RC 20059 (05/10/95)
Computer Science

IBM Research Report

A High Performance Multimedia Server For Broadband Network Enviromment

Manoj Kumar, Jack L. Kouloheris, and Mary J. McHugh
IBM Research Division
T.J. Watson Research Center
Yorktown Heights, NY 10598

Sneha Kasera
Computer Science Department
University of Massachusetts
Amherst, MA 01003

IBM Research Division
Almaden . T.J. Watson . Tokyo . Zurich

ABSTRACT

We present a multimedia server comprising of multiple stream controllers connected to a host (work station) I/O bus. Each stream controller manages an array of disks in which multimedia data is stored in format of network packets. When the host instructs the stream controller to deliver a multimedia file to a client, the stream controller retrieves the network packets of that file, completes the missing header/trailer fields, and forwards these packets to the network interface directly, avoiding any further involvement of the host.

To avoid interference on the disks, data is interleaved across all disks connected to a stream controller in fixed play back time units. This helps reduce the jitter in the response time of the disks, and therefore, the size of the buffers needed to maintain interruption free delivery. Metadata is stored with the network packets of a video/multimedia file to enable the stream controller to autonomously fetch a complete multimedia/video file without host intervention.

In a departure from traditional RAID, the stream controller simultaneously issues a set of read commands periodically, one for each active multimedia stream. Each read command retrieves a full interleave unit from a single disk, and the set of simultaneously issued read commands are distributed across all disks.

1. Introduction

It is widely believed that the advances in technology will enable a wide variety of residential and commercial interactive multimedia services. Advances in VLSI technology have helped bring down the cost of compression/decompression hardware and enabled technologies like ADSL (Asymmetric Digital Subscriber Loop), making it possible to deliver motion video over the telephone network. Similarly, the advances in fiber optic transmission technology and its declining costs have enabled the upgrades in the cable TVs trunk and feeder systems which increase the bandwidth of the network sufficiently to provide each active subscriber his dedicated channel to the head-end for receiving compressed digital video. Personal computers and set top boxes enable networked multimedia applications, taking advantage of the low cost video compression/decompression hardware.

While the end user (client) systems and the network infrastructure is evolving rapidly to meet the requirements of interactive multimedia services, the servers are not. The current choice of servers for interactive multimedia services continues to be off-the-shelf mainframes or work-station based parallel/clustered computing systems. Their system architecture, hardware organization, operating systems, and I/O subsystems are ill suited for delivery of multimedia data.

In mainframes or work station based parallel or clustered systems multimedia data is stored as a standard operating system file on an array of magnetic disks. From there it is read into some processor's data memory where it is processed through a network protocol prior to its transmission over a network interface. This unnecessarily slows down the transmission of multimedia data from the server to the clients because to deliver the same video file to different clients, the same programs are executed repeatedly with essentially the same input data to retrieve the data from the disk and process it through the network protocol stack. This processing, which limits the number of video streams that can be delivered simultaneously, is mostly redundant, and can be eliminated to improve the cost-performance of video servers.

The hardware of the mainframes and work stations is optimized for processing intensive applications, with very limited emphasis on moving data efficiently between the network interfaces and storage devices, which is the primary requirement for a video server. For example, the bandwidth from the memory to cache in an RS/6000 workstation is 400 MBytes/sec. while that from storage or network devices to the system is only 80 MBytes/sec. The floating point unit adds to the cost of the system without providing any benefit to the delivery of video/multimedia data. The caches are too small for capturing any locality in the accesses to multimedia data.

Similarly, the operating system of mainframes or parallel computer based servers is optimized to maximize the utilization of the CPU, and to maximize the system throughput in a time sharing environment. The response time for the same operating system service can vary significantly between requests. Therefore, large buffers in system memory are needed for multimedia data being retrieved from secondary storage.

The above mentioned limitations in using general purpose computers as video servers force the price/performance of such servers to be much higher than that of a system designed optimally for delivery of video. The operating system drawbacks have been addressed by fine tuning the operating system services. Dan and Sitaram have proposed innovative methods for optimizing the use of the stream buffers in the system memory of a general purpose computer being used as a video server [4,5]. Haskin proposes increasing the block size in the file system to improve performance [7]. Rangan and Vin propose innovative techniques for placement of data on disks [10,13]. Several researchers have suggested improvements in scheduling disk read commands [8,11,14]. Tobagi et.al. propose a real time kernel based system in which a periodic process schedules the retrieval of multimedia data from a disk array [12].

Budhikot et.al. propose a system for directly delivering multimedia data from an array of disks to an ATM network [1]. But as is the case with RAID-II [6], the host is still responsible for initiating every data transfer from the disk. Therefore, the scalability of this approach is limited by the host processing capability. Chang [2] and Lougher [9] propose striping multimedia data across an array of disks in units of fixed play back time to control the interference on disks.

In this paper we describe the design of a scalable multimedia server which we have built in our laboratory, and is operational. It solves the above mentioned problems by off-loading the delivery of continuous media data from the host processor of a traditional server to a stream control subsystem, consisting of several stream controllers. The stream controller is designed to efficiently move data from a disk array to a network interface. Figure 1 presents a high level view of the video/multimedia server described in this paper. Each stream controller controls an array of disks, dedicated for storing continuous media data.

To support a large number of streams from a stream controller, protocol processing overhead must be minimized. To accomplish this data is stored on the disk array in the format of network packets. All the precomputable fields in the header/trailer of the packet are precomputed, and the remaining fields are calculated in the stream controller prior to the transmission of the packet. This prepacketized data is stored in the disks with link fields to allow the stream controller to access all data of a video/multimedia stream autonomously from a given starting address. Thus, the host is involved only in initiating the transfer of a multimedia stream. Using the metadata embedded in the stream, the stream controller autonomously fetches the successive blocks of the stream and transfers them over the network at appropriate times, without further involvement of the host processor. In contrast, in the schemes in [1,6] the host is continuously involved. The stream controller accepts commands from the host which are similar to the VCR commands like PLAY, PAUSE, FAST FORWARD, and REWIND etc..

Each stream controller contains a microprocessor based subsystem running a real time kernel. The stream controller periodically issues read requests to the disks to retrieve the data needed by each active video stream. The network protocol processing needed on the data read from the disks is carried out in the stream controller, and the network packets thus generated are then transferred directly to the network interface from the stream buffers.

To minimize interference on the disks, data is interleaved across all disks of a stream controller. Furthermore, the unit for interleaving data, accessing data, and for storage allocation, is a group of blocks (GOB). For efficient utilization of disk bandwidth, the GOB size is chosen to be more than one track of data, 64 Kbytes in todays technology. This makes the GOB too large to be transmitted to a client over a broadband network in a single burst. Therefore, it is subdivided into packet clusters which are transmitted in bursts at regular intervals. The playback duration of compressed continuous media data stored in a GOB is fixed for a stream controller, which results in variable GOB sizes if video is compressed at variable bit rate, as in [2,9]. By controlling the interference on the disks, and using a real time kernel to consistently provide fast

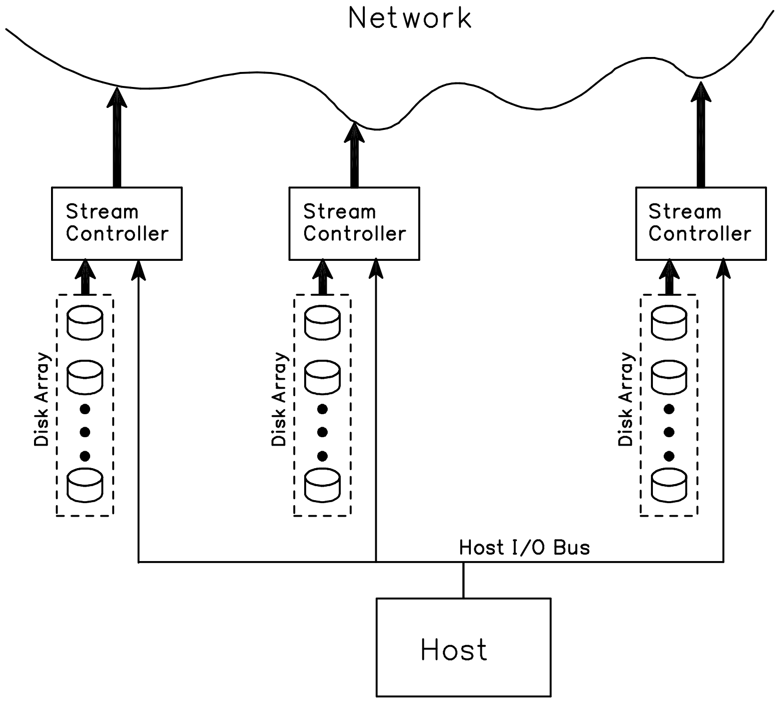

Figure 1: Proposed multimedia/video server system. Thick arrows indicate the flow of video/audio data.

response to OS service requests, we minimize the size of the buffers required for each multimedia stream.

In the next section we explain the stream controller design in further detail. Then we describe the format for storing multimedia/video data on the disk array managed by the stream controller. In section 4 we describe the implementation of the stream controller prototype. In section 5 we describe the programs for delivering continuous media streams autonomously from the stream controller. In section 6 we discuss the current status of the server prototype and some performance measurements. Finally, we conclude by discussing the possible directions for future research and summarizing the results of this paper in the next two sections.

2. The stream controller subsystem

The stream controller is designed for delivering a large number of continuous media streams from an array of disks to network interface cards simultaneously. The design is optimized for handling a large number of low bandwidth I/O transfers, the operating conditions in a video server. In contrast, traditional RAID systems are optimized to support a few high bandwidth streams.

The design of the stream controller is shown in Figure 2. At the core of the stream controller is the stream buffer memory constructed from semiconductor storage, and organized as a two ported memory. One port is used to receive data from the disk, and labeled in Figure 3 as the

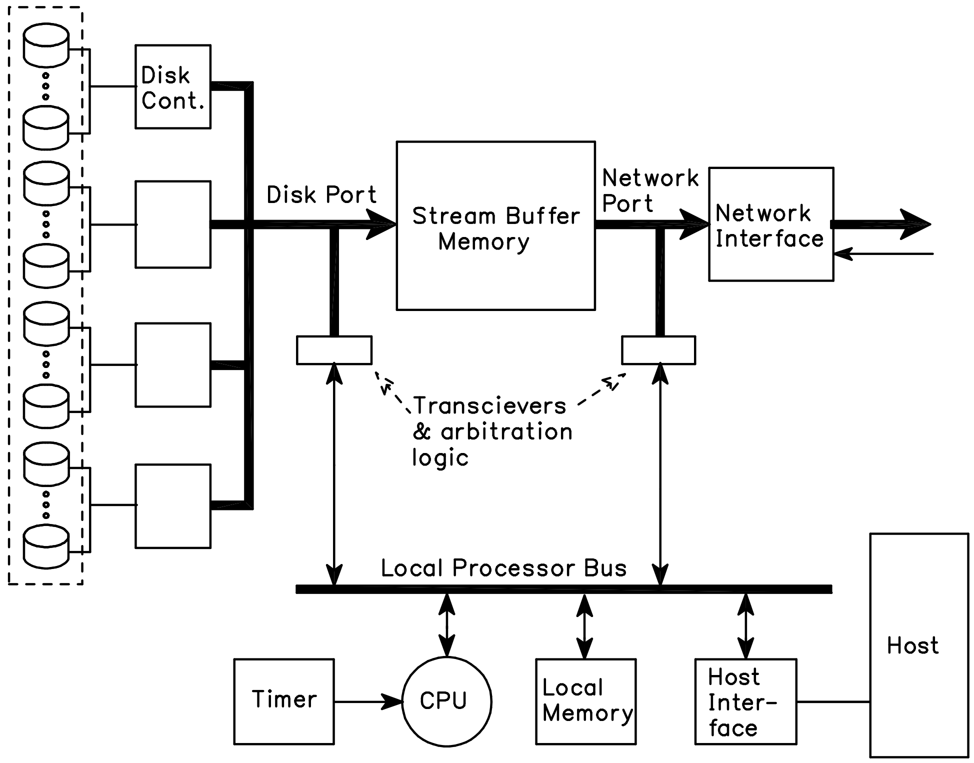

Figure 2: Block diagram of the stream controller.

disk port. The other port is used predominantly to read network packets for delivery to the network, and labeled as the network port. The stream buffer memory is partitioned into multiple stream buffers. Two stream buffers are allotted to each active video stream. The disk port of the stream buffer memory interfaces to multiple disk controllers, each connected to multiple disks. The network port of the stream buffer memory connects to the network interface logic which handles the broadband network connection. A real time control processor, typically an off the shelf microprocessor, is connected to its local memory through the local processor bus. The host interface in the stream controller also connects to the local processor bus, and allows transfer of data between the host and the local memory, and between the host and stream buffers. The transceivers and arbitration logic allow the control processor to access the stream buffers, the control registers in the disk controllers, and the control registers and storage in the network interface. It also allows the network interface logic and the disk controllers to interrupt the control processor and access its local memory.

The control processor in the stream controller sets up the disk controllers to transfer pre-packetized video data from the disks attached to them to the stream buffers. It writes a command list of read commands for each disk controller in an area of the stream buffer memory reserved for the purpose, and passes the pointer to the command list to the disk controller. The disk controllers indicate the completion of the read commands by interrupting the control processor. Alternatively, the command list could also be stored in the local memory.

The missing fields in the network header and/or trailer are computed and updated in the stream buffer by the control processor, and the completed packet is forwarded to the network interface logic. The network interface logic also receives a command list from the control processor. Each command in the command list specifies the address in the stream buffer of a network packet to be transmitted to the network, the size of the packet, and optionally a header that should be appended to the packet. Similar to the command lists of the disk controller, the command lists are also written by the control processor into either the stream buffer or the storage in the network interface, and the starting address and size of the command list are written into the control memory of the network interface logic. To serve 250 MPEG-1 video streams the network connection will preferably be an OC-12 (Optical Carrier level 12) or four OC-3 connections.

3. Optimized format for storing video/multimedia data on disks

The compressed multimedia/video data is stored in the disk array in form of network packets. This greatly reduces the CPU requirements for protocol processing carried out in the stream controller. In our prototype, the compressed video data is stored in the form of IP/UDP packets. The IP and UDP headers (except for the destination address and port numbers) are precomputed at the time video data is loaded in the disk array. When the packets are scheduled to be sent out to the network interface, the destination address and port numbers are filled in and the checksum(s) modified as required. Note that much less CPU time is required to do this than to compute the entire header and checksum from scratch. This technique is equally applicable to ATM (Asynchronous Transfer Mode) networks. For example, the entire ATM AAL5 (ATM Adaptation Layer 5) convergence sublayer is precomputable and and can be precomputed at the time video is stored on the disk.

The size of the network packets stored on the disks is typically in the range of 512 bytes to 4 Kbytes. Reading one packet at a time from the disk will result in inefficient use of the disk bandwidth. Therefore, several network packets are combined into a single disk access unit of size 64 Kbytes to 256 Kbytes. We use the term Group Of Blocks (GOB) for this access unit. A GOB is also the basic unit of storage allocation. The format of a GOB is shown in Figure 3.

The compressed video data stored in a GOB has a predetermined playback time, fixed for all GOBs of a stream controller. For video/multimedia data compressed at fixed bit rate, all GOBs in a stream controller will be of the same size. However, if video/multimedia data is compressed at variable bit rate, as was the case in our prototype, the GOBs have varying number of disk blocks.

The storage for multimedia/video data is allocated in chunks of several hundred Megabytes, and partitioned into GOBs when the data being stored there is pre-analyzed and partitioned into chunks having the fixed playback time used by the stream controller. Each GOB is referenced by a GOB pointer comprising of the triplet <*disk number*, *starting block on the disk*, *number of blocks in GOB*>.

While the disk controller accesses data from the disk one GOB at a time, a GOB (64 Kbytes to 256 KByte) is too large to be sent to a client in a single burst. So the GOB is further divided into *n* packet clusters of equal playback duration, and therefore, of possibly different sizes.

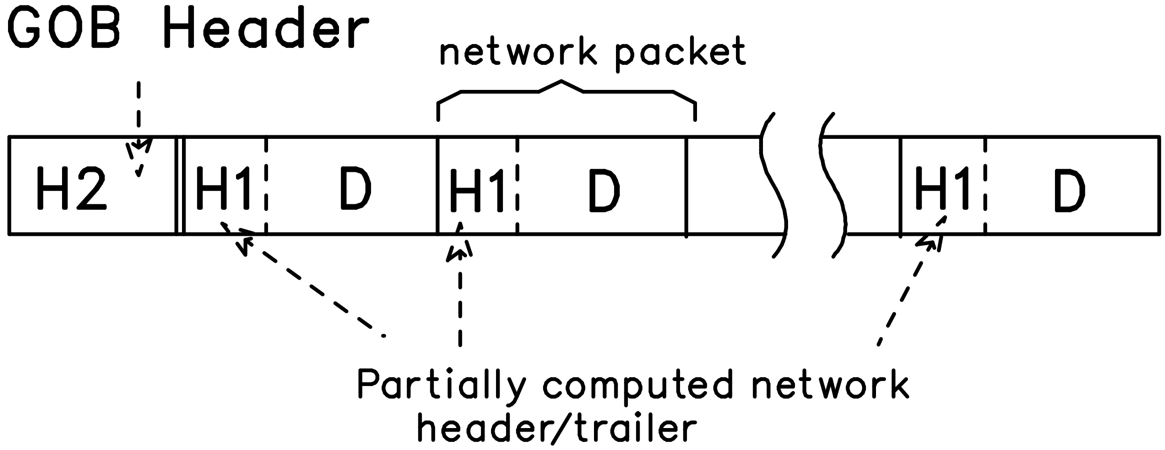

Figure 3: The internal structure and format of a GOB (Group Of Blocks), the unit of interleaving video data on disks.

While each packet in a GOB has its own network header (and possibly a trailer), the entire GOB has a GOB header as illustrated in Figure 3. The GOB header enables the stream controller to autonomously access consecutive GOBs of a multimedia stream. Thus, a GOB header necessarily includes the GOB pointer to the next GOB of the same multimedia stream. To check integrity of the stored video data we also included in the GOB header the pointer to the preceding GOB of the same video stream. If fault tolerance is implemented by storing a parity GOB for a group of data GOBs, then the GOB header includes pointers to all GOBs in the parity group of the next GOB. Finally, the GOB header also contains the pointers to the packet clusters.

4. Stream controller hardware

The stream controller in our first prototype is a set of three Microchannel adapter cards which plug into an RS/6000 work station as shown in Figure 4. One card has the network interface logic, the second has the control processor and its local memory, and the third has the stream buffer memory and 4 SCSI controllers that provide four fast and wide SCSI buses, each having a peak transfer rate of 20 MBytes/sec. At the time of our first prototype's implementation, interfaces for ATM networks were not available, and therefore we used the interface for the ORBIT network. ORBIT is an optical network based on buffer insertion ring approach, and supports speeds up to 640 Mbits/sec [3]. A separate network interface also gives us the flexibility to change the network interfaces to allow our server to be used in different networks. ORBIT software available for workstations supports the Internet protocol suite. The control processor is an off the shelf ARTIC 960 card running a real time OS kernel.

The stream controller is capable of serving 250 MPEG-1 (motion picture experts group) digital compressed video streams. This requires video to be delivered at a sustained bandwidth of 48 MBytes/sec. Therefore, each SCSI bus has to sustain a transfer rate of 12 MBytes/sec., which is achieved by connecting 8 disks on each fast and wide SCSI bus. The disks are 3.5" SCSI disks capable of storing 2 GBytes of data. In our prototype, the GOB play back time is

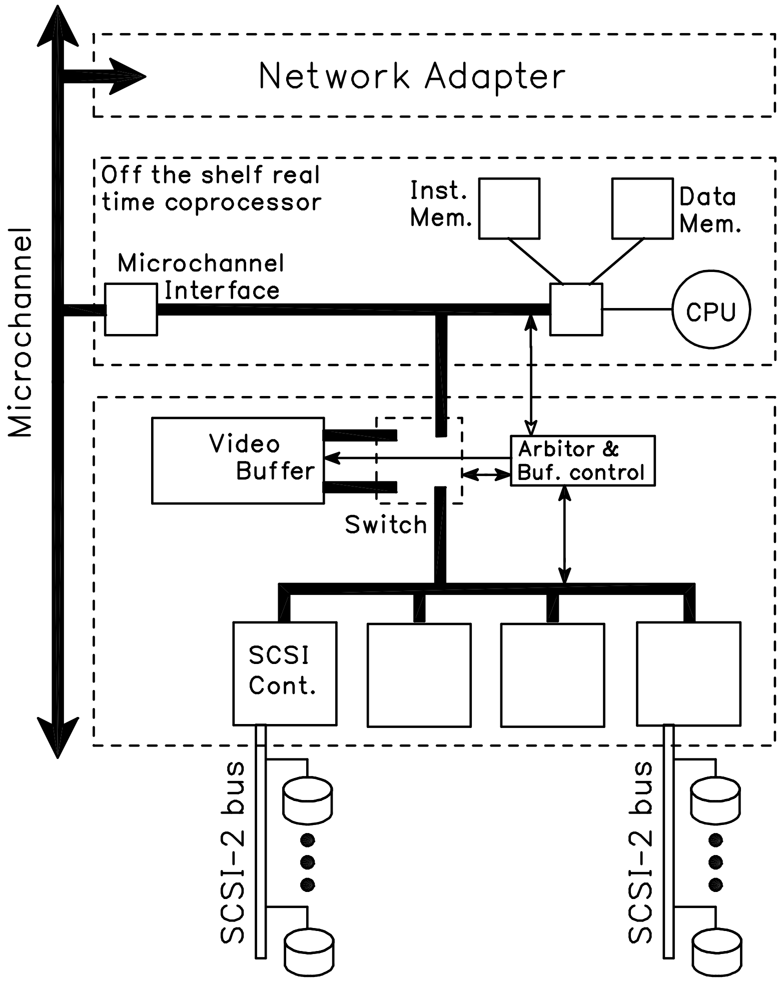

Figure 4: First prototype of the stream controller.

200 milliseconds, which corresponds to approximately 64 Kbytes, roughly the size of a track on modern magnetic disks. At that transfer size, these disks easily sustain the required bandwidth of 1.5 MBytes/sec., especially since the queued read requests are reordered automatically in the disk to minimize seek latency.

The control processor sets up the SCSI controllers by means of a script written into the stream buffer memory. This script to a SCSI controller comprises of multiple read commands for each of the several disks connected to it, and each read command to a disk specifies the transfer of a GOB of a video/multimedia stream. Then the control processor gives each SCSI controller the address of its script in stream buffer by writing this address along with other control information in the SCSI controllers control registers.

Similarly, to transfer network packets from the stream buffer to the network, the control processor creates a DMA transfer list in the control processor's local memory. Each entry in the list specifies the DMA transfer of a packet to the network interface, which in turn forwards the packet to the network. The DMA transfer list contains a list element for each packet in the packet cluster to be transferred for each active video stream. The address of the DMA list is then written in the control registers of the Microchannel interface in the control processor.

5. Stream controller software

The key software running on the control processor in the stream controller consists of two real time processes, the disk read process and the network transmit process, which transfer data from the disks to stream buffers and from there to the network. The disk read process is a periodic process, each period being equal to the GOB playback duration for the stream controller. In each GOB playback interval the disk read process issues a read command for each active multimedia/video stream to retrieve a GOB of that stream from a disk. The GOB playback period is also referred to as disk cycle. Each GOB playback interval is further divided into n network cycles, and the network transmit process executes once in every network cycle to transmit a packet cluster from the stream buffer to the network for each active video/multimedia stream. In our prototype, timer generated interrupts define the disk and network cycles, and initiate the disk read process and the network transmit process respectively.

The stream control table, shown in Figure 5, is the central data structure in the stream controller. It maintains the state for each active continuous media stream in a data structure called the stream control block, and drives the disk read process and the network transmit processes. The stream control block has pointers to two stream buffers in the stream buffer memory, one to the read buffer which is receiving data for that stream from the disks, and the other to a previously filled buffer which holds data being transmitted to the network. As shown in Figure 5, each of the buffer pointers indicate the starting location of the buffer in the stream buffer memory, the size of the buffers, the location being currently read/written, and a marker bit indicating for read buffers the completion of transfer from disk.

The next pointer in the stream control block points to the GOB being transferred from the disk to the stream buffer. The play back mode field indicates whether the stream is in normal play back mode, paused, or in fast-forward/rewind mode. The client information field points to the block of information needed to complete the network protocol processing on the network packets retrieved from the disk. In our prototype it contains the destination IP address and UDP port number for the video stream. Optional fields may be used to store the pointers to arrays of header and trailer templates stored in control memory when the data on the disk is not stored with the network protocol headers and trailers. The last field is also optional and specifies the stopping condition for the stream as either the pointer to the last GOB to be played back or the time remaining until the completion of playback.

Figure 6 shows the flow of disk read process and the network transmit process. The first step of the disk read process is to change the roles of the read and transmit buffers, and initiate the network transmit process. Then read commands are issued to the disks to transfer the GOBs read from the disk into the stream buffers. In step 3 we check for the completion of the GOB transfer initiated in the previous step and use its GOB header to compute the DMA (Direct Memory Access) list for the network transmissions in the first network cycle of the next disk cycle. Finally, in step 5 we create the SCSI scripts for reading GOBs from the disk in next disk cycle. All of the above steps are performed for each active video stream.

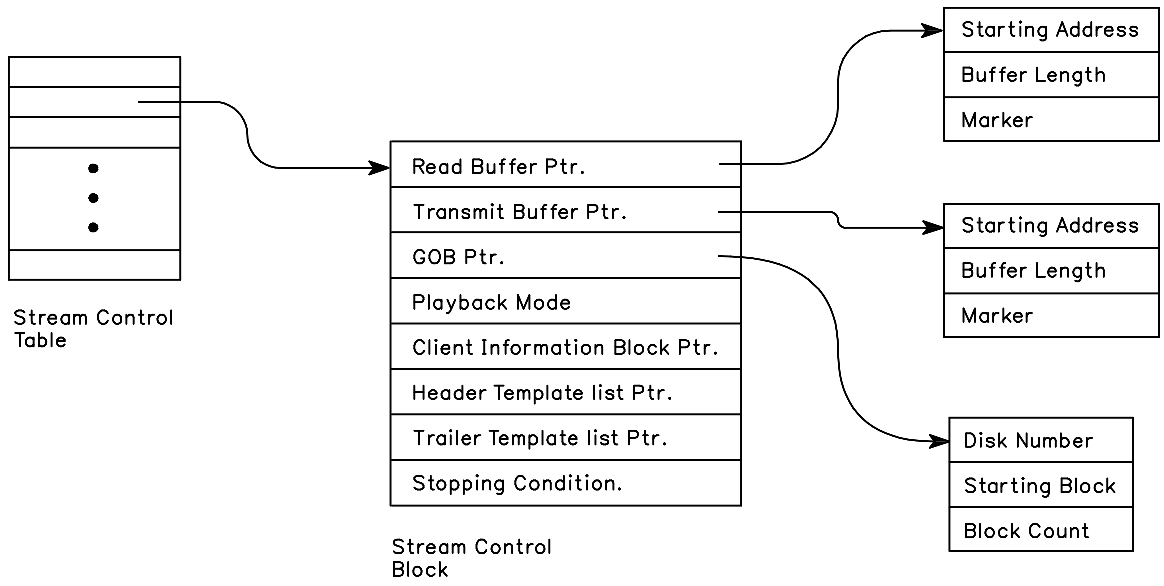

Figure 5: The stream control table and stream control block.

The network transmit process consists of three steps. In step 1, DMA commands are issued to network interface to transmit a packet cluster from the transmit buffer of each active video stream into the network. Step 2 checks whether the current network cycle is the last network cycle of a disk cycle. If it is not, DMA lists are created in step 3 for network transmissions in the next network cycle. Note that DMA lists computed by the disk read process are used in the first network cycle of a disk cycle, while these computed by the network transmit process itself are used in the remaining network cycles.

The data read from the disk by a read command must have the play back time of one disk cycle because exactly one read command is issued for each I/O request in one disk cycle. Similarly, the data transmitted over the network for each active stream in one network cycle must have the play back time of one network cycle.

A non real time process presents a VCR type interface to the host and in response to the host requests, allocates the stream controller resources to a continuous media stream being created, or releases the resources of a stream being terminated. Service processes to load data from the host to the disks managed by the stream controller, to allow the host to check various data structures in the stream controller, and to verify the contents of the disks are also part of the stream controller software.

Since a video file is interleaved across all disks with each GOB having fixed playback time, the multimedia/video streams hop from one disk to the next in one disk cycle and thus maintain their relative position with respect to each other. If a new video stream can be added to a group of streams accessing a disk in a disk cycle without overloading the disk, the expanded group will move from one disk to next together, accessing consecutive disks in consecutive disk cycles without overloading them.

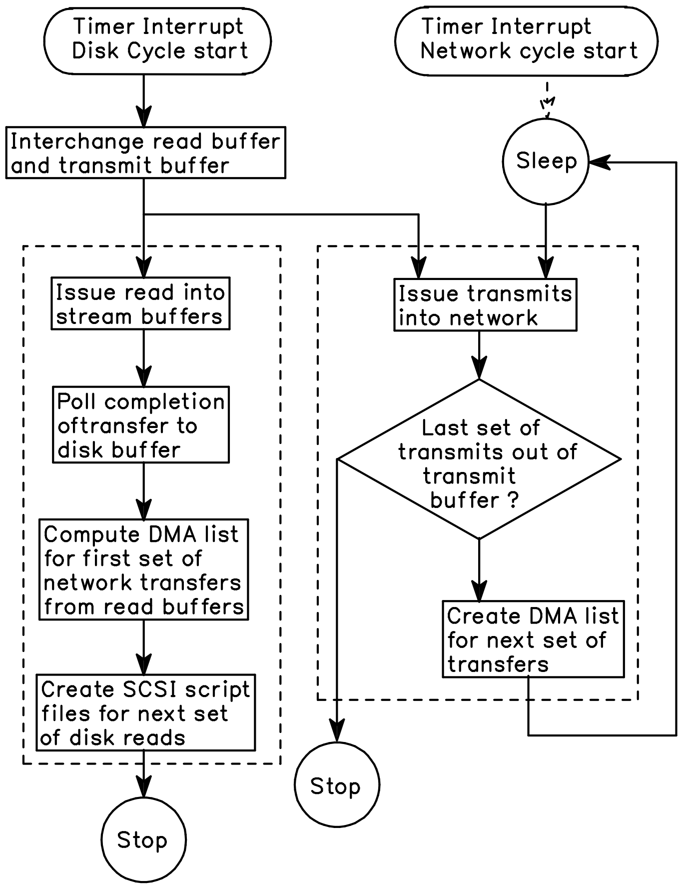

Figure 6: Flow chart of the real time software in stream controller.

We have assumed that all disks have identical throughput. We have further assumed that there is a reasonable upper bound on the size of a GOB, so that we can guarantee the maximum number of GOBs that can be read from a disk in a disk cycle.

Interference on the disks is managed by using a schedule table shown in Figure 7. It has one column for each disk controlled by the stream controller. The number of rows in the schedule table equals the maximum number of streams that can read a GOB from one disk in a disk cycle. Since the relative position of the streams does not change, stream S1 is arbitrarily assigned to first disk in first row. To start a new stream S_n, the scheduler simply finds an empty cell in the schedule table and enters the stream identifier in it. Suppose the cell is in column i. The new stream will always stay $i-1$ streams ahead of stream S1 accessing disks with other streams in column i.

6. Status and performance

We implemented a testbed comprising of our video server prototype, the ORBIT 620 Mbit/sec. optical ring network [3], and a pair of multimedia stations. The ring network was programmed to circulate all streams transmitted from the server, and each multimedia station

1	2	3			N
S1		S2			S7
S3		S5			S8
S4					S9

Figure 7: The schedule table for a disk array having N disks. Each disk can read 5 GOBs in each GOB play back interval.

could select any two streams and display them in two windows. The video server and the testbed were operational in November 1994. With essentially unoptimized code, the server sustained 80 simultaneous MJPEG streams.

With prepacketization of the video stream, the i960 CA processor (approximately 7 MIPS performance) could handle the 80 streams with 30% processor utilization, even though the code was not fully optimized. The path length required per IP/UDP packet for header modification and preparation of the network DMA transmit list is well under 300 instructions. Given the above facts, we are confident that the full system performance of 250 streams is achievable with the i960 processor.

Striping by time was key to keeping the buffer requirements minimal. In the first implementation described above, 2 buffers of only 64K bytes each were used per stream. This was possible because the disk response time could be guaranteed to be less than one GOB play back period. By comparison, video servers based on a conventional Unix operating system and file systems will require several Megabytes of buffer memory per active stream, and the fastest of available processors will deliver around 50 MPEG-1 streams. To increase the performance of the prototype to 250 MPEG-1 streams, we are optimizing the video delivery code to further reduce buffer requirements. Stream buffers are currently the limiting resource.

7. Future research

In this paper we described the most basic implementation and operation of our video server design. We plan to, and encourage other researchers to pursue the following extensions which can significantly improve the usability of the server.

The first and foremost issue is the one of fault tolerance. We alluded to it when describing the GOB header which stores pointers to all GOBs in the parity group of the next GOB. However, all disks in a cost effective multimedia/video server will be reasonably loaded. If the GOBs in the parity groups of each GOB on the failed disk are stored on the same set of disks, these disks too will be overloaded. Some mechanism is required to spread the parity groups of GOBs on each disk across all disks in the array. An even better scheme would be to compute parity across consecutive GOBs of the same stream, so that additional fetches to compute the GOB on the failed disk can be avoided. Details of these approaches have to be worked out.

We have not completely worked out the mechanism for supporting fast forward and rewind functions in the proposed server. The issue to be researched is how to control interference on the disks when different streams are moving from one disk to the next at different speeds. Our initial approach is to reserve a fraction of disk bandwidth for the fast moving streams.

The network protocol we used in our Lab was UDP/IP, and it worked well because MJPEG is much more error resilient than MPEG. Furthermore, the network confined to our Lab. was rather error free. In real environments, one would need to address the issues of reliable delivery, when required. Another very important issue is one of multi-protocol support for inter-operability. We are looking into different data structures for storing data which allow us to efficiently construct network packets for different protocols stacks.

A cost effective solution to the above problem of delivering multimedia/video data to large number of clients simultaneously is also applicable to transferring large files from servers to clients over a broadband network. In this situation, even though the clients are capable of accepting data from the network at a much higher rate than a typical multimedia/video client, the cost structure expected for the network usage will require data to be sent in regularly spaced chunks of small size at a pre-negotiated aggregate bandwidth much lower than the bandwidth of the I/O subsystem in the server.

8. Summary and conclusions

In this paper we presented the architecture, design, and implementation of a multimedia/video server, which embodies two important concepts. First, storing multimedia/video data in the format of network packets, and second, embedding metadata in the video stream for efficient retrieval of sequentially accessed data. With the implementation we demonstrated that high performance cost effective multimedia/video servers can be built using these two concepts.

References

[1] M.M. Buddhikot, G.M. Parulkar, J.R. Cox, "Design of a large scale multimedia storage server," Computer networks and ISDN systems, 27(3), Dec. 1994, pp. 503-517.

[2] E. Chang and A. Zakhor, "Variable bit rate MPEG video storage on parallel disk arrays," Proc. lst IEEE Int. Workshop on Community Networking, San Francisco CA, July 13-14, July 1994, pp. 127-137.

[3] I. Cidon et.al., and M. Kaplan, "The plaNET/ORBIT high speed network," J. High Speed Netw. (Netherlands) 2(3), 1993, pp. 171-208.

[4] A. Dan and D. Sitaram, "Buffer management policy for an on-demand video server," IBM Research Report RC 19347.

[5] A. Dan and D. Sitaram, "Scheduling policy for an on-demand video server with batching," IBM Research Report RC 19381.

[6] A.L. Drapeau et.al., "RAID II: a high bandwidth network file server," Proc. of 21 Int. Symp. on Computer Architecture Chicago IL, April 1994, pp.234-244.

[7] R. Haskin, "The Shark continuous media file server," Proc. IEEE COMPCON 1993 (San Francisco CA, 1993), pp. 12-15.

[8] D. Kandlur, M. S. Chen, and Z. Y. Shae, "Design of a multimedia storage server," Proc. SPIE on high speed networking and multimedia computing, Vol. 2188, San Jose CA, 1994, pp. 164-178.

[9] P. Lougher and D. Shepherd, "The Design of a storage server for continuous media," the Computer Journal, 36(1), 1993, pp. 32-42.

[10] P.V. Rangan and H.M. Vin, "Designing file systems for digital video and audio," Proc. 13th Symp. on Operating Sys. Principles, Oct. 91, pp. 81-94.

[11] N. Reddy and J. Wyllie, "Disk scheduling in a Multimedia I/O system," Proc. ACM Multimedia 93, Anaheim CA Aug 93, pp. 225-233.

[12] F. A. Tobagi, J. Pang, R. Baird, and M. Gang, "Streaming RAID - A disk array management system for video files," Proc. ACM Multimedia 1993, pp. 393-400.

[13] H. M. Vin and P. V. Rangan, "Designing a multiuser HDTV storage server," IEEE Jour. on Selected Areas in Communications, 11(1), Jan. 1993, pp. 153-164.

[14] P. Yu, M. Chen, and D. Kandlur, "Design and analysis of a grouped sweeping scheme for multimedia storage management," Proc. Third int. workshop on Network and Operating Systems Support for Digital Audio and Video., La Jolla, CA, USA 12-13 Nov. 1992, pp. 44-55.