MARS: A High Performance Large Scale Multimedia Storage Server *

Milind M. Buddhikot
milind@dworkin.wustl.edu

Department of Computer Scince,
Washington University in St. Louis,
St. Louis, MO 63130.
+1 314 935 4203

May 8, 1995

Abstract

This proposal deals with the design and prototype implementation of a large scale high performance multimedia storage server which will be an important part of the emerging distributed multimedia computing infrastructure. The proposed server architecture, called MARS, *uses some of the well-known techniques in parallel I/O, such as data striping and Redundant Arrays of Inexpensive Disks (*RAID*) and an innovative* ATM *based interconnect inside the server to achieve a scalable architecture that transparently connects storage devices to an* ATM*-based broadband network.*

In conjunction with this architecture, we have developed a framework for distributed data layout and scheduling schemes. The salient features of these schemes are: allow potentially all clients to access the same data without data replication, support both buffered as well as bufferless clients, provide deterministic or statistical guarantees to each active client for normal playout as well as for a full spectrum of interactive stream control operations (namely, fast forward, rewind, random access, frame advance, slow play, slow rewind, pause, stop-and-return and stop*).*

In this proposal, we will describe open problems in the areas of distributed data layout, scheduling, and storage node design, and present our preliminary solutions and a plan of action for the on-going work.

1 Introduction

Rapid advances in the related fields of optical communication, fast packet switching, data compression, and processor design indicate that a ubiquitous distributed multimedia computing infrastructure will be feasible in the near future. Large scale multimedia storage servers and compute servers will be an integral part of this infrastructure which will support exciting new applications such as multimedia mail, orchestrated presentations, high quality on-demand audio and video, collaborative multimedia document editing, remote multimedia archives browsing, and virtual reality environments. The recognition that the existing network based storage servers cannot meet the requirements of these new applications has led to a flurry of research activity in the area of multimedia storage server design. The challenges in this research area are a result of the following requirements:

*This work was supported in part by the ARPA, National Science Foundation, and an industrial consortium of Ascom Timeplex, BNR, Goldstar, NEC, NTT, Bay Networks, and Tektronix.

- **Large Storage:** A large scale server must support potentially terabytes of storage and must allow multiple storage types. For example, a lecture archive in a typical university department which offers 50 courses in a semester with 50 1-hr lectures, would require roughly 6 TBs of storage.
- **Large network and storage throughput:** The server must be able to deliver storage and network throughput in excess of a few Gbps. For example, in case of a digital lecture archive, assuming the peak load is 500 clients, the throughput requirement is approximately 2.5 Gbps, which is three order of magnitude higher than the throughput of a single high performance disk. Given the modest rate of improvement in secondary and tertiary storage technology, storage throughput of Gbps magnitude is difficult to provide.
- **Playout Control:** Since, interactivity is important for future applications, the server must be able to support independent, interactive playout control operations for each active client. For example, in case of a movie server, if there are 200 clients active concurrently, ideally, each one of them must be allowed to be in any playout control state independent of other clients. The server must support the full spectrum of interactive operations such as *ff*, *rw*, *pause*, *slow-play*, *frame-advance*, *stop*-and-return and random access. Also, any requested playout control operation should be effected with minimal latency, typically less than one second.
- **Quality of Service Guarantees:** The server must be able to provide QOS guarantees in terms of throughput and periodic retrieval to all clients that are active concurrently. Such guarantees must be satisfied during normal play and playout control operations.
- **Non real-time accesses:** In addition to real-time streams, the server must support access to non real-time streams such as text, images, data files etc. with bounded latency.

One of the greatest hurdles in designing high performance storage servers that meet all of the above requirements stems from the persistent storage I/O bottleneck. The primary reason for such a bottleneck is that the processor, memory and network speeds have been improving faster than the speed of on-line secondary storage [9]. The mismatch between the requirements of multimedia and the traditional operating system support for secondary storage devices aggravates this bottleneck even further. Also, in the traditional servers, the data flow between the storage devices and the network involves multiple data copies, hence reducing the overall throughput for network oriented tasks to a small fraction of the total network throughput. Thus, the design of high performance storage I/O is a challenging task that needs innovative solutions. Our project called the *Massively-parallel and Real-time Storage* (MARS) project represents an effort in this direction.

The salient features of our architecture are the use of some of the well known techniques in parallel I/O such as data striping and Redundant Arrays of Inexpensive Disks (RAID), and an innovative ATM based interconnect inside the server, to achieve a scalable architecture that transparently connects storage devices to an ATM based broadband network. In it's simplest form, a MARS architecture can be looked upon as a set of storage nodes interconnected by a fast ATM interconnect. The ATM interconnect in the proposed architecture uses a custom ASIC called the *ATM Port Interconnect Controller* (APIC), which serves as the basic building block for a high bandwidth *networked*-I/O *subsystem.* We expect such ASICs will become off-the-shelf component in the near future and thus, allow high speed interconnects to be built inexpensively. Our architecture also relies on innovative data striping and real-time scheduling to allow a large number of guaranteed concurrent accesses and uses separation of metadata from real data to achieve a direct flow of the media streams between the storage devices and the network.

For this dissertation, we have identified following three research problems, the solutions of which will demonstrate the viability of the proposed architecture: (1) design of distributed data layout schemes that can support maximum concurrency and parallelism, and allow unconstrained playout control; (2) design of a distributed scheduling scheme that synchronizes data prefetch from storage devices and data transmission from nodes on the ATM interconnect to satisfy the QOS guarantees for all active clients. Such a scheme must implement all playout control operations with minimal latency; (3) the design and prototype implementation of hardware and software systems at the storage node that efficiently implement these data layout and scheduling schemes. The solutions to these problems have to account for MPEG as the default compression standard, which puts a unique set of constraints on the system.

The rest of this proposal is organized as follows: Section 2 presents background on models for on-demand multimedia and playout control implementations. Section 3 discusses the prototype architecture of the MARS server and describes a simple distributed data layout and scheduling scheme that represents the first step in meeting the goals. Section 4 presents the open problems in distributed data layout, scheduling and storage node design. The preliminary solutions and a plan of action for the proposed research are presented in Section 5. Section 6 presents a cross-section of the great body of notable related work in this field. Finally, Section 7 presents the expected results of the proposed research.

2 Background, Models, and Environment

Before discussing our proposed architecture and the associated data layout and scheduling schemes, we present some background information on service models and playout control operations.

2.1 Models for On-demand Multimedia

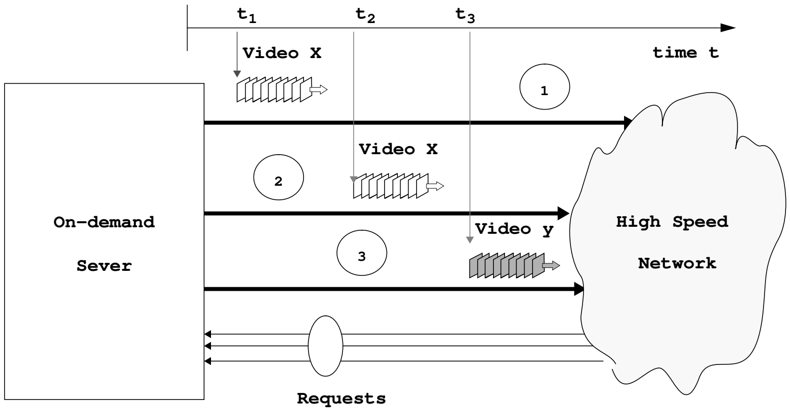

Figure 1: MOD server for a Dedicated Viewing service (DV)

We assume that the server provides multimedia access using a *Dedicated Viewing* (DV) service. In this type of service, illustrated in Figure 1, the client accesses the multimedia documents by sending requests to the server. The server consults an admission control procedure, which, based on the existing load and the resource usage, admits or rejects the client request. Each such request, from the same or different client, is treated independent of all other requests at the server. For example, in Figure 1, requests received at time t_1 and t_2, though close in time and for the same video X, are treated as two separate requests and hence, require separate retrieval and delivery. Thus, in an on-demand movie server supporting a maximum of 1000 users with DV service, all 1000 users, in the extreme case, may watch the same movie independently and will therefore, require 1000 independent retrievals, and deliveries over separate network connections. Note that each of these 1000 clients has its own strict timing requirements that need to be met in the presence of other clients that are active concurrently. A serious implication of this is that it requires independent data retrieval and transmissions for each active connection and thus, increases network and storage throughput requirements immensely. The primary advantage of this service is that each active connection can independently exercise desired playout control operations on any multimedia document. Thus, in the example above, any subset of 1000 active clients can be performing fast forward, rewind, random access or other playout control on the same or a different movie. In short, the DV service is a natural paradigm for **highly personalized, interactive** multimedia delivery, which we believe is an important advantage of digital media.

