A Prototype VOD Server to Support Many Concurrent MPEG Streams Using a Novel Disk Scheduling Strategy

H-.J. Chen, R. Krishnan, G. Shekhtman and T.D.C. Little
Multimedia Communications Laboratory
Department of Electrical and Computer Engineering
Boston University, Boston, Massachusetts
{*tdcl*}@*bu.edu*

August 20, 1995

MCL Technical Report No. 08-20-1995

Abstract–Conventional video-on-demand servers concentrate on striping approaches to support a large number of concurrent streams. In this work, we have implemented a server prototype which supports a large number of sesssions from a single disk. This is achieved by a novel scheduling technique and the use of offline MPEG data reorganization based on a-priori knowledge of stream profiles. The retrieved streams are also delivered to clients over a network using a simple application specific protocol.

Keywords: Video-on-demand, scheduling, MPEG, distributed player

1 Introduction

Our project has the primary objective of building a VOD server prototype supporting several concurrent MPEG streams from a single disk drive. These streams would have to be delivered to workstations on the campus LAN where they can be decoded and played out. In particular, we wanted to make an evaluation of how well multiple streams are supported by the disk scheduling strategy discussed in [1–3] by the use of a working prototype.

A secondary objective is to make a modular testbed available that will enable us to evaluate various VOD scenarios and server subsystems.

Unlike conventional data types, video data has strict timing and large bandwidth requirements. Further the long and continuous playout characteristic of video coupled with characteristics of MPEG encoding allows us to gain from reorganization and placeent of data.

In this paper, the next section identifies the project components. The design of this VOD server prototype in Section 3. An architectural overview of the system is provided in Section 3.1. The detailed design of the scheduler and the distributed player is discussed in Sections 3.2 and 3.3.

The pseudocode for the sequencing protocol is also given in Section 3.3. The possible future enhancements to this player are considered in Section 4.

The problem is split into 2 major areas viz., (a) Storage and Retrieval and (b) Distributed Player.

2 Storage and Retrieval for VOD

This part of the project involves the development of a scheduler for retrieval of video data from the disk and perform necessary reorganization of the data before presentation to the client.

2.1 Disk Scheduler

The core of the project is the implementation of the novel scheduling policy discussed in the literature. In essence, the scheduler utilises the information available in the data stream to schedule subsequent requests for various streams sequentially in a round-robin fashion. If all requests cannot be met within the specified interval, frames are dropped from all sessions equally. This scheduler must schedule video data retrieval and delivery in real time besides being able to dynamically respond to client requested VCR actions.

2.2 Data Reorganization

The proposal expects to achieve scalability by dropping selected frames. This requires a re-organization of the data stream in the specified format. This reorganization is done offline through a program. However the data will have to be converted to the regular MPEG orgnization before sending over the network.

2.3 Networking for VOD

This part of the project involves the development of an application-layer protocol for real-time MPEG stream and session control. It also involves implementation of a distributed MPEG player application.

2.4 Session Control

Allow client station to create new sessions, request movies, perform VCR control actions and also stop a session.

2.5 Client Network Interface

Establish network connection with the server, communicate client requests, accept incoming MPEG frames and send it to the player.

2.6 Server Network Interface

Receive client requests (including initiation and termination) and communicate to the scheduler. Also provide a mechanism to the scheduler to despatch the retrieved video frames over the network to the client.

2.7 MPEG Player

Decode the MPEG stream received from the network and play it out using an X Windows application. In the project we have made use of the public domain software `mpeg_play` made available through the University of California, Berkeley without any modifications.

2.8 Abbreviations and Acronyms

```
VOD      - Video On Demand
VCR      - Video Cassette Recorder
MCL, BU - Multimedia Communications Laboratory, Boston University
WWW      - World Wide Web
MPEG-I   - Motion Picture Experts Group - I
UDP      - User Datagram Protocol
TCP      - Transmission Control Protocol
TCP/IP   - Transmission Control Protocol/Internet Protocol Suite
```

3 Design

3.1 Architectural Overview

An architectural overview of the system is depicted in Fig. 1. The design follows the client-server paradigm that lends itself to natural implementation in the Unix environment. The main client process and the MPEG Player are part of the client. The server processes consists of the session manager and the disk scheduler processes.

The disk scheduler is implemented as a set of two processes that handle the retrieval of data for all the sessions. For any of the sessions, each process reads data from a different file. Since requests from the client arrive asynchronously, a pipe is used between the two processes to resolve which of the two processes is to read the first file for the given session.

Due to inherent system limitations in the amount of shared memory available it was decided not to use shared memory for buffering the retrieved data. A mechanism had to be implemented whereby all the buffering would be done in the process heap space and let the scheduler directly invoke a function to write packets to the network.

This would have required a complex set of IPCs between the two scheduler processes. Each would have to convey to the other how much it has read so the other can seek forward. A further enhancement was to modify the storage format further by splitting each movie into two different files. This format is illustrated in Fig. 2.

The video data is sent over the network as UDP packets. Fragmentation and the information necessary for reassembly are handled by the "put_frame" function.

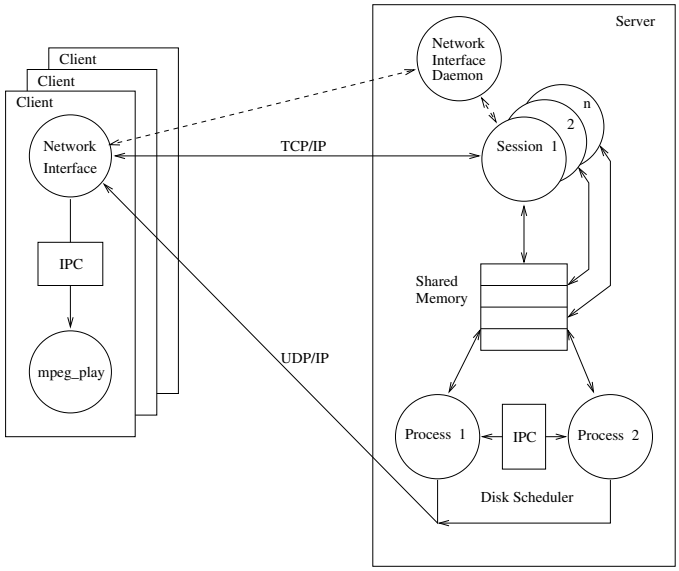


Figure 1: System Architecture

Each client process forks and execs "mpeg_play" to which it communicates through a pipe. It opens a TCP socket for session control and a UDP socket to receive video frames. It makes a connection request and subsequent VCR action requests over the TCP channel. The video frames received through UDP packets are stripped of protocol information, assembled and delivered to the player via the pipe. The player is an X-Windows application that decodes MPEG streams and plays out the movie in a window.

On receiving a connection request, the session manager at the server end forks a concurrent server for this connection. All client requests are logged by this new process on the shared memory. The port number of the UDP port in which the client receives video data is communicated to the server through the TCP channel and the server records this information also in the shared memory. This is stored in the network byte order.

Before each retrieval cycle, each scheduler process reads the shared memory to configure any new requests, handle VCR actions requested by the clients and drops out any sessions corresponding to clients that have requested termination.

Throughout the design, modularity has been emphasized. Apart from achieving the objective of providing us with a system to evaluate the disk scheduler proposal, the project has resulted in a testbed that can allow evaluation of other aspects of the VOD scenario. For instance, novel flow control mechanisms can be tested by simply replacing that section without having to build the entire system.

This underlying philosophy is reflected in the construction of a session database in shared memory. There is an implicit client-server type of interaction between the network module and the disk scheduler. The data structures used in this session database closely follow those which are exchanged with the client.

Since the network bandwidth and availability of workstations would limit the number of concurrent sessions, clients can also request dummy sessions which would merely generate workload for the scheduler. The retrieved data in these cases is not sent over the network.

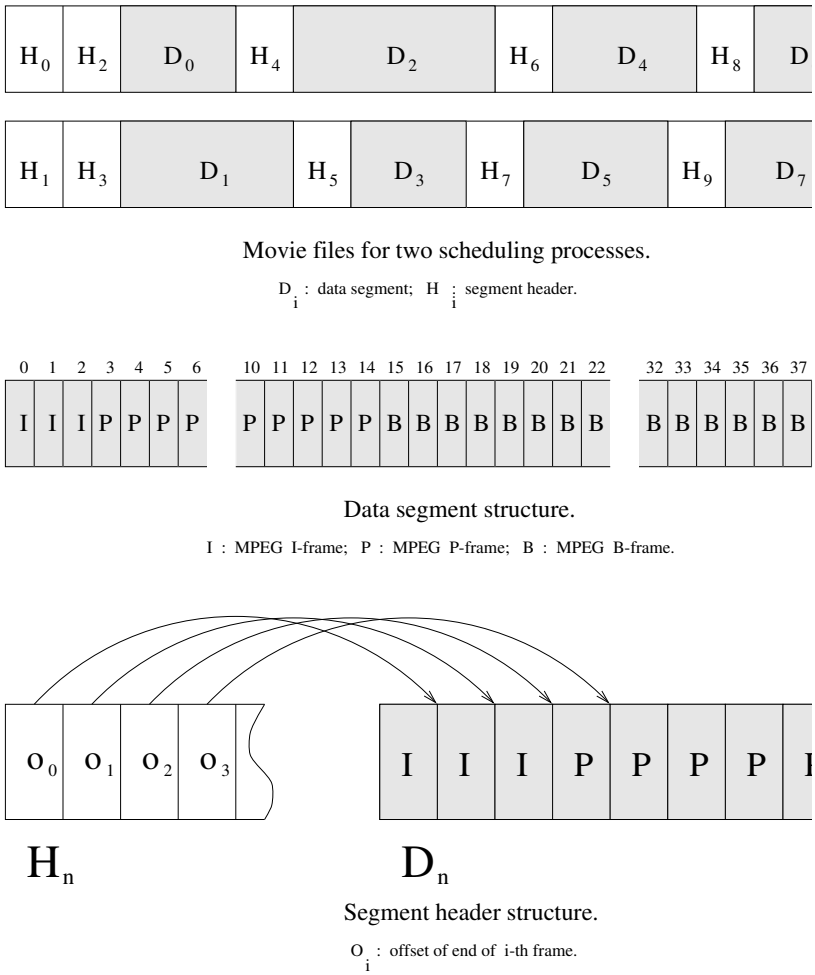


Figure 2: Files

3.2 Design of the Scheduler

Disk scheduler is designed as a pair of synchronized processes. As shown in Fig. 3, the scheduler can be in one of the two states. In one state process 1 sends data to clients and process 2 reads data from the disk, in another one process 1 reads next data segments from the disk and process 2 sends next data segments. The transition between states takes place in our case every 1.625 seconds (39 frames / 24 frames/sec).

There is no information interchange between two processes. IPC is used only for synchronization. Each process reads session information from the shared memory table at the beginning of each cycle.

Due to the fact that only one read request should be served at any moment, all sessions are handled sequentially by a single process in round-robin manner. The top-level flow chart of the scheduler is shown in Fig. 4.

The process checks shared memory table which is being updated by the network interface process. If any changes have been detected, one of the following session management functions is called: `init_sess()` to add new session to the list if new session request has been received, `remove_sess()` if a session has been terminated, `update_sess()` if session pa-

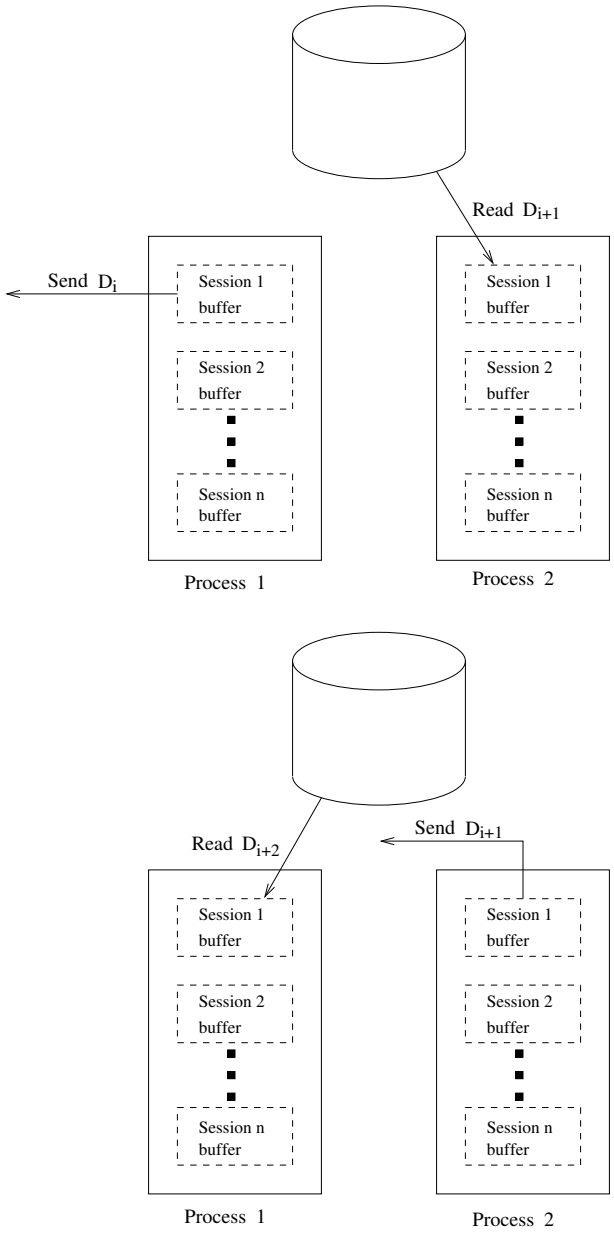


Figure 3: Two States of the Disk Scheduler

rameters (VCR action or its argument) have changed.

Then the process reads next portion (header and data segment) for each session sequentially from the corresponding movie file. The total amount of data for all sessions to be read is known in advance. If that total is greater than the precalculated threshold for the given disk, the fixed number of frames is dropped for each session.

After process completed the reading, it is synchronized with its partner by calling parent-child synchronization routines from [6].

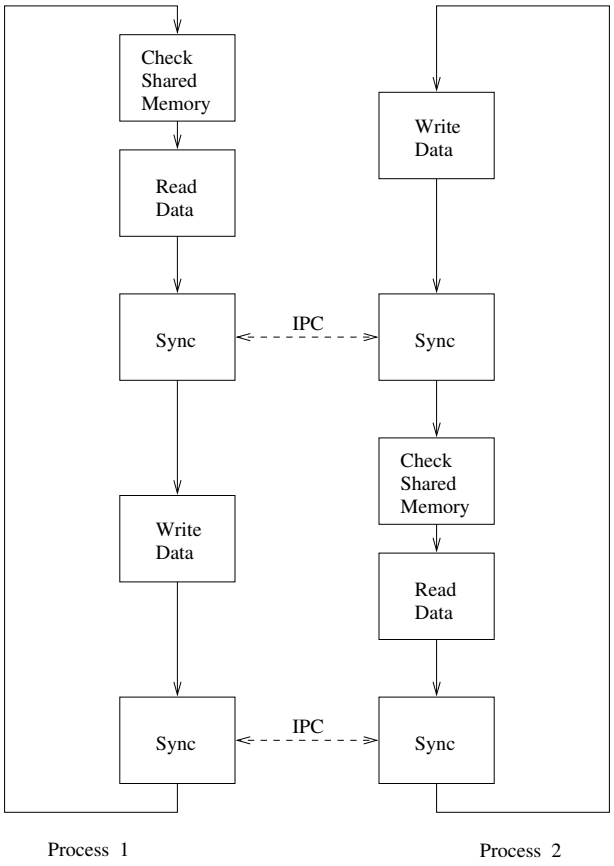


Figure 4: Top-Level flow chart of the two scheduling processes

After processes have been synchronized (the transition between two scheduler's states has been completed), the process starts writing frames periodically (every 1/24 sec) for all sessions sequentially. If session is dummy, data is written to a null device, otherwise it is sent to the client.

When all frames have been written, the processes are synchronized and state transition takes place again.

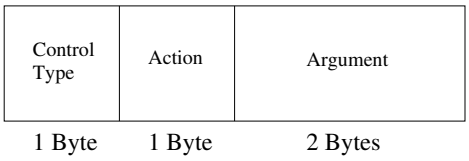
3.3 Design of the Distributed Player

The Session control channel is a TCP channel as described earlier. A TCP connection offers the reliability that is desired of the controlling channel and also vastly simplifies the design. A concurrent server process handles each session.

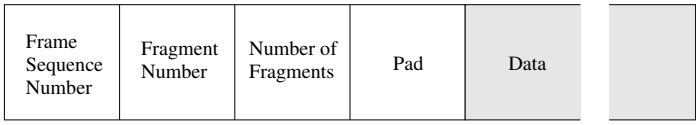
However video data is time sensitive and can tolerate losses better than delay. Therefore UDP is preferred for video delivery. The UDP port used by the client is communicated to the server via the TCP channel.

Session control actions are requested by the client through protocol packets. These packets contain 3 fields namely, Control type, the requested action and an argument. There are several other possible values that these fields can take. The protocol packet structure is illustrated in Fig. 5. For example, the client can register the UDP port number for video by sending a packet with :

```
Control Type = FLOCTRL
```



Session control protocol.



Video delivery protocol.

Figure 5: Data and Control Protocol Packets

```
Action        = UDPREG
Argument      = UDP Port in network byte order.
```

3.4 Sequencing and Frame Losses

UDP is a non-guaranteed datagram service. Further the underlying network may require that blocks of data (MPEG frames in our case) be broken down to smaller chunks (size of MTU) in order to transmit them as UDP datagrams. Due to these reasons, we need to take care of packet losses and out-of-sequence delivery of packets. As frame losses can be tolerated better than delays, we use a simple algorithm as outlined below.

The video delivery protocol is illustrated in Fig. 3. As stated earlier, the server needs to communicate any frame fragmentations to the client for proper reassembly. This is done by including sequencing information, a cyclic frame sequence and a natural fragment sequence, as a strap-on header with each UDP packet.

Algorithm Flush_outdated

```
Initialize expected frame number;
Initialize expected fragment to 0;
Create buffer of max frame size;
while TRUE
    receive packet;
    if(older frame)
      flush packet;
      continue;
    end if
    if(later frame than expected frame)
      flush packet;
      reset buffer;
      set expected frame number to current;
```

```
        set expected fragment number to zero;
    end if
    if(wrong fragment)
        flush packet;
        reset buffer;
        increment expected frame number;
        return empty frame;
    end if
    if(expected fragment)
        copy fragment to buffer;
        if (last fragment)
            increment expected frame number;
            set expected fragment number to zero;
            return frame;
        else
            increment expected fragment number;
            continue;
        end if
    end if
end while
```

3.5 Deviations from the Original Proposal

There have been some deviations and simplifications from the original proposal. These do not significantly impact the applicability of the proposal and are intended only to simplify implementation in the given environment. We list the most important ones here.

The file reorganization format was modified from a single file to two files to better suit our implementation architecture. The structure of the files are the same as proposed in the literature but the information for alternate intervals are written in different files.

A standard virgin Unix file system was used. The file system was not modified to support writes of very long blocks of data contiguously.

The rate at which data is sent over the network is only 9 frames/second. This is due to the limitations of using a software decoder.

Scaling by dropping of frames is done in a coarse-grained fashion. Whenever the next request exceeds the available bandwidth, all sessions drop a pre-defined number of frames which may be more than the exact number of frames that are required to be dropped to meet the bandwidth criterion.

3.6 System Scalability

Only two processes are required to schedule the reads of any single disk. The economy that such a scheme allows us to support a very large number of sessions with a small number of processes. There will be one set of scheduler processes for each disk and thus we can scale the system to support several disks at the same time.

However since each control channel locks up one file descriptor and there is a limitation on the total number of open files that a process can have, all sessions cannot be handled by the same

network server.

It is of course possible to use a multiplexing scheme with several control channels handled by ieach network server process. This scheme was abandoned in favor of concurrent servers in the interests of simplicity.

Otherwise the system has no other logical limitations and we can scale as much as the physical limitations of the hardware would permit.

3.7 Hardware and Software Requirements

Hardware:

The Server is a Pentium (P-90) based PC with a Ethernet card and a SCSI disk drive for the video data.

The client workstation can be any Unix workstation running X-Windows.

Software:

Solaris 2.4 for the PC and Pro Works Software Development System for the Server.

For the client any generic Unix platform with an ANSI C Compiler and support for System V IPC and Berkeley Sockets is required.

4 Future Enhancements

4.1 Graphical User Interface

Currently the client has only a command line interface. Future enhancements could include a graphical interface to support the VCR controls more effectively.

4.2 Fast Forward and Pause Controls

Though there is embedded functionality for the fast forward and pause controls, these are not operational. A different data organization would be necessary to make these operational.

4.3 Buffering and Flow Control to smooth the Network Playout

Currently the client end of the distributed player does not buffer any frames. It relies on the smoothing achieved through the isochronous nature of the source. However, occasional frame losses and the time non-determinism of `mpeg_play` make the playout jerky at times.

This can be improved by the use of buffering at the client end and the use of a flow control mechanism between the server and client.

4.4 Rewind Controls

The provision of rewind controls is more difficult under the proposed disk scheduling scheme. Variations of the scheme that would enable us to provide rewind can be explored.

4.5 Porting to other OSs

The project has been executed in C under Unix with System V IPC and Berkeley Sockets. It can therefore, easily be ported to a variety of platforms.

4.6 File System Enhancements

A new system call that can directly write long consecutive blocks of data can significantly enhance performance and will enable more sessions to be supported off the same disk. The VOD server can then read the whole block of video data from the disk without experiencing seek latencies.

4.7 Embed in a WWW Application

Currently WWW applications only partially support video by first downloading the entire video clip to the client. The client then launches a player and the playout is done from local disk. This has numerous disadvantages :

1. Large disk space requirements at the client.
2. Very large latency before playout.
3. This approach limits one to using small clips.
4. User has to download the entire clip.

This project can be embedded into a WWW application to enhance video support.

5 Conclusion

A distributed MPEG Player and the Disk Scheduler were constructed and were found to perform satisfactorily. We were able to maintain 5 video sessions delivered to workstations on the same subnet and 10 other dummy sessions (to create the workload). The sessions supported MPEG "quarter screen" video.

The real sessions were operated at only 9 frames/sec due to the limitations of the software decoder. Considerable performance gains would be made by the use of hardwae decoders in the client.

References

[1] H-. J. Chen, T. D. C. Little, D. Venkatesh, "A Storage and Retrieval Technique for the Provision of Scalable Delivery of MPEG-Encoded Video," *Journal of Parallel and Distributed Computing (Special Issue on Multimedia Processing and Technology),* Vol. 30, No. 2, November 1995, pp. 180-189.

[2] H-. J. Chen, A. Krishnamurthy, T. D. C. Little, D. Venkatesh, "A Scalable Video-On-Demand Service for the Provision of VCR-Like Functions," *Proc. 2nd Intl. Conf. on Multimedia Computing and Systems, (ICMCS'95),* Wahington DC, May 1995, pp. 65-72.

[3] H-. J. Chen, "A Disk Scheduling Scheme and MPEG Data Layout Policy for Interactive Access from a Single Disk Storage Device," *Ph. D. Dissertation,* Dept. of Electrical, Computer and Systems Engineering, Boston University, August 24, 1995.

[4] M. J. Bach, *The Design of the Unix Operating System,* Prentice Hall Inc., 1986.

[5] W. R. Stevens, *Unix Network Programming,* Prentice Hall Inc., 1990.

[6] W. R. Stevens, *Advanced Programming in the Unix Environment,* Prentice Hall Inc., 1992.