byte}} = 1.5\text{ Mbps}$$

Solving, we get x = 48. So a maximum of 48 service points can be provided by each HD, which is certainly enough.

We now calculate the main memory capacity. This depends on the CPU time slice. It is best to synchronize the Front Node and the Assistant Nodes, as we use 1 slice per second for the Front Node, just like an Assistant Node. Thus, to handle 76 service points, we will need 76 × 1.5 Mbps / 8 = 14.25 MB of main memory, with 4 of 192KB disk buffer for two HD, and 2 of 650KB for CD-ROM, and 4MB for program and data. The total amount of minimum memory needed is 20.25 MB, we use 24 MB to make it standard.

Now that we have designed the Front Node, we summarize its features below:• Intel Pentium Processor 60 MHz (112 MIPS);

- 24 MB 70 ns DRAM;
- PCI 64 bits Local Bus;
- Two 2.4GB Hard Drive, with Average Seek Time 8 ms, and Data Transfer Rate 10 MB/sec;
- Two 4x speed CD-ROM w/18 CD changers;

As we have calculated earlier, each Front Node can supply 76 service points total.

4.3.3 Modem

The modem is a very important part. Modems are usually used to perform peer to peer communication. If we use one Modem to one customer, that is a large expense, and we have to provide many just to one service point. But if we connect two or more user to one Modem by using a digital amplifier, we can tremendously increase the system capacity, and save the expense as well.

A problem arises when we deal with incoming Video Functions. We noted that when video service is active, we use only mid bandwidth, and thus we can use lower bandwidth for communication between the Video Terminal and the Video Server to process the video command. Thus, if we make a band filter that can discharge the lower bandwidths and connect them to the Service Computer, then the Front Nodes can focus on the video processing and let the Service Computer deal with the rest. The reason why we want to do this is that we need the Service Computer to deal with network balance and manage video sessions.

4.4 Performance Evaluation -- Static Scheduling

As we have explained in Chapter 3, Static Scheduling is based on the reservation table (or workload model) we get before the peak time. In order to distributed the work load evenly, we divided the network into 6 Units, and make an equal distribution of 250 videos to each Unit (recall that there are 500 videos and two layers of the network). Thus the simulator will need to simulate only one Unit. Since the Assistant Nodes are connected with low speed links, we make them as busy as possible during the 8-hour service time. The Front Nodes will do the rest of the work including performing network load balancing and handling the Video Commands of each session.

Static Scheduling in a Distributed System is more complicated than it was in the Hypercube System. The major difference is that the File Server and the Modems are built into the system, thereby fixing the position of the data. In this case, if we allocate a modem to only one customer during his video session, the node should provide the whole 2 hours of video data by itself, which translate to 120 / 5 = 24 service points. This can not be supplied in its entirely by a single Assistant Node, which only has 20 service points available from its 4 Hard Drive. So here we introduce a Dynamic Switch, which means that one customer can be served in different nodes at different time. For example, when a customer starts a session, he can be served by Node a, but after several minutes, the server can be switched to Node b. This is called a switch. Recall that each Hard Drive can only supply 5 service points, which corresponds to 5 × 5 = 25 minutes of video. In order to minimize the number of switches, we store only one video per Hard Drive at the Assistant Node. At the Front Node, we use 5 service points per video (see Figure 4.10). Therefore, we limit the number of switch times to 4 for a 2-hour video session.

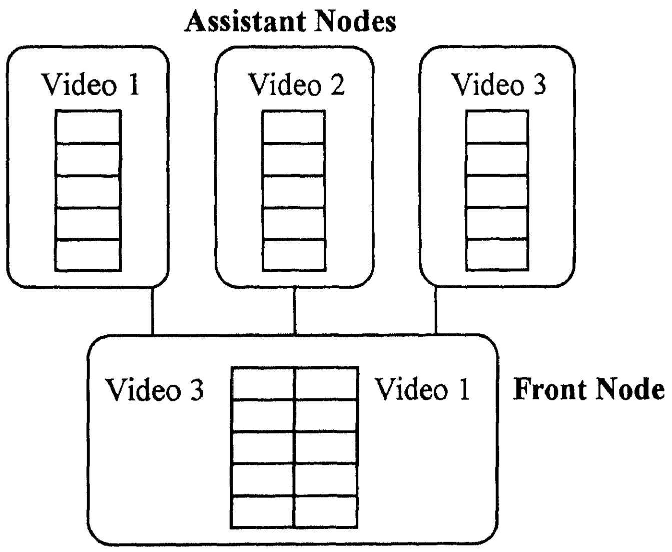
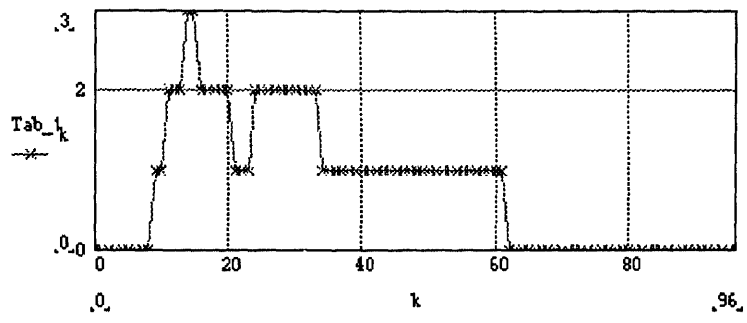
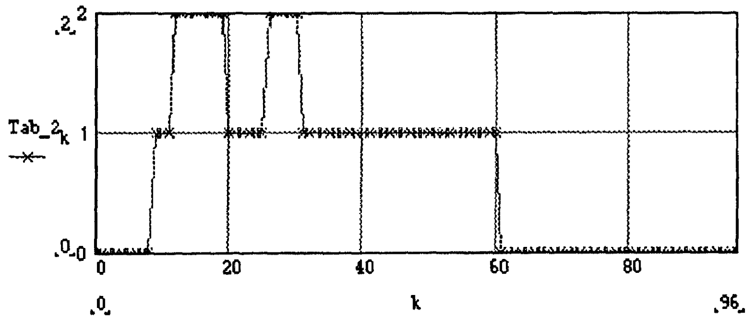


Figure 4.10 Video Storage on Each Node

We now discuss how the videos are allocated to the nodes of the network given the reservation table (i.e., Figure 2.4). Since the network is symmetric, it is sufficient to look at a Front Node and its three Assistant Nodes. For example, in Figure 4.2, Node a is the Front Node and Node 0, 7 and 8 are the Assistant Nodes. In Section 4.2, it was determined that each Front Node should be assigned 5 of the most requested videos. Let us call these 5 videos Video No. 1 through Video No. 5. Using the reservation table (Figure 2.4) and the distribution of video request frequencies, we can determine the reservation table for Video 1 through 5, as shown in Figure 4.11.

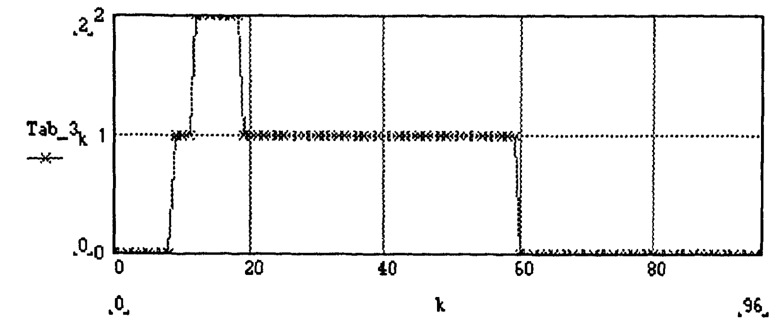


(a) No. 1 Video

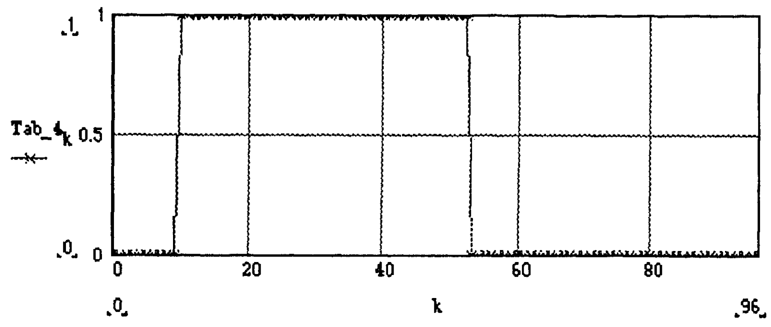


(b) No. 2 Video

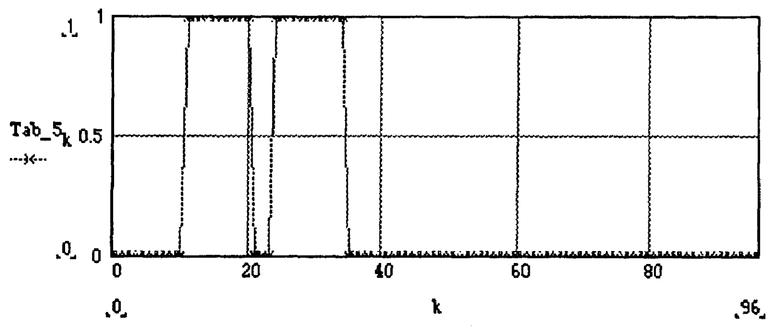
Figure 4.11 Reservation Table of the 5 Videos: Tab_i $_k$: number of sessions of video i to be started at time k.



(c) No. 3 Video



(d) No. 4 Video



(e) No. 5 Video

Figure 4.11 (continued)
Reservation Table of the 5 Videos:
Tab_i $_k$: number of sessions of video i to be started at time k.

When a session starts, normally it will stay in the system for about 2 hours before it ends, and that means a service point should be held for 2 hours before it can be reused by the system. The actual number of sessions in the system is the sum of newly started sessions and the number of current sessions in the system. From the reservation table of the 5 videos, we determine that the total number of sessions is maximum at k = 9, and is equal to 161. From our previous calculation, the number of sessions available from a Front Node is 76 (this is without duplication). From an Assistant Node, the number of sessions is 38. Therefore, the total number of session is 38 × 3 + 76 = 190, from which 190 - 161 = 29 can be used for other purposes.

Each Assistant Node can hold only 20 service points at most from the Hard Drive, and we would like to have it busy all the time. Among 5 videos, starting from time 9 (6:45 p.m.), for a duration of 24 service points (2 hours) , we get the tables shown in Table 4.1.

Table 4.1 Distribution Table of Peak Time Sessions

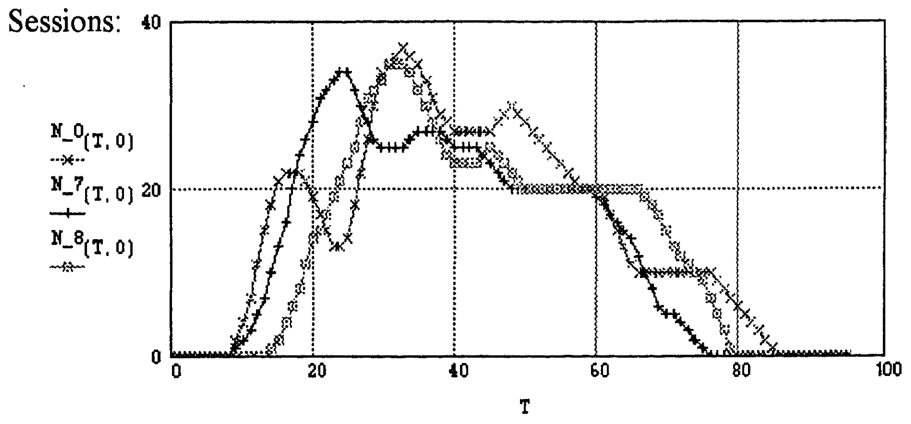
k	Vid-1	Vid-2	Vid-3	Vid-4	Vid-5
9 (6:45 - 7:10)	10	8	8	5	4
10 (7:10 - 7:35)	11	10	9	5	5
11 (7:35 - 8:00)	7	5	5	5	2
12 (8:00 - 8:25)	10	9	5	5	5
13 (8:25 - 8:50)	9	6	5	5	5

Here each column represents the number of newly started sessions for a given video at the specific period. For example, (0,0) represents that there are 10 newly started sessions of No. 1 video at time 6:45 p.m. - 7:10 p.m., while (2,2) represents that there are 5 newly started sessions of No. 3 video at time 7:35 p.m. - 8:00 p.m. We now try to assign as many sessions as possible to the Assistant Nodes. The remaining will be taken up by the Front Node. Each Assistant Node has 4 Hard Drives, and each Hard Drive can supply a 25-minute block of video. Table 4.2 shows one possible assignment of the 5 videos to the Hard Drives. From the table, each row represents a block of video (25 minutes long), and each column represents the time. So row (V-1, B-1) represents video -1 , block -1 (first 25 minutes). Therefore in the first 25 minutes, there are 10 newly started sessions, and in the next 25 minutes, there are 11 newly started sessions, etc. The table shows a mapping of these blocks to the Hard Drive, subject to the constraint that each Assistant Node does not exceed 38 sessions. For example, (V-1, B-1), (V-1, B-2), (V-1, B-4), (V-2, B-1) are assigned to Assistant Node 7. The total number of sessions is 36 maximum which is less than 38.

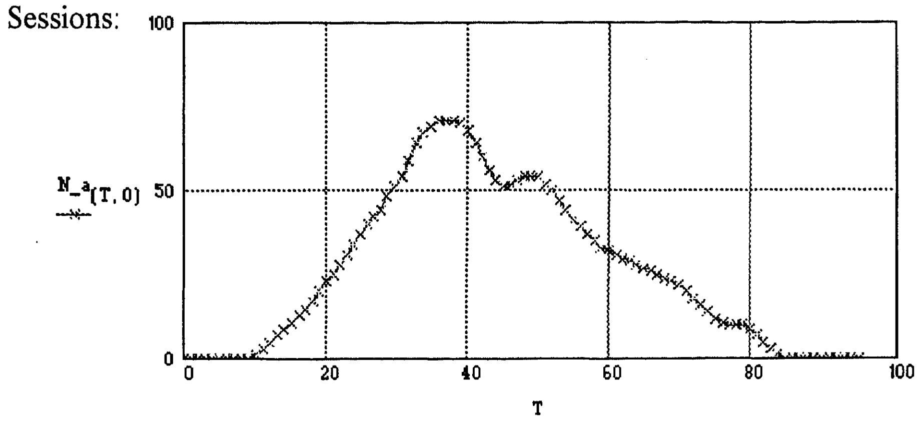
Figure 4.12 (a) and (b) show the number of sessions that can be supported by the Front Node and the Assistant Nodes. Figure 4.12 (c) shows the number of sessions that can be supported by a single unit, where a unit is a piece of the network as illustrated in Figure 4.3. As can be seen, the number of sessions per unit is 420, or a total of $420 \times 6 \times 2 = 5040$ sessions for the two-layer system. In other words, the system has to support up to 5040 simultaneous sessions in order to handle the workload depicted in Figure 2.4. However, the maximum capacity of the system is greater : $76 \times 24 + 38 \times 30 = 5928$ sessions. This number is better that that of the centralized system.

Table 4.2 Table of Mapping

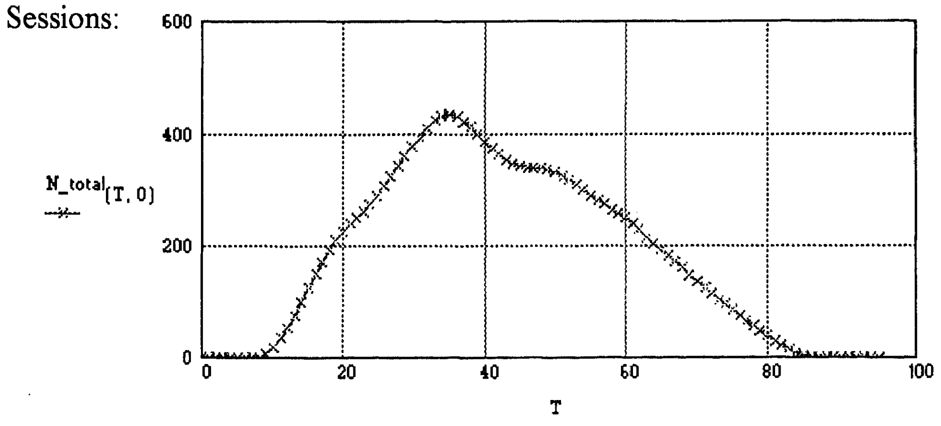
V-1, B-1:	10	11	7	[10]	9	0	0	0	0	N-7, HD-1
V-1, B-2:	0	10	11	[7]	10	9	0	0	0	N-7, HD-2
V-1, B-3:	0	0	10	11	7	[10]	9	0	0	N-0, HD-1
V-1, B-4:	0	0	0	[10]	11	7	10	9	0	N-7, HD-3
V-1, B-5:	0	0	0	0	10	[11]	7	10	9	N-0, HD-2
V-2, B-1:	8	10	5	[9]	6	0	0	0	0	N-7, HD-4
V-2, B-2:	0	8	10	5	9	[6]	0	0	0	N-0, HD-3
V-2, B-3:	0	0	8	10	5	[9]	6	0	0	N-0, HD-4
V-2, B-4:	0	0	0	8	10	5	9	6	0	
V-2, B-5:	0	0	0	0	[8]	10	5	9	6	N-8, HD-1
V-3, B-1:	8	9	5	5	5	0	0	0	0	
V-3, B-2:	0	8	9	5	5	5	0	0	0	
V-3, B-3:	0	0	8	9	[5]	5	5	0	0	N-8, HD-2
V-3, B-4:	0	0	0	8	[9]	5	5	5	0	N-8, HD-3
V-3, B-5:	0	0	0	0	[8]	9	5	5	5	N-8, HD-4
V-4, B-1:	5	5	5	5	5	0	0	0	0	
V-4, B-2:	0	5	5	5	5	5	0	0	0	
V-4, B-3:	0	0	5	5	5	5	5	0	0	
V-4, B-4:	0	0	0	5	5	5	5	5	0	
V-4, B-5:	0	0	0	0	5	5	5	5	5	
V-5, B-1:	4	5	2	5	5	0	0	0	0	
V-5, B-2:	0	4	5	2	5	5	0	0	0	
V-5, B-3:	0	0	4	5	2	5	5	0	0	
V-5, B-4:	0	0	0	4	5	2	5	5	0	
V-5, B-5:	0	0	0	0	4	5	2	5	5	
Total:				36	30	36				



(a) Assistant Nodes



(b) Front Node



(c) Unit

Figure 4.12 Performance of the Distributed System

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS FOR FURTHER STUDY

5.1 Performance Comparison

As determined by our performance analyses, the total number of sessions is 4992 for the Centralized System and 5928 for the Distributed System. The Distributed System is apparently better than the Centralized System.

The hybercube-based Centralized System has some advantages. It doesn't have a lot of external line connections because most of the system components are integrated into circuit boards. Consequently, it is easier to trouble shoot. Moreover, because the processors are tightly coupled, inter-processor communication is fast; and hence it is easier to perform load balancing.

The Distributed System has its own advantages. It is cheaper, as we shall soon show. This is very important for those companies who want to invest. Moreover, it is easy to configure, and can be expanded in an incremental fashion. (In the hypercube system, expansion is achievable only by doubling the sign of the system.) It is more robust than the Centralized System. In the Centralized System, a component failure would most likely crash the entire system. In the Distributed System, once the source of failure is identified (e.g., a Front Node), the network can be reconfigured to isolate or bypass the failed components. The system can still function, although at a reduced level of performance.

5.2 Cost Estimation

5.2.1 Distributed System

Based on the current advertisement, we now have a price list[5][6][7]:

• Pentium 60MHz mother board w/24 MB 70 ns DRAM & 256 K cache	$2600
• Intel 486 DX 2 - 66 w/8 MB 70 ns DRAM & 256 K cache	$1300
• 2 × 340MB Hard Driver w/1 VESA IDE controller	$ 600
• 2 × 2.4 GB HD w/1 PCI IDE controller	$4000
• 4 x speed CD-ROM w/6 CD changer	$1500
• 4 x speed CD-ROM w/18 CD changer	$2000
• FDDI Token Ring Card (100 Mbps)	$3000*
• EtherNet Card (10 Mbps)	$ 150

(* Price is estimated, no information available.)

Now we can estimate the total expense for the Distributed System:

a. Network Connections:

• 18 EtherNet with 6 nodes each. 1 FDDI Token Ring with 18 nodes. Total cost is:

$$(18 \times \$3000 + 18 \times 6 \times \$150) \times 2 = \$140400;$$

b. Node Price:

• Assistant Nodes:

$$(\$1300 + \$1050 + \$1200) \times 30 \times 2 = \$213000$$

• Front Nodes:

$$(\$2500 + \$4000 + \$2000 \times 2) \times 24 \times 2 = \$504000$$

Total:

$$\$140400 + \$213000 + \$504000 = \mathbf{\$857400}$$

It seems that we have spent a lot, but to see the price we use are from the retail price, we are sure that the total price won't exceed the amount above.

5.2.2 Centralized System

Now we conclude our Hypercube Centralized System. We conclude previous model and list those parameters below:

- 64 Processors;
- Each Processor with 16 MB of local memory;
- 4 Processors contained in a mother board, and connected with 1 FDDI Token Ring with 400 Mbps to mass storage;
- 4 Processors use another 1 FDDI Token Ring with 400 Mbps for output;

In order to compare the price, we assume that each processor w/16 MB of memory will cost the same as a Pentium Processor w/24 MB of memory which is \$2600.

Then to each unit, we assume 400 Mbps FDDI will cost 4 times the normal FDDI link (in fact, it is less than actual amount). And we further assume that 1 MB of HD will cost \$2 for HD and IDE controller, and each File Server will be controlled by a processor same to the unit processor. To provide 240 Service Point, we need $240 \times 5 \times 60 \times 1.5 / 8 = 13.2$ GB of Hard Driver capacity, and $32 / 9 = 4$ CD-ROM w/18 CD changer. Then each unit will cost:

$$\$2600 \times 4 \text{ (PEs)} + \$3000 \times 4 \times 2 \text{ (FDDI)} + \$2000 \times 4 \text{ (CD-ROMs)}$$
$$+ 13.2 \times 1024 \times \$2 \text{ (HD)} + \$2600 \text{ (File Server)} = \$72033$$

Thus total expense will be: $\$72033 \times 16 =$ **\$1152528**, but here we only use the price similar to the price with our system, generally with such a higher speed network, the expense should be higher than the price we estimated above.

5.2.3 Other Distributed Systems

It is hard to estimate the price and performance of other distributed systems. A basic estimation comes from a simplest model. Here only one FDDI link 100 Mbps link will be used. To provide the same sessions, we should have and 5928 / 78 = 76 of INTEL Pentium nodes.. Thus single unit cost will be:

$$\$2600[\text{mother board}] + \$4000[\text{5GB HD}] + \$2000[\text{ CD-ROM}]$$
$$+ \$3000[\text{ FDDI}] = \$11600$$

and total : $\$11600 \times 78 =$ **$904800** which is a little bit higher than the cost of our system. The disadvantage of such a system is that the average bandwidth that each node can get from the network is less than that of our system's.

5.3 Conclusions

From a cost/performance perspective, we conclude that a Distributed System is better than a Centralized System. Moreover, the Distributed System enjoys important advantages over the Centralized System, such as incremental scalability and robustness against failure. Finally, the Distributed System is based on currently available technology, which makes it more practical than the hypercube-based Centralized System, which requires more advanced technology.

5.4 Recommendations for Further Study

Although we have described the design of the Distributed System in some detail, further study is recommended, especially the specifics of the hardware and software design, network protocol design, and scheduling . Here we want to talk a little bit about dynamic scheduling.

Dynamic scheduling is needed to balance the load of each node. Recall that we assumed 90% of the customers will reserve their video requests in advance. Therefore, the system remaining 10%, which are "drop-in" customers. When a "drop-in" request comes in, the system will first try to find the location of the node holding the requested video, and then use the reservation table to predict the overall load of that node. This is because we should first make sure that we have enough resources for those reserved customers. If the node can serve the drop-in request, a session will be assigned. If not, the system should locate another lightly loaded node to take over some of the workload of the overloaded node, so that the drop-in request can be served. Clearly, this requires dynamic scheduling of the video streams, which is a complicated process. Whether this can be achieved in real-time and with little overhead is a question that remains to be studied.

APPENDIX A

Source Code List for Simulator on the Centralized System

1. Tab1.c:

```
/* ************************************************ */
/* Tab1.C:  Generate Table for Video Distribution in one layer  */
/* ************************************************ */

#include <stdio.h>
#include <stdlib.h>

main()
{
 int i, l;

 for (l=1;l<512;l=l+32)
 {
  for (i=0;i<16;i++)
   printf("%3d ", l+i);

  printf("\n");

  for (i=16;i>0;i--)
   printf("%3d ",l+15+i);

  printf("\n");
 };
 return 0;
};
```

2. Sim_sup.c:

```
/* ***************************************************** *
 *                                                       *
 *          Simulator on N-CUBE Super Computer           *
 *       ================================                *
 *                                                       *
 *                  By Ge Dai, NJIT,  1994.4             *
 *                                                       *
 * ***************************************************** */

#include <stdio.h>
#include <stdlib.h>

/* **************** *
 * Global Variables  *
 * **************** */

#define Sim              0     /* Simulate No. 0 Group              */
#define NNCUBE           64    /* Total PEs in a NCUBE Computer     */
#define NPEs             4     /* Number of PEs per group           */
#define NGRP             16    /* Number of Groups in system        */
#define NMaxSessions     10000 /* Maximum Sessions per PEs          */
#define NMaxSvcPoint     80    /* Maximum Service Points            */
#define NPackets         160   /* Maximum Packets per FDDI link     */
#define NVidLen          25    /* Length of a Video, 25 Svc Pt.     */
#define NVideo           32    /* Number of Videos per Group        */
#define TIME             96    /* Peak Time                         */
#define VIDEO            250   /* Number of video/2                 */

struct Node                    /* Node Parameter            */
{ int Sessions;                /* Number of Sessions        */
  int SvcPoint;                /* Number of Service Points  */
} N[NPEs][TIME];

int Pos,Dir;                   /* Timer                     */
int Group[NVideo][NGRP];       /* Video Group Table         */
float Vid[TIME];               /* Video Distribution Table  */
float Sel[VIDEO];              /* Video Selection Table     */


/* **************** *
 * Initialize        *
 * **************** */
```

```
void Init(void)
{
 FILE *fl;
 int  i,l;

 /* Load Tables */
 if ((fl=fopen("Video.DAT","r"))==NULL)
 { printf("\n\7Open Video.DAT Error!");
   exit(1);
 };

 for (i=0;i<TIME;i++)
 { if (fscanf(fl,"%f", &Vid[i])==EOF)
    { printf("\n\7Abnormal EOF.");
      exit(1);
    };
 };
 fclose(fl);

 if ((fl=fopen("Sele.DAT","r"))==NULL)
 { printf("\n\7Open Sele.DAT Error!");
   exit(1);
 };

 for (i=0;i<VIDEO;i++)
 { if (fscanf(fl,"%f", &Sel[i])==EOF)
    { printf("\n\7Abnormal EOF.");
      exit(1);
    };
 };
 fclose(fl);

 if ((fl=fopen("Tab1.DAT","r"))==NULL)
 { printf("\n\7Open Tab1.DAT Error!");
   exit(1);
 };

 for (i=0;i<NVideo;i++)
 { for (l=0;l<NGRP;l++)
    if (fscanf(fl,"%d", &Group[i][l])==EOF)
     { printf("\n\7Abnormal EOF.");
       exit(1);
     };
};
fclose(fl);


/* Clear Output Table */
for (i=0;i<NPEs;i++)
{ for (l=0;l<TIME;l++)
  N[i][l].Sessions=N[i][l].SvcPoint=0;
};

Pos=0;
Dir=1;
};

/* **************** *
 * Pks Distribution     *
 * **************** */
int Wh1(void)
{
 float ss;

 ss=random(100);

 if (ss<25) return 0;
 if (ss<50) return 1;
 if (ss<75) return 2;

 return 3;
};

int Wh2(void)
{
  Pos++;

  if (Pos==NPEs) Pos=0;

  return Pos;
};

/* **************** *
 * RunTime Simu.        *
 * **************** */
void RUN(int str)
{
```

```
int i,l,num,tmp,w,Time;

/* Run */
for (Time=0;Time<TIME;Time++)
{ for (i=0;i<NVideo;i++)
  { num=(int) (Vid[Time]*Sel[Group[i][Sim]/2]);
    if (str==1) w=Wh1(); else w=Wh2();
    for (l=0;l<NVidLen;l++)
     if ((num>0) && ((tmp=Time+l)<96))
      { N[w][tmp].Sessions+=num;
          N[w][tmp].SvcPoint++;
      };
  };
 };

 /* Check Valid */
 for (Time=0;Time<TIME;Time++)
  for (i=0;i<NPEs;i++)
   if ((N[i][Time].Sessions>NMaxSessions) ||
     (N[i][Time].SvcPoint>NMaxSvcPoint))
      printf("\n\7Overflow: N%d,T%d -- (%d,%d).", i, Time,N[i][Time].Sessions,
 N[i][Time].SvcPoint);


  printf("\n");

  /* Print */
  for (Time=0;Time<TIME;Time++)
  {
   tmp=num=0;
   for (i=0;i<NPEs;i++)
   { printf("%6d %3d  ", N[i][Time].Sessions, N[i][Time].SvcPoint);
     tmp+=N[i][Time].Sessions;
     num+=N[i][Time].SvcPoint;
   };
   printf("%6d %3d\n", tmp, num);
  };

  };

  /* **************** *
   * Main Procedure      *
   * **************** */
  main()
```

```
{
 int i;

 scanf("%d",&i);
 if (i!=2) i=1;
 Init();
 RUN(i);

 return 0;
};
```

APPENDIX B

Source Code List for Simulator on the Distributed System

1. Tab1.c:

```
/* ********************************************** */
/* Tab1.C:  Generate Table for Video Distribution in one layer  */
/* ********************************************** */

#include <stdio.h>
#include <stdlib.h>

main()
{
 int i, l;

 for (l=1;l<253;l=l+12)
 {
  for (i=0;i<6;i++)
   printf("%3d ", l+i);

  printf("\n");

  for (i=6;i>0;i--)
   printf("%3d ",l+5+i);

  printf("\n");
 };
 return 0;
};
```

2. Sch.c:

```
/* ***************************************************** *
 *                                                      *
 *                  Static Scheduling                   *
 *                  ================                    *
 *                                                      *
 *           By Ge Dai, NJIT, 1994.4                    *
 *                                                      *
 * ***************************************************** */

#include <stdio.h>
#include <stdlib.h>

#define total   36
#define matx    5
#define maty    5
#define buf     4
#define srv     4

int max,num,ord;
int Out[buf][srv], Ord[matx*maty];
int D[matx*maty], I[matx*maty];

void sort(int *wrk)
{
 int i,l,xx;

 for (i=0;i<num;i++)
  for (l=i+1;l<num;l++)
   if (wrk[l]>wrk[i])
   { xx=wrk[l];
     wrk[l]=wrk[i];
     wrk[i]=xx;
   };

 for (l=num; l<(matx*maty); l++)
  wrk[l]=0;

 };

 int ins(int *wrk)
 {
```

```
 int i,l,st;

 if (num>srv) return 1;

 st=0;
 for (i=0; i<ord; i++)
 {
  for (l=0; l<srv; l++)
   if (wrk[l]==Out[i][l]) st++;
  if (st==srv) break;
   else st=0;
 };

 if (st==0)
 {
  printf("\n");
  for (l=0; l<srv; l++)
  {
   Out[ord][l]=wrk[l];
   if (wrk[l]!=0) printf("%3d  ",wrk[l]);
  };
  if ((++ord)==buf) exit(0);
 };

 return 0;
};



int match1(int tot)
{
 int i,l,st;

 if (tot<3)
 { for (i=0;i<num;i++)
    Ord[i]=D[I[i]-1];
   sort(Ord);
   ins(Ord);
   I[--num]=0;
   return 0;
 };

 for (i=1;i<(matx*maty);i++)
 { st=0;
```

```
  for (l=0;l<num;l++)
   if (i==I[l]) st=1;
  if ((st==0) && (tot>=D[i-1]))
  { I[num++]=i;
    match1(tot-D[i-1]);
  }
 };

 if ((--num)==0) return 1;
 I[num]=0;
 if (max>tot) max=tot;
 return 0;
};

main()
{
 FILE *fl;
 char *fn;
 int  i,l;

 /* Open DAT1.DAT and Read */
 if ((fl=fopen("DAT1.dat","r"))==NULL)
  { printf("\n\7Error!\n");
    exit(1);
  };

 for (i=0;i<maty;i++)
  for (l=0;l<matx;l++)
  {
   if (fscanf(fl, "%d", &D[i*matx+l])==EOF)
    D[i*matx+l]=0;
  };

 fclose(fl);

 /* sort */
 num=matx*maty;
 sort(D);

 /* add up to see if result exits */
 max=0;
 for (i=0;i<srv;i++)
  max+=D[i];
 if (max <total)
```

```
{ for (i=0;i<srv;i++)
   printf("%3d ",D[i]);
  exit(0);
};

/* Initialize */
ord=0;
for (i=0; i<buf; i++)
 for (l=0; l<srv; l++)
  Out[i][l]=0;

/* match 1: total */
num=0;
max=total;
match1(total);

return 0;
};
```

3. Strip.c:

```
/* ***************************************************** *
 *                                                       *
 *                        Strip                          *
 *                   ================                    *
 *                                                       *
 *               By Ge Dai, NJIT, 1994.4                 *
 *                                                       *
 * ***************************************************** */

#include <stdio.h>
#include <stdlib.h>

main()
{
 FILE *FI, *F1, *F2, *F3, *F4;
 int  i,I;

 if ((FI=fopen("Dat2.DAT","r"))==NULL)
 { printf("\n\7Error!");
   exit(1);
 };

 F1=fopen("Dat21.DAT","w");
 for (i=0;i<4;i++)
 { fscanf(FI,"%d", &I);
   fprintf(F1,"%4d  ",I);
 };
 fclose(F1);

 F2=fopen("Dat22.DAT","w");
 for (i=0;i<4;i++)
 { fscanf(FI,"%d", &I);
   fprintf(F2,"%4d  ",I);
 };
 fclose(F2);

 F3=fopen("Dat23.DAT","w");
 for (i=0;i<4;i++)
 { fscanf(FI,"%d", &I);
   fprintf(F3,"%4d  ",I);
 };
 fclose(F3);

 F4=fopen("Dat24.DAT","w");
 for (i=0;i<4;i++)
 { fscanf(FI,"%d", &I);
   fprintf(F4,"%4d  ",I);
 }
 fclose(F4);

 return 0;
};
```

4. Filter.c:

```
/* ************************************************** *
 *                                                    *
 *                    Filter                          *
 *              ================                      *
 *                                                    *
 *            By Ge Dai, NJIT, 1994.4                 *
 *                                                    *
 * ************************************************** */

#include <stdio.h>
#include <stdlib.h>

#define matx 5
#define maty 5

int D[matx*maty];
int O[matx*maty];

main()
{
 FILE *fl,*fo;
 int  i,l,num;

 /* open file */
 fl=fopen("DAT1.DAT","r");
 for (i=0;i<maty;i++)
  for (l=0;l<matx;l++)
   fscanf(fl, "%d", &D[i*matx+l]);

 fclose(fl);

 fl=fopen("DAT2.DAT","r");
 fo=fopen("DAT3.DAT","w");


 while (1)
 {
  for (l=0;l<(matx*maty);l++)
   O[l]=D[l];

  for (i=0;i<4;i++)
   { if (fscanf(fl,"%d", &num)==EOF)
```

```
    { fclose(fl);
      fclose(fo);
      exit(0);
    };

    for(l=0;l<(matx*maty);l++)
     if (num==O[l])
     { O[l]=0;
          break;
     };
   };

  for (l=0;l<(matx*maty);l++)
   if (O[l]!=0) fprintf(fo,"%2d ", O[l]);
  fprintf(fo,"\n");
 };
};
```

4. Sim.c:

```
/* ***************************************************** *
 *                                                       *
 *            Simulator on Distributed System            *
 *            ===============================            *
 *                                                       *
 *              By  Ge Dai, NJIT, 1994.5                 *
 *                                                       *
 * ***************************************************** */

#include <stdio.h>
#include <stdlib.h>

/* **************** *
 * Global structure *
 * **************** */

#define TIME        96    /* Number of Service Points          */
#define VIDEO       250   /* Number of Total Videos            */

struct Node
{ int VID;                /* Video ID                          */
  int Type;               /* Source:    0 - HD,
                                        1 - CDx2,
                                        2 - CDx4,
                                        3 - Network            */
  int Offset;             /* Offset from the strat of video    */
} N[40];

int   N_par;              /* Number of Parameters              */
int   N_Ses;              /* Maximum number of sessions        */
int   N_Srv;              /* Maximum number of SrvPoint        */
int   N_CD;               /* Number of CD can access           */
int   Sessions[TIME];     /* Total sessions on the node        */
int   SvcPoint[TIME];     /* Total Service Points              */
int   CDROM[TIME];        /* Service Points from CDROM         */
float Load[TIME];         /* Total load                        */
float T_dist[TIME];       /* Video Distribution Table          */
float T_sele[VIDEO];      /* Video Selection Table             */

/* **************** *
 * Initial Data     *
 * **************** */
```

```
void LdTab(char *fname, int flen, float Tab[])
{
 FILE *FL;
 int  i;

 if ((FL=fopen(fname,"r"))==NULL)
 { printf("\n\7Open %s Error!\n",fname);
   exit (1);
 };

 for (i=0;i<flen;i++)
  if (fscanf(FL,"%f",&Tab[i])==EOF)
   { printf("\n\7Abnormal End of File!\n");
     exit(1);
   };

 fclose(FL);
};


void Init()
{
 int  i;
 FILE *FL;

 /* Read Maximum number of Sessions and Service Points */
 scanf("%d %d", &N_Ses, &N_Srv);

 /* Read Static Distribution Table */
 for (i=0;i<40;i++)
  N[i].VID=N[i].Type=N[i].Offset=0;

 for (i=0;i<40;i++)
 { scanf("%d", &N[i].VID);
   if (N[i].VID==-1)
    { N_par=i;
      break;
    } else
      scanf("%d %d", &N[i].Type, &N[i].Offset);
      if (N[i].Type==1) N_CD=2;
      if (N[i].Type==2) N_CD=9;
 };
```

```
/* Clean output tables */
for (i=0;i<TIME;i++)
 Sessions[i]=SvcPoint[i]=Load[i]=CDROM[i]=0;

/* Load Video Distribution Table */
LdTab("I:\Video.dat",TIME,T_dist);


/* Load Video Selection Table */
LdTab("I:\Sele.dat",VIDEO,T_sele);

};


void RunTime()
{
 int tmp,Ses;
 int time,tab,i;

 /* Run */
 for (time=0;time<TIME;time++)
 { for (tab=0;tab<N_par;tab++)
   if ((tmp=time+N[tab].Offset)<(TIME-5))
   { Ses=(int) (T_sele[N[tab].VID]*T_dist[time]+0.5);
    if (N[tab].Type==0)
    { for (i=0;i<5;i++)
        { Sessions[tmp+i]+=Ses;
         if (Ses>0) SvcPoint[tmp+i]+=1;
        };
    } else
    { for (i=0;i<24;i++)
        { if (((tmp+i)<TIME) && (Ses>0))
          { Sessions[tmp+i]+=Ses;
           CDROM[tmp+i]++;
          };
        };
    };
   };
 };

 /* Valid */
 for (time=0;time<TIME;time++)
 { Load[time]=Sessions[time]/(float) N_Ses;
```

```
  if ((Sessions[time]>N_Ses) || (SvcPoint[time]>N_Srv)
     || (CDROM[time]>N_CD))
   printf("\n\7Overload at : %i",time);
 };

 /* Print */
 for (time=0;time<TIME;time++)
  printf("%5i  %5i  %5i  %f\n", Sessions[time], SvcPoint[time], CDROM[time],
Load[time]);
};

main()
{
 /* Initialize parameters */
 Init();

 /* Run and check valid */
 RunTime();

 return 0;
};
```

REFERENCES

1 Didier Legall, "MPEG-A Video Compression Standard for Multimedia Applications", *Communications of the ACM*, vol. 34, No.4, April, 1991, p 47-58.

2 _____. May 1994, Advertisement. *Macworld*. CA: Macworld Communications, Inc.

3 Patel, Ketan, Brian C. Smith and Lawrence A. Rowe. 1993. "Performance of a Software MPEG Video Decoder", *ACM Multimedia '93 Conference*, CA:IEEE Computer Society Press, Los Alamitos, 1993, p. 46

4 _____. May 1994, Advertisement. *PC World*. CA: PC World Communications, Inc.

5 _____. May 1994, Advertisement. *Micro Center Catalog*. PA: Micro Center.

6 _____. April 1994, Technique Booklet. CA: Pioneer New Media Technologies, Inc.

7 _____. April 1994, Advertisement. *LYBEN Catalog*. MI: Lyben Computer Systems

8 Nash, Kim S., "NCube Piggybacks on Oracle Plans", *Computerworld*, vol.28, No.5 January 31, 1994, p. 33.