Other service models, such as Shared Viewing (SV) (also called *Pay-per-view*) and Shared Viewing with Constraints (SVC) (also called *"near video"*) have been proposed [19, 6, 22, 1]. These models reduce the storage and network throughput requirements using the bandwidth scaling property of multicast communication and are satisfactory for a class of less sophisticated multimedia applications. We believe that in future, these three services will co-exist and cater to different classes of applications. The DV services will be important for a large class of exciting new applications that are inherently interactive.

2.2 Playout Control

Future on-demand multimedia applications will require interactivity in the form of stream playout control that allows the user to do *fast forward, rewind, slow play, slow rewind, frame advance, pause,* and *stop-and-return* on a media stream. Also, unlike the linear access supported by present-day video cassettes, a user may access the media streams in a random fashion (as permitted by existing CD-ROMS and laser disks). These operations can be classified into three groups as shown in Table 1. Note that logically similar operations are grouped together. For example, *ff* and *rw* increase the perceived rate of display and hence are in the same group. Similarly, *slow-play, slow-rewind* and *frame-advance* reduce the perceived rate and hence, belong to the same group.

Table 1: Classification of playout control operations

Group 1	*ff, rw, fast play (fp), fast rewind (fr)*
Group 2	*slow-play, slow-rewind, frame advance*
Group 3	*pause, stop-and-return , stop*

The two ways of implementing the stream control operations are as follows:

- **Rate Variation Scheme (RVS):** In this scheme, the operation changes the rate of display at the client and hence the rate of data retrieval and transmission at the server. The performance analysis of a large-scale server using such a scheme has been reported in [13].
- **Frame Skipping Scheme (FSS):** In this scheme, the operation changes the sequence of frame display and hence the sequence of data retrieval and transmission at the server. The display rate at the client is unaltered but the data retrieval and transmission rate used at the server may be affected.

As an example, consider the fast forward operation. In the implementation using **RVS**, the display rate at the client terminal is increased to give the user a perception of fast forward. For example, a video stream may be played at 90 frames/sec (fps) instead of standard 30 fps. On the other hand in case of **FSS**, irrespective of whether a video stream is in normal play mode or *ff* mode, the display rate is always 30 fps. The perception of fast forward is achieved by displaying a altered frame sequence, e.g.: playing only every alternate frame, every 5^{th} frame, or in general, playing every d^{th} frame ($d =$ fast forward distance).

The **RVS** implementation of *ff* and *rw* increases network and storage BW requirements as well as the buffers required at the client side. Also, it is inappropriate for the current real-time decoders as they can decode and display data only at rates smaller than or equal to a maximum frame rate. Hence, we implement *ff* and *rw* using **FSS** approach. We distinguish *fast-play* (*fp*) and *ff* as two different operations and implement *fp* using the **RVS** approach. A similar distinction can be made between *fast-rewind* (*fr*) and *rw*. Typically, *ff* (*rw*) will be supported for all active connections, whereas *fp* (*fr*) will be a special operation, subject to resource availability.

Note that operations such as slow play and slow rewind can only be implemented using the **RVS** approach. However, these operations reduce the resource usage, and hence are easier to implement. Also, operations such as pause, frame-advance and stop-and-return do not fall under any particular classification and are easy to implement.

3 Proposed Architecture

The basic idea of the *Massively-parallel and Real-time Storage* (MARS) architecture has been discussed in reference [2]. Here, we will discuss a prototype implementation of the MARS architecture. We will also, describe an example data layout and scheduling scheme that represents the first step in meeting all the requirements of a large scale server stated earlier.

3.1 Storage Cluster

A prototype implementation of the MARS architecture, called Storage Cluster (SCL), is illustrated in Figure 2. It consists of three basic building blocks: a cell switched ATM interconnect, storage nodes and the central manager. The ATM based interconnect uses a custom ASIC called *ATM Port Interconnect Controller* (APIC). The APIC chip is the basic building block for a high bandwidth *networked*-I/O *subsystem*, that provides a direct interface to the network for the host (workstations as well as servers) and a variety of I/O devices.

In its simplest form, the APIC behaves like a 3×3 switch, two of whose ports can be treated as ATM ports and the remaining one as a non-ATM port. Using one of the ATM ports and a line interface, the APIC can be directly interfaced to the input port of an ATM switch. As shown in Figure 2, using the ATM ports, multiple APIC chips can be connected in a bidirectional daisy chain. Each port is designed to operate at full 1.2 Gbps (SONET) rate and thus, the aggregate data rate on the interconnect is ≈ 2.4 Gbps. The non-ATM port of an APIC can be treated as a read/write port to a data source, such as memory or an I/O device. The APIC performs incremental AAL5 fragmentation and reassembly of ATM cells through this port.

Each APIC manages a set of connections by maintaining state information for each active connection in a Virtual Circuit Translation Table (VCXT). The APIC also performs a simple single parameter (such as average bandwidth) based rate control or source pacing for each connection, which is important in scheduling multimedia streams from storage node to the network. The description of the detailed design of the APIC is beyond the scope of this document and can be found in [15, 14, 16].

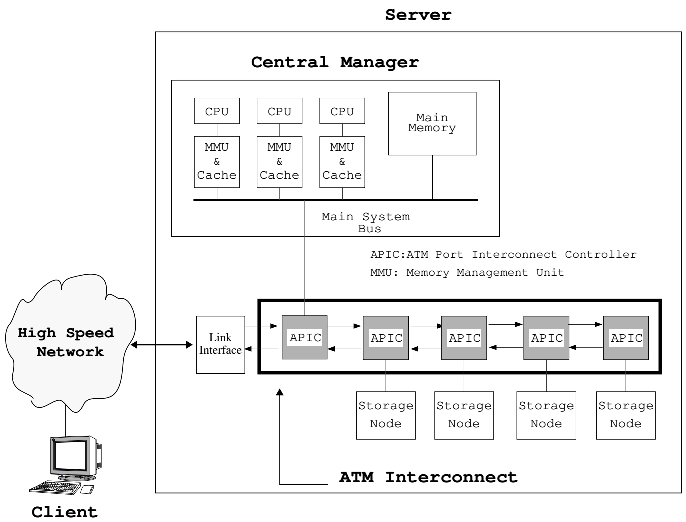

Figure 2: A prototype architecture

Each storage node provides storage in the form of RAIDs, tape libraries, optical disks etc. and provides resource management functions such as file system support, scheduling support and compute support.

The central manager, shown in the figure, is responsible for managing the storage nodes and the APICs in the ATM interconnect. For every media document, it decides how to distribute the data over the storage nodes and manages the associated meta-data information. It receives the connection requests from the remote clients and based on the availability of resources and the QOS required, admits or rejects the requests. For every active connection, it also schedules the data read/write from the storage nodes by exchanging appropriate control information with the storage nodes. Note that the central manager only sets up the data flow between the storage devices and the network and does not participate in actual data movement. This ensures a high bandwidth path between storage and the network.

A more scalable architecture can be realized using several SCLs interconnected through a multicast ATM switch and controlled by a central manager. The storage and network throughput of this architecture increases linearly with the number of switch ports and storage clusters.

3.2 Distributed Data Layout and Scheduling: A Simple Example

We now illustrate a simple distributed data layout and scheduling scheme for the prototype architecture and show that it can support a large number of clients in normal playout mode.

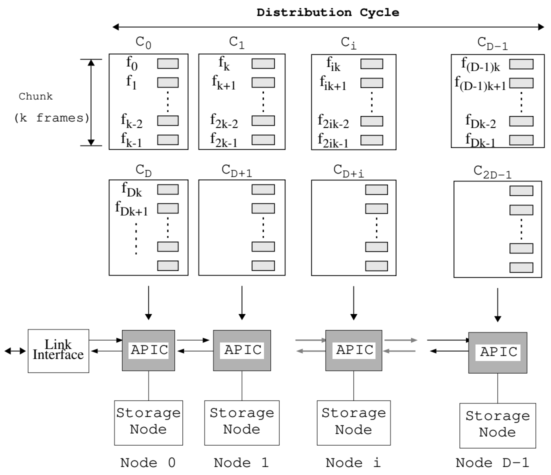

Figure 3: A simple chunked data layout scheme

We use the fact that the multimedia data is amenable to spatial striping to distribute it over several autonomous storage nodes within the server. For example, a video stream can be looked upon as a succession of logical units repeating periodically at a fixed rate. A logical unit for video can be a single frame or a collection of frames, and the period of repetition can be the frame period, say 33 msecs/frame, or an integral multiple thereof. Each such logical unit or parts of it can be physically distributed on different storage devices and accessed in parallel. One of the possible layout schemes, called *Distributed Cyclic Layout* (DCL_k), is shown in Figure 3. The layout uses a basic unit called *"chunk"* consisting of k consecutive frames. The chunk size can range from $k = 1$ to $k = F_{max}$, where F_{max} is the maximum number of frames in the multimedia document. For example, in case of MPEG compressed streams, the group-of-pictures (GOP) is one possible choice of chunk size. A chunk is always confined to one storage node. The successive chunks are distributed over storage nodes using a logical layout topology. For example, in Figure 3, the chunks have been laid out using a ring topology. Note that in this scheme, the two consecutive chunks at the same node are separated in time by kDT_f time units, D being the number of storage nodes and T_f the frame period for the stream. Thus, if the chunk size is one frame (DCL_1 layout), from the perspective of each storage node the stream is slowed down by a factor of D or the throughput required per stream from each storage node is reduced by a factor of D. This facilitates prefetching of data to mask the high rotational and seek latencies of the magnetic disk storage. Note that if $k = F_{max}$, the entire document is stored on the one storage node and the frames/sec perceived by the storage node is the native frames/sec of the document.

A simple scheme for scheduling data retrieval from unsynchronized storage nodes when clients are bufferless is as follows. In this scheme, each storage node maintains C_a buffers, one for each active connection. The data read from the disks at the storage node is placed in these buffers and read by the APIC. At the time of connection admission, every stream experiences a playout delay required to fill the corresponding buffer, after which the data are guaranteed to be periodically read and transmitted as per a global schedule.

The global schedule consists of periodic cycles of time length T_c at each node. Each cycle consists of three phases: *data transmit*, *handover* and *data pre-fetch*. During the *data transmit phase* (T_{Tx}), the APIC corresponding to a storage node reads the buffer and transmits it over the interconnect to the network. Once this phase is over, the APIC sends control information (control cell) to the downstream APIC so that it can start it's data transmit phase. This time phase of transfer of transmission control constitutes the *handover phase*. The last phase in the cycle, namely the *data pre-fetch phase* (T_{pf}) is used by the storage node to pre-fetch for each connection the data that will be consumed in the next cycle.

As an example, consider a prototype with 15 storage nodes ($N = 15$), each with a disk array capable of providing sustained storage throughput of 5 MBps. Since, the data layout described above allows seek and rotational latencies to be effectively masked, we can operate each disk array at its full bandwidth. Also, the simple scheduling scheme above allows each APIC to operate at full 1.2 Gbps. Thus, the aggregate storage and network throughput from the server is 1.2 Gbps, which can support as many as ≈ 54 compressed HDTV clients or 220 standard NTSC quality identical MPEG clients.

Given this introduction, we now define two performance metrics. The first one called *parallelism* (P_f), is defined as the number of storage nodes participating concurrently in supplying the data for a document f. The second metric called *concurrency* (C_f) defines the number of active clients that can simultaneously access the same document f.

The value of P_f ranges from 1 to D, where D represents the number of storage nodes. P_f is D, when the data is distributed over all nodes, whereas, it is one when the entire document is confined to a single storage node. A higher value of P_f implies, larger number of nodes are involved in transfer of data for each connection/request, which in turn improves node utilization and proportionately increases concurrency. The P_f metric also has implications on reliability. If a document is completely confined to a storage node ($P_f = 1$), in the event of a failure of that node, it is entirely unavailable. However, with loss tolerant media, such as video, animation, and graphics, a higher P_f will still keep the document partially available.

If each storage node n ($n \in [1 \ldots D]$) has an available sustained throughput of B_n, and the average storage/network throughput required for accessing the document f is R_f, then the concurrency supported by a layout with parallelism P_f is

$$C_f = \min_n \left(\frac{B_n P_f}{R_f} \right)$$

The maximum value of the concurrency metric is a function of the total interconnect bandwidth I_{bw} and is defined as

$$C_{f,max} = \frac{I_{bw}}{R_f}$$

Clearly, maximum concurrency is achieved if $I_{bw} = B_n D$. From above expressions, it is clear that concurrency is a function of the parallelism supported by the data layout. Higher concurrency is desirable as it allows larger numbers of clients to simultaneously access the same document and thus minimizes the need for replicating the document to increase concurrent accesses.

4 Open Problems

In the previous section, we illustrated an architecture that connects storage devices transparently to the external network. The simple data layout and scheduling described there assumed an ideal environment in which all connections have identical display rate and are always in normal playout mode. Also, all the data units are of the same size. Such a situation is embarrassingly parallel, where storage nodes can be easily synchronized and peak performance can be extracted from the system. However, in a non-ideal environment in which clients require independent interactive playout control and data streams are variable bit rate in nature, performance can be significantly affected. In order to eliminate or minimize such performance deterioration, several open problems need to be addressed. In the following discussion, we will first outline these problems, and then present preliminary solutions and a plan of action for our on-going work.

4.1 Distributed Data Layout

Given a multimedia document, the problem of distributed data layout deals with how to distribute the data on different storage nodes. The two main problems in distributed data layout design are: First, how to maximize the concurrency C_f and the parallelism P_f metrics during normal play and arbitrary playout control operations. Second, how to implement playout control operations such as *ff* and *rw* on MPEG compressed documents which have interframe dependencies. Each of these problems is discussed in greater detail below.

4.1.1 Support for interactive control

Earlier we illustrated a simple layout called Distributed Cyclic Layout (DCL_1) that maximizes the parallelism P_f and the concurrency C_f metrics, when the document is accessed during the normal playout mode independently for each active client. However, with such a layout, implementing *ff* and *rw* by frame skipping can lead to potential load-imbalance situations. Consider a connection in an example system with $D = 6$ storage nodes, a DCL_1 layout, and a fast forward implementation by skipping alternate frames (i.e. the frame skipping distance for fast forward $d_f = 2$). The frame sequence for normal playout is $\{0, 1, 2, 3, 4, 5, \ldots\}$ and these frames are retrieved from nodes $0, 1, 2, 3, 4, 5$ in a linear (mod 6) order. On the other hand, for the fast forward by skipping alternate frames, the frame sequence is altered to $\{0, 2, 4, 6, 8, 10, \ldots\}$ and these frames are retrieved from nodes $0, 2, 4$ in linear order. This implies that in this example, the odd-numbered nodes are never visited for frame retrieval during *ff*. Thus, when a connection is being serviced in *ff* mode, the load measured in terms of the number of frames retrieved doubles for even numbered nodes and reduces to zero for odd numbered nodes. In other words, the parallelism P_f is reduced from D to $D/2$ during *ff* of a connection and the concurrency must be proportionately reduced. Clearly, in presence of a large number of connections independently exhibiting interactivity, this can lead to occasional severe load imbalance in the system and can make it difficult to satisfy the QOS contract agreed upon with each client at the time of connection setup. Thus, if we can ensure that P_f and consequently, C_f are unaffected during *ff* or *rw*, we can guarantee load-balance situations.

One way to do this, is to use only those frame skipping distances that do not affect P_f and C_f. We call such skipping distances as "safe skipping distances" (SSD). Thus, given a data layout, we want to know in advance all the SSDs a distributed data layout can support. In order to provide a rich choice of ff *(and* rw*) speeds, the number of such SSDs should be maximized.*

4.1.2 Interactive Control on MPEG Compressed Documents

The properties of MPEG streams have serious implications for *ff* and *rw* operations[1]. The MPEG compression introduces inter-frame dependencies and thus, does not allow, frame skipping at arbitrary rate. This in effect means that *ff* by frame skipping can realize only a few rates. Specifically, a *ff* sequence obtained by skipping a constant number of frames, can contain only I and/or P frames and no B frames. If the first P frame in a GOP is skipped, no other P frame from the same GOP can belong to the *ff* sequence. Thus, the simplest *ff* frame sequences for such a stream are [IPP $\cdots$ IPP $\cdots$ IPP $\ldots$] or [IIII$\ldots$]. This implies that the set of fast forwarding distances (d_f) for a **FSS** implementation of *ff* on a MPEG stream will be a function of the GOP size and the separation

[1] MPEG allows D-frames to support simple, but limited quality *ff*. These frames are not likely to be included in final MPEG-2 standard and hence, are not considered in our study.

between I and P frames. Also, such a fast forward implementation will work if the GOP size and structure is constant for the entire duration of the stream.

There are two problems with sending only I frames on *ff*. First, it increases the network and storage BW requirement by at least a factor 3. For example, a MPEG compressed NTSC video connection that requires 5 Mbps on average during playout would require approximately 15 Mbps for the entire duration of the *ff*, if only I frames are transmitted. In the presence of many such connections, the network will not be able to easily meet such dramatic increases in bandwidth demand, and reserving such BW at the connection setup will be immensely wasteful. Another problem is that if the standard display rate is maintained, skipping all the frames between consecutive I frames, may make the effective fast forward rate unreasonably high. For example, if I-to-I separation is 9 frames, perceived fast forward rate will be 9 times the normal playout rate.

Given this, we need to design and evaluate schemes for implementing ff *and* rw *operations on* MPEG *compressed data that offer a sufficiently large choice of* ff *and* rw *speeds, and at the same time keep the storage capacity, storage and network throughput requirements and the visual quality unaffected. Also, a clear characterization of variation of network and storage bandwidth requirements during* ff *and* rw *and an evaluation of the subjective quality of various options must be obtained.*

4.2 Distributed Scheduling

The distributed scheduling can be defined as the periodic retrieval and transmission of data for all active connections from unsynchronized storage nodes. It is required to provide QOS guarantees to the active clients during the normal playout as well as playout control operations.

In the simple scheduling described earlier, each storage node groups and services all active connections together. Also, the sequence of transmissions from storage nodes is identical for all connections and is dictated by the layout scheme. However, such a scheme breaks down when some subset of clients are performing playout control.

Consider a connection in an example system mentioned earlier in the context of the load-balance property of data layouts. In this example, the set of nodes from which the frames are retrieved in the normal playout is $\{0, 1, 2, 3, 4, 5, 0, 1 \ldots\}$. Upon *ff* this node set is altered to $\{0, 2, 4, 0, 2, 4, \ldots\}$. Apart from load imbalance, one serious implication of this is that as the display rate is constant during *ff*, node 2 for example, must retrieve and transmit data in a time position which is otherwise allocated to node 1 in normal playout. Thus, in addition to creation of "hot-spots", the stream control alters the sequence of node-visits from the normal linear (modulo D) sequence and so, the transmission order is no longer the same for all connections when some of them are doing fast forward or rewind. Therefore, the transmission of all connections can no longer be grouped into a single transmission phase.

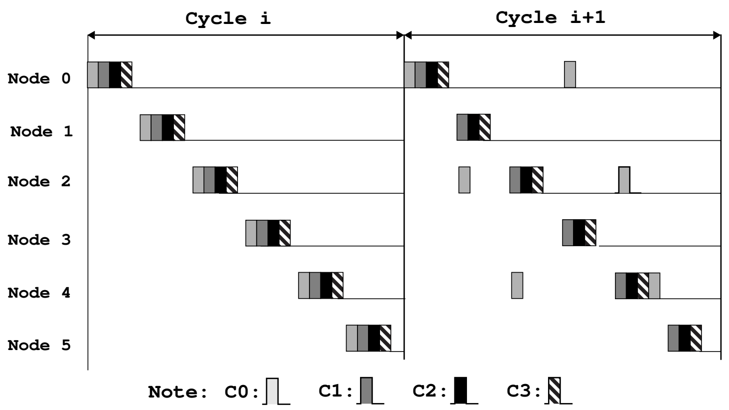

Figure 4: Revised schedule when C_0 performs fast forward

Figure 4 illustrates this with an example of a system with $D = 6$ nodes and 4 active connections, of which C_0 is performing *ff*. It shows two consecutive (i^{th} and $(i+1)^{th}$) cycles. The transmission order in the i^{th} cycle is represented by the ordered node set $S_{play} = \langle D_0, D_1, D_2, D_3, D_4, D_5 \rangle$, which is identical for all connections. When the *ff* request for connection C_0 received in the i^{th} cycle becomes effective, the transmission order for it is altered to the ordered node set $S_{ff} = \langle D_0, D_2, D_4, D_0 \ldots \rangle$ in the $(i+1)^{th}$ cycle. The transmission order for the rest of the connections is unchanged.

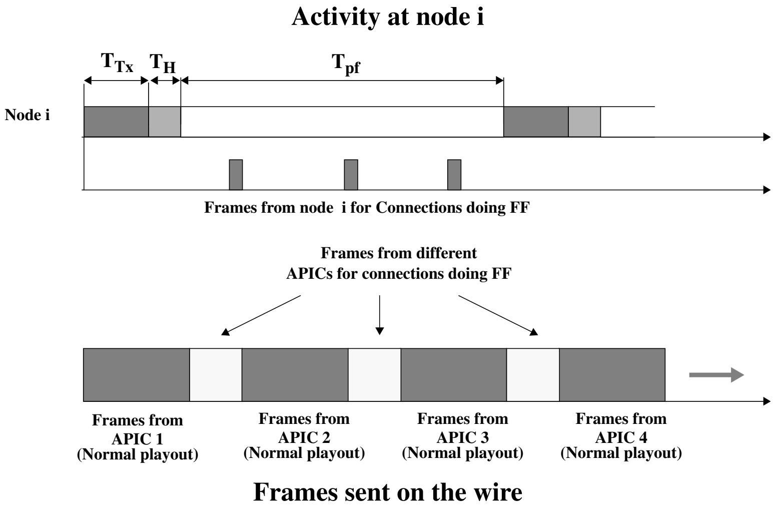

Figure 5: General case of M out of C_a connections doing fast forward

Figure 5 illustrates this when M out of C_a active connections are performing fast forward. At a typical node i the transmission occurs in multiple phases, one of which is for connections performing normal playout and the rest are for connections performing fast forward. These phases cannot be combined into a single phase as the transmission order of all the M connections performing fast forward is not identical. The sequence of frames appearing on the wire consists of two sequences: a sequence of frames transmitted from an APIC followed by frames transmitted from possibly all APICs for connections performing fast forward. It must however be noted that at any time, only one APIC transmits frames for a connection.

The side effect of this revised schedule is that the prefetch and the transmission phases for a storage node overlap. In presence of a large number of connections doing fast forward and rewind, this overlap makes it difficult for cyclic prefetch scheduling to guarantee that data to be transmitted will be available in the buffers.

Thus, we need to design a scheduling scheme which decouples prefetch and transmit operations at a node and allows data transmissions from different nodes to be independently synchronized. Such a scheme should implement playout control operations with minimal latency and support buffered or bufferless clients with identical or different display rates. Also, it should optimize the state information required at each node, and the overhead resulting from message exchange between nodes to order transmissions.

4.2.1 Admission Control

Each storage node in our architecture has finite resources in the form of buffer of capacity B and the storage bandwidth R_N. A fraction of these resources must be reserved for each active periodic real-time connection to provide guaranteed accesses. Such resource reservation requires an admission control procedure that must be consulted before any new reservations are made. Specifically, when a new access request is received, the server interprets the request to decide if it is for real-time (periodic) or non real-time access. If there are C_a real-time connections active at present, the new $(C_a+1)^{th}$ real-time connection should be is admitted, if and only if, QOS guarantees to all C_a+1 clients can be met.

Since, MPEG data is VBR in nature, providing deterministic guarantees requires the wasteful approach of peak BW reservation. Also, multimedia data is tolerant of occasional errors and missed deadlines. Given this, we are interested in statistical or probabilistic guarantees, an approach in which guarantees can be occasionally violated, if the "bursts" for different connections collide. For this approach to work successfully, multiple VBR streams must be statistically multiplexed together. In a cycle based scheduling scheme such as ours, the lack of sufficient BW due to burst collisions translates to total retrieval time exceeding the cycle length required to meet the periodic requirements of all clients. We will attempt to characterize such multiplexing overloads in two steps: the buffer overflow and the cycle overflow. Some of this characterization is inspired from [7, 8].

Due to the VBR nature of streams, the amount of data fetched at each node for each connection varies from cycle to cycle, or stated formally, it is a random variable (**rv**) $D_i, i \in [1 \ldots C_a]$, whose distribution depends on the statistical properties of the MPEG compressed document being accessed. If such distribution is known for each **rv** D_i, the probability that the sum of D_i's exceeds the total buffer capacity B can be computed. If the tolerable buffer overflow probability is $P_{buffover}$, and if the Equation 1 is true for $n = C_a$, then upon admitting $(C_a+1)^{th}$ client, it should still be valid.

$$Pr\left[\sum_{i=1}^{n} D_i \leq B\right] \leq P_{buffover} \tag{1}$$

Similarly, the probability of cycle overflow can be estimated using the knowledge of node level layout and node throughput characteristics, such as disk transfer rate, minimum/maximum disk seek and rotation times. The various node level prefetch schemes and transmit options will have different cycle overflow performance. Specifically, if for a given setup, $T_r(D_i)$ represents time to fetch data of size D_i, and if Equation 2 is true for $n = C_a$, then upon admitting $(C_a+1)^{th}$ client, Equation 2 should still hold true.

$$Pr\left[\sum_{i=1}^{n} T_r(D_i) \leq T_{cycle}\right] \leq P_{cycleover} \tag{2}$$

Note that Equation 1 and Equation 2 need to be satisfied at each node in our architecture. Thus, the admission control policy uses a computation procedure at each node which, given a pair of values $[P_{buffover}, P_{cycleover}]$, verifies if the aforementioned two equations are satisfied in the event of admission of a new request. This formulation is versatile: for example, an aggressive admission control can be realized by using higher buffer and cycle overload probabilities. Alternatively, a conservative admission control policy can be realized using smaller overload probabilities. We can also construct a service with multiple levels of QOS. Specifically, we can consider different values for 2-tuples $[P_{buffover}, P_{cycleover}]$ to represent different classes of clients. We order such classes in increasing order of overflow probabilities. Thus, a client belonging to class C_1 can tolerate smaller overflow probabilities than the client belonging to the class C_5. In other words, $P_{buffover,1} < P_{buffover,5}$ and $P_{cycleover,1} < P_{cycleover,5}$. In the event of overloads, the service is discontinued for clients from higher classes. Thus, the server can overbook its resources and on an average admit larger number of customers.

In the above formulation of performance of statistical multiplexing, Equation 1 depends on the chunk size used by the layout over the nodes, whereas Equation 2 depends on how the chunks are stored at each node, that is on the node level data layout. The node level data layout depends on the local storage devices at the node. We will assume that each storage node has a disk array, which can be of two types: one, an ordinary disk array in which all disks are used to store the data and the second, called Redundant Array of Inexpensive Disks (RAID) [26], in which some of the disks are used to store redundant information to improve reliability and data availability in the event of disk failures. When using commercial RAID devices, as we intend to do, the node level striping scheme is fixed to the one enforced by the RAID device. Given this, the chunk size happens to be the only design parameter that affects the efficiency of statistical multiplexing. The larger the chunk size, the smaller is the variability of the chunk size and offered load. Also, large chunk sizes amortize seek and rotational latency overhead at the disks in a disk array over large data transfers and thus, improve disk utilization. On the other hand, for good subjective quality of *ff* by chunk skipping, a small chunk size is desirable. In the presence of a large number of concurrent clients, small chunk size minimizes per connection buffer requirements. Also, a smaller chunk size allows quick response time for interactive playout control operations.

To summarize, *given a chunk size and a node specific data layout, we need to characterize the variation of overload probabilities at a node with the number of admitted clients. Using this characterization, we need to design, analyze and evaluate statistical admission control algorithms.*

4.2.2 Supporting diverse access models

In our work, we have assumed the Dedicated Viewing (DV) service model. However, several research efforts have attempted to develop multicast based service models that sacrifice unlimited interactivity for dynamic request aggregation and multicasting to achieve economies of scale [1, 22]. Such service models need to concurrently co-exist with the other services such as SV and SVC services. Also, the server may have to support advance reservations with specified QOS. However, such coexistence of various service models has implications on resource reservation, admission control, network signaling, scheduling and implementation of playout control operations.

4.3 Storage Node Design

The capability of our architecture to provide massive parallelism and support large number of concurrent accesses strongly depends on the functionality provided at the storage nodes.

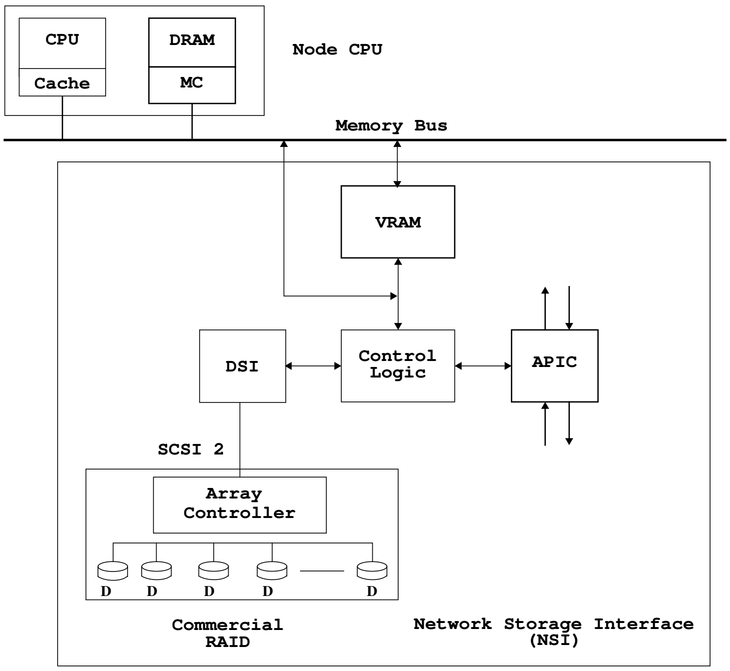

Figure 6: Storage Node Design

The main objective in the design of the storage node is to provide a high bandwidth, efficient data path between the external network and the storage devices. In our first prototype architecture, the storage at the nodes will be in the form of commercial disk arrays such as RAIDs, whereas the network interface will be in the form of the APIC chip. In order to understand the requirements that have to be satisfied in the design of such a storage node, we examine the data and control path within the storage node.

The data is typically prefetched and stored in a buffer before it is transmitted on to the network. The commercial RAIDs commonly provide up to 40 MBps or 320 Mbps throughput. The APIC is being designed for 1.2 Gbps throughput per port. Thus, in the worst case, the total throughput in and out of the buffering system must be of the order of 2 Gbps. However, memory systems constructed out of commercial DRAMS can provide a peak bandwidth of 500 Mbps. Given this, data copying must be avoided and alternate high bandwidth memory alternatives/structures must be explored.

A rough estimate of the size of such a memory system is also in order. If the average BW requirement of an MPEG stream is assumed to be 5 Mbps, the interconnect BW of 1.2 Gbps can accommodate roughly 240 clients. If the storage node prefetched 1 GOP of typically 9 frames, each connection prefetch buffer is roughly 0.2 MBs, and over all 240 connections it is about 48 MBs. Clearly, increasing this prefetch granularity increases the required buffer size. Also, as much as twice the size of prefetch buffers may be required to ensure smooth playout and playout control operation. In short, the data path must provide high capacity of the order of a few hundred megabytes and must sustain throughput of the order 2 Gbps.

The control path is responsible for admission control as well scheduling of prefetches from disk arrays and transmissions by the APIC. These tasks involve allocating/deallocating the data buffers, and scheduling related data structures, periodically updating scheduling information, and retrieving and managing the metadata. Such operations will typically be implemented in software as a part of a node level operating system and will require periodic timely access to the memory, storage and network system to guarantee overall correctness of scheduling operations. Clearly, in addition to the aforementioned data path throughput, this added load must be sustained.

Finally, as much as possible, off-the shelf components and commercially available operating systems should be used to construct the storage node hardware and software system.

In short, our primary hardware design task is to demonstrate that a storage node that meets the above requirements can be built. Similarly, the software design involves identification, design and implementation of the file system, buffer management, and scheduling functions required by the node operating system.

5 Preliminary Solutions and On-going Work

In this section, we illustrate solutions to the open issues described earlier and present on-going work. We will first describe analytical results on the distributed data layouts that facilitate efficient implementation of interactive operations such as *ff* and *rw*. We will then illustrate the distributed scheduling frame work that efficiently implements various playout control operations. We plan to use simulation and actual measurements to evaluate various tradeoffs. Specifically, we are developing a simulation tool called *MARSSim* is described in the Appendix. Also, we will use an existing SPARC workstation equipped with a commercially available RAID array as an experimental platform for measurements we propose to perform.

5.1 Load Balance Properties of Data Layouts

In order to efficiently support interactive operations such as *ff* and *rw*, it is necessary to know the safe skipping distance supported by a distributed layout. To this end, we have characterized a load balance property of distributed cyclic layouts described earlier. We define a set of D frames to be load-balanced if the set of nodes from which these frames are retrieved contains each of the D nodes only once. The set of nodes from which the frames are retrieved is also termed as a balanced node set. The following theorem illustrates load balance property of a DCL_1 layout[2].

Theorem 1 *Given a* DCL_1 *distributed layout over* D *storage nodes, the following holds true:*

- *If the fast forward (rewind) distance* d_f *is relatively prime to the number of storage nodes* D*, then*

[2]The result was first pointed out, in a different form, by Dr. Arif Merchant of the NEC Research Labs, Princeton, New Jersey, during the first author's summer research internship.

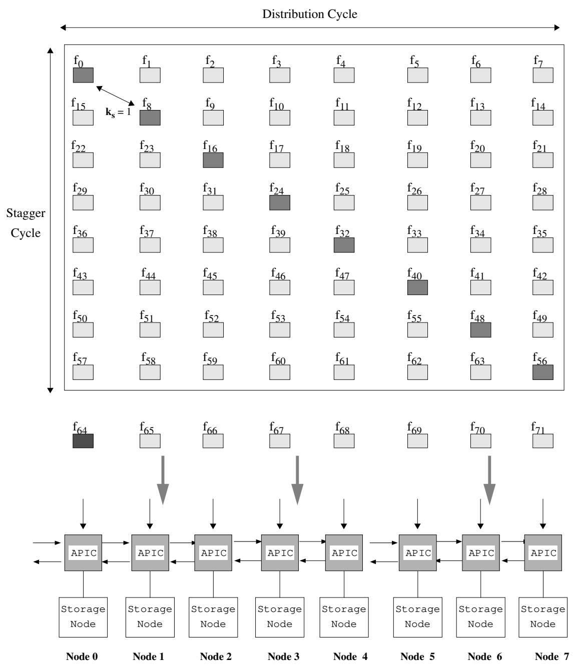

Figure 7: Staggered Distributed Cyclic Layout(SDCL) with $k_s = 1$

1. *The set of nodes S_n from which consecutive D frames in fast forward (rewind) frame set S_f are retrieved is load-balanced.*
2. *The fast forward (rewind) can start from any arbitrary frame (node) number.*

Thus, if $D = 6$, skipping by all distances that are odd numbers $(1, 5, 7, 11 \ldots)$ and are relatively prime to 6 will result in a balanced node set. We can see that if D is a prime number, then all distances d_f that are not multiples of D produce a balanced node set.

Since, a DCL$_1$ layout over a given number of storage nodes allows only a certain set of *ff* distances, additional data layouts that can increase the choice of *ff* distances are required. To this end we will describe a more general layout called *Staggered Distributed Cyclic Layout* (SDCL). Figure 7 illustrates an example of such a layout.

We define a distribution cycle in a layout as a set of D frames in which the first frame number is an integral multiple of D. The starting frame in such a cycle is called an anchor frame and the node to which it is assigned is called an anchor node. In the case of a DCL$_1$ layout, the anchor node for successive distribution cycles is always fixed to node 0. On the other hand, for the layout in Figure 7 the anchor node of the successive distribution cycles is staggered by one node in a $(\bmod\ D)$ order. This is an example of a staggered layout with stagger factor of $k_s = 1$. Other SDCL layouts with a non-unit stagger distance are possible. The following theorem illustrates the load-balance properties of SDCL with $k_s = 1$ [4]. Note that DCL$_1$ is a special case of SDCL layout with the stagger distance of $k_s = 0$.

Theorem 2 *Given a SDCL layout with $k_s = 1$ over D storage nodes and numbers $d_1, d_2, d_3, \cdots d_p$ that are factors of D, the following holds true:*

- **Load balance condition for fast forward:** *If the fast forward starts from an anchor frame with a distance d_f, then the node set S_n consisting of nodes from which D frames in the fast forward frame set S_f are retrieved is load-balanced, provided:*

1. *$d_f = d_i$ (where $1 \le i \le p$) or*

2. $d_f = m \times D$ *where* m *and* D *are relatively prime or*
3. $d_f = d_i + kD^2$ $(k > 0)$

- *The same result holds true for rewind if the rewind starts from a frame* $2D - 1$ *after the anchor frame.*

Thus if $D = 16$, then $d_f = 2, 4, 8, 16$ are "safe" skipping distances. We have developed proofs for both Theorem 1 and Theorem 2. Also, we have explored generalized SDCL with arbitrary stagger distance k_s and have conjectured that result similar to above result is possible. Specifically, if the the stagger distance k_s is relatively prime with the number of nodes D, then a result similar to Theorem 2 is possible. At present, we are working towards validating and formally proving this conjecture.

5.1.1 Interactive Control of MPEG Documents

In Section 4.1.2 we described problems associated with *ff* and *rw* on MPEG streams. Two ways to rectify these problems are as follows:

- **Store an intra-coded version of the movie along with a interframe coded version:** This option offers unlimited fast forward/rewind speeds, however, it increases the storage and throughput requirement. There are three optimizations possible that can alleviate this problem to some extent: 1) Reduce the quantization factor for the intra-coded version, but this may lead to loss of detail. 2) Reduce the display rate, however, this may cause jerkiness. 3) Store the spatially down-sampled versions of the frames, that is, reduce the resolution of the frames. This requires the frames to be up-sampled in real-time using up-sampling hardware. We believe that all three optimizations will be necessary, especially at high *ff* rates, to keep the network and storage throughput requirements unaltered.

- **Use the interframe coded version but instead of skipping frames skip chunks:** In this option, the skipping granularity is increased to chunks instead of frames, and chunks are transmitted from the individual storage nodes similar to the normal play mode (but in a different order). The load balance results illustrated earlier are valid for chunk skipping, with the frames skipping distance d_f replaced by chunk skipping distance d_{cf}. This option has the advantage that it keeps the storage and throughput requirements unchanged from the normal play, however, it may prove to be visually unappealing, especially with large chunk sizes.

We plan to use the software MPEG player and/or a settop box device capable of real-time decoding of MPEG along with display equipment to experiment with these options. Also, we will use the *MARSSim* simulator discussed in the appendix to study variation in the network and storage bandwidth for two options of *ff* and *rw* discussed earlier. The results from the simulation will be validated by performing measurements on our experimental platform that will store actual MPEG documents.

5.2 Design of Basic Distributed Scheduling Scheme

In this section, we illustrate the basic scheme and data structures used to schedule periodic data retrieval and transmission from storage nodes. Note that in addition to this scheme, each storage node has to schedule reads from the disks in the disk array and optimize disk head movements.

In a typical scenario, a client sends a request to the server to access a multimedia document. This request is received and processed by the central manager at the server. Specifically, the central manager consults an admission control procedure which based on current resource availability admits or rejects the new request. If the request is admitted it establishes a network connection to the client with appropriate QOS and informs the storage nodes of this new connection. Each storage node updates its scheduling information and allocates appropriate buffers. If an active client wants to perform a playout control operation, it sends a request to the server. The central manager receives it, and in response, changes the transmission and prefetch schedule. This will add a latency of one cycle in the worst case [3].

The global schedule consists of two concurrent and independent cycles: the "prefetch cycle" and the "transmission cycle", each of length T_c. During the prefetch cycle, each storage node retrieves and buffers data for all active connections. In the overlapping transmission cycle, the APIC corresponding to the node transmits the data retrieved in the previous cycle, that is, the data transmitted in the current i^{th} cycle is prefetched during previous $(i-1)^{th}$ cycle. Each storage node and associated APIC maintain a pair of ping-pong buffers, which are shared by the C_a active connections. The buffer that serves as prefetch buffer in current cycle is used as a transmission buffer in the next cycle and vice-versa. The APIC reads the data for each active connection from the transmit buffer and paces the cells, generated by AAL5 segmentation, on the ATM interconnect and to the external network, as per a rate specification. Note that the cells for all active connections are interleaved together.

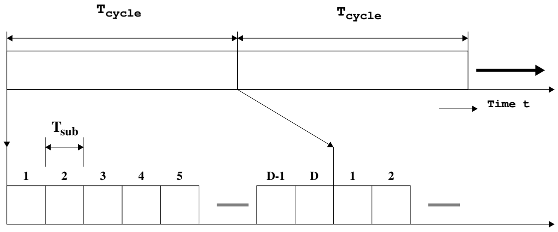

Figure 8: Cycle and Subcycle

Figure 8 and Figure 9 illustrate the mechanics of transmit scheduling. As shown there, the global transmission cycle consists of D identical sub-cycles, each of time length $\frac{T_c}{D}$. The end of a sub-cycle is indicated by a special control cell sent periodically on the APIC interconnect. The APIC associated with the central manager reserves a multicast control connection, with a unique VCI, that is programmed to generate these control cells at a constant rate $\frac{T_c}{D}$. Each of the remaining APICs in the interconnect, copies the cell to the storage node controller and also multicasts it downstream on the APIC interconnect. Each storage node counts these cells to know the current sub-cycle number and the start/end of the cycle.

The two important data structures, the **Prefetch Information Table** (PIT) (illustrated in Figure 10) and the **Slot Use Table (SUT)**, shown in Figure 9, are used by each node in the prefetch and transmission scheduling respectively. Each entry in the PIT provides prefetch information for some connection $VCI = k$. For example, in Figure 10 the entry corresponding $VCI = 2$ describes that in the current prefetch cycle, two (No_of_frms = 2) frames with $FrameID = 4, 5$ need to be prefetched using the $FrameAddr_1, FrameAddr_2$ into the buffers described by the $BuffDescr_2$.

On the other hand, the i^{th} entry in the SUT table lists the set of VCIs for which data will be transmitted in the i^{th} sub-cycle. This table is computed by the central manager at the start of each cycle and transmitted to each storage node. The computation uses simple modulo arithmetic and load-balance conditions described earlier.

Plan of Action for On-going Work:

- **Extensions:** In the above description, the storage nodes followed a frame-by-frame data prefetch and transmission policy and the clients were assumed to be bufferless. In our on-going work, we will extend this basic scheme to support other desirable features. Specifically, the cases of buffered clients, heterogeneous connections with different display rates, and non real-time accesses will be considered.

- **Tackling overloads:** Note that since prefetch and transmit cycles are interdependent, if the prefetch cycle overflows due to overload, corrective action must be taken to avoid mis-ordered transmission. The two alternatives are: 1) delay the start of the transmit cycle till overload subsides. This, however, affects all

[3]A cycle is typically a few seconds or a few hundreds of milliseconds duration.

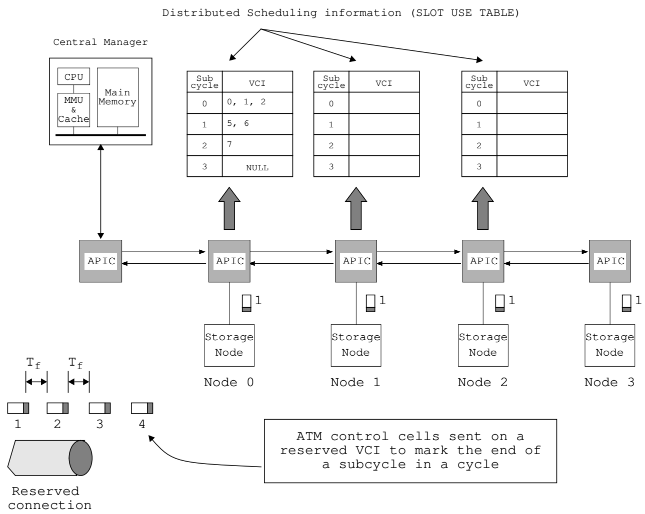

Figure 9: Distributed scheduling

PIT

Connection	Prefetch Information
VCI_1	No_of_frms = 1, Frame ID = 8, $FrameAddr_1$, $BufferDescr_1$
VCI_2	No_of_frms = 2, Frame ID = 4,5, $FrameAddr_1$, $Addr_2$,$BufferDescr_2$
VCI_3	No_of_frms = 1, Frame ID =1000, $FrameAddr_3$, $BufferPtr_3$
VCI_{100}	No_of_frms = 1, Frame ID =8500, $FrameAddress_{100}$, $BufferDescr_{100}$

Figure 10: Prefetch Information

connections, some of which may already have valid prefetched data. 2) Delay transmissions only for those connections that are affected by the overload events. This, however, requires additional message exchange between the nodes and complicates the scheduling scheme. We are at present investigating both these options in greater detail and will incorporate support for the appropriate option in the final design.

- **Detailed algorithms:** We are at present developing the algorithms that manipulate the scheduling tables and per connection state information to effect normal play and any playout control operations. These algorithms will be developed for the most general case of heterogeneous clients.

- **Experimentation:** We will implement above algorithms in our *MARSSim* simulator to emulate the scheduling scheme described above and verify its correctness. We will also implement the modifications required to tackle the overloads and measure the message exchange overhead for each option as the load is increased. Specifically, fraction of the scheduling cycles that overflow and average message exchange for each overflow cycle will be characterized.

We plan to implement the prefetch and transmit scheduling described above in the NetBSD operating system on our experimental platform with a disk array that will store real-life MPEG data sets. We will measure prefetch and transmit cycle overload probabilities for increasing loads and compare the same obtained from our simulation measurements.

5.2.1 Admission Control

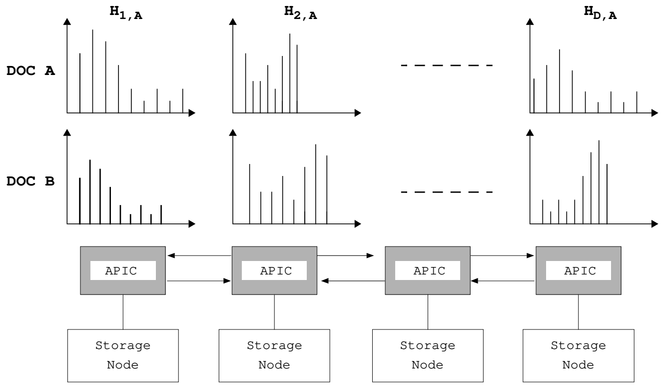

Figure 11: Approach to Admission Control

Section 4 provided a formulation of the admission control problem. The computation of the buffer overload ($P_{buffover}$) (Equation 1) and the cycle overload probability ($P_{cycleover}$) (Equation 2) described there need prior knowledge of probability distribution (pdf) of the size of data being accessed. We propose that when a document is stored on the server, the histogram of the sizes of the frames assigned to each storage node be computed and made available. Such histograms approximate pdf of data units retrieved for a document at each node. Thus, as shown in Figure 11, Document A has histograms $H_{1,A}, H_{2,A}, \cdots H_{D,A}$ associated with it at nodes $0, 1, \cdots D-1$ respectively. Given these histograms, the pdf distribution of data unit of size k frames retrieved at a node is obtained by convolving the histogram k times. However, such computation assumes that the frames are completely uncorrelated, which is not quite true in case of MPEG. Hence, we may have to account for temporal correlation among frames using autocorrelation statistics. If such statistics are needed, they will be pre-computed and stored at the server.

When new clients are admitted, the Equations 1 and 2 are computed on-line using the knowledge of the existing load, the tolerable overload probabilities, and the histogram and other statistics of the requested document. Note that in these equations pdf distributions of $\sum_{i=1}^{n} D_i$ and $\sum_{i=1}^{n} T_r(D_i)$ are computed incrementally. For example, if the number of currently active connections is n and a new $(n+1)^{th}$ request is received, the computation of pdf of $\sum_{i=1}^{n+1} D_i$ uses the histogram of the requested document and incrementally computes pdf $\sum_{i=1}^{n} D_i$. If the $(n+1)^{th}$ client is admitted, the nodes remember $\sum_{i=1}^{n+1} D_i$ and $\sum_{i=1}^{n+1} T_r(D_i)$ to process any new incoming requests. Also, whenever a client session completes, the value of these variables is recomputed.

Plan of Action for On-going Work

- **Design:** We are at present working on the design and evaluation of this algorithm. We will estimate the complexity and the storage overhead associated with the a-priori computation of statistics of MPEG streams required by this algorithm.

- **Experimentation:** We plan to implement this on-line admission control scheme in the *MARSSim* simulator. We will perform a set of experiments in which we will always admit a connection. The admission criteria will be used only to guess the overload probability. The actual measured overload probabilities will then be compared with the predicted probabilities. The number of extra clients admitted and the QOS guarantees supported, will characterize the effectiveness of such an aggressive admission procedure.

Also, we will, repeat above experiments on our target platform and compare the results with those from the simulations. Such a comparison will illustrate implications of some of the system components, such OS overheads, that are not modeled in simulations, on QOS guarantees.

5.3 Design, Prototyping and Measurements

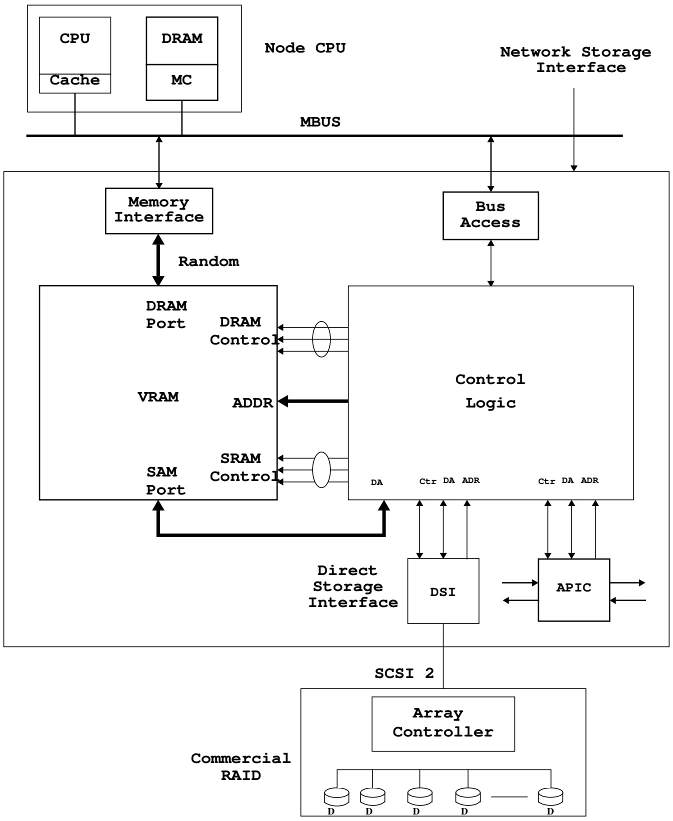

Figure 12: Storage Node Design

Figure 12 illustrates the basic design of the storage node, which employs a dual ported memory called Video-RAM (VRAM), with a serial access port and a traditional random access port. The commercially available VRAMs typically have a serial access time of the order of 20 ns. Thus, ignoring the refresh and setup times, the SAM port with data path width of 64 bits can sustain approximately 3 Gbps. Also, the DRAM port when operated in fast-page mode supports roughly 1.5 Gbps. Thus, commercial VRAMs do provide the magnitude of throughput we mentioned in Section 4.3.

The Network Storage Interface (NSI) block shown in Figure 12, controls three different interfaces: a standard random access memory interface to the CPU-memory bus, a set of peripheral interfaces, and a network interface, specifically APIC in our case. The control logic in the NSI arbitrates over the CPU accesses to the random access port of the VRAM, and to the memory mapped control registers of the APIC and the Direct Storage Interface (DSI) block. It also controls the accesses of the DSI and the APIC to the sequential access port of the VRAM.

The Direct Storage Interface (DSI) block shown in Figure 12, provides two functions: a data transfer block (like DMA), and one or more peripheral bus interfaces such as the *Small Computer Systems Interface* (SCSI). The commercially available storage products like disk arrays, RAIDs, tape libraries, with compatible interface can be connected to the storage node using the peripheral interface. The DSI receives the read (write) requests and using this interface, retrieves (transfers) the data from (to) the physical devices and transfers it directly to (from) appropriate buffers allocated in the VRAM. Several commercial chips, such as NCR53C90, NCR53C810, NCR53C720, that interface to SCSI bus on the device side and to the standard CPU–memory buses or to standard I/O buses such as Sun's S-bus, the Intel's PCI bus on the host side, can be used in the design of DSI.

Note that the prefetch and transmit buffers, and their associated buffer descriptors are allocated in the VRAM. The per connection state information in the form of Connection State Block (CSB) and the scheduling information such as the SUTs, and the PITs are also allocated in the VRAM.

Typical Operation

In a typical scenario, the central manager (Figure 2) receives a client access request and informs each node of this request. If the request is for real-time data, each node uses the admission control procedure outlined earlier to decide if the request can be admitted. Using a special control message, each node informs the central manager of the decision and the QOS in the form of overload probabilities that the node can guarantee. The central manager makes the final admission decision based on the control messages from all the nodes. In the event the request is accepted, the central manager uses appropriate network signaling protocols to establish a new connection between the server and the client, and informs each node of this new connection. Specifically, it notifies each node, the connection identifier (VCI) and the QOS it supports.

In response to the connection notification, each node sets up the appropriate data and control path. The data path consists of the storage devices, the DSI block, the data buffers in the VRAM, and the APIC. The control path involves the node level CPU and the associated operating system that uses the memory mapped register interface to control the APIC and the storage devices through the DSI. In order to set up the control path functions for the new connection, the node creates appropriate state information such as the Connection State Block (CSB), and initializes the scheduling data structures such as SUT and PIT. It also uses the local operating system services to set up data path by allocating an estimated amount of buffers.

The flow of data along the data path results from prefetch and the transmit activities co-ordinated by the control path. These activities are summarized below: Each node periodically receives a "slot/sub-cycle" marker cell which indicates the start of a sub-cycle, typically, every 33 msecs. The interrupt routine that processes this control ("marker") cell counts the sub-cycles in the current cycle. Using this number and the SUT and PIT data structures, the node operating system controls the prefetch and transmit activities.

If the current sub-cycle is the first in the cycle, the node computes the new entries in the SUT for the present cycle. Also, it checks the validity of the prefetch buffers filled in the previous cycle and then, updates the entries in the PIT table. The new PIT entries record the metadata and the buffer descriptors for the data to be fetched in the current cycle. Using the metadata information in the PIT entries, prefetch requests are generated for the device driver. The data so retrieved from the storage devices are stored into the buffers described by the buffer descriptors.

Using the sub-cycle number and the SUT data structure, the node decides which connections will be active during the current sub-cycle. In case of the APIC, a connection is active if there is a valid buffer descriptor queued for transmission. Thus, activating a connection simply involves queuing a valid prefetch buffer that has been already filled in the previous cycle. The APIC reads the data from these buffers, segments it using the AAL5 algorithm and transmits the ATM cells using the rate specification computed at the start of each cycle.

Note that during the prefetch and transmit cycles a storage node may have to process significant number of interrupts. Specifically, each "marker" cell received causes an interrupt which requires processing with the highest priority. Also each completed prefetch request to disk array generates an interrupt. For example, in a simplistic scheme if the cycle length is 500 msecs and the number of active clients is 200, on an average the disk array will interrupt the node CPU every 2.5 msecs. In addition, the "marker" cell interrupts will be received every 33 msecs.

The asynchronous nature of such interrupts and the context switching and processing overhead associated with them, often proves to be a hurdle in providing predictable deterministic QOS guarantees. Hence, ways to minimize interrupts must be explored. For example, if the DSI can keep track of the completed requests and interrupt the node CPU at the end of the prefetch cycle, the prefetch interrupts are reduced to only one per cycle.

Plan of Action for On-going Work:

- **Storage Node Design:** We will develop a detailed chip level design of the DSI and the control logic in the NSI. We will investigate suitability of alternate memory components such as synchronous DRAMs and RAMBUS in building high throughput storage node. We will employ VHDL simulation to verify the correctness of our design. If the APIC chip becomes available, we will attempt to demonstrate a working storage node subsystem.
- **OS Modifications:** We plan to modify the NetBSD file system to support the data and metadata separation, and the chunked data layout. As mentioned earlier, in presence of a large number of active clients, the context switching overhead and the asynchronous nature of the interrupt driven prefetch scheduling makes it difficult to provide QOS guarantees. In order to minimize such overhead and exploit inherent periodicity in scheduling, we will modify the process scheduling in NetBSD, using techniques proposed in [27]. We will implement node level prefetch and transmit scheduling using these modifications and compare its performance with that of an interrupt driven implementation.

  Since, multimedia data requires large buffers, we may have to investigate into suitability of NetBSD buffer system and consider alternate buffer management systems such as the UCMIO buffer system [12].

6 Related Work

High performance I/O has been a topic of significant research in the realms of distributed and supercomputing for quite sometime now. In recent years, the interest in integrating multimedia data into communications and computing has lead to a flurry of activity in supporting high performance I/O that satisfies special requirements of this data. Here we summarize some notable efforts.

6.1 High Bandwidth Disk I/O for Supercomputers

Salem et al. [29] represents some of the early ideas on using disk arrays and associated data striping schemes to improve effective storage throughput. Observing that large disk arrays have poor reliability and that small disks outperform expensive high-performance disks in price vs. performance, Patterson et al. [26] introduced the concept of RAID. A RAID is essentially an array of small disks with simple parity based error detection and correction capabilities that guarantee continuous operation in the event of a single disk failure in a group of disks. The RAID was expected to perform well for two diverse types of workloads. One type, representative of supercomputer applications such as large simulations, requires infrequent transfers of very large data sets. The other type, commonly used to characterize distributed computing and transaction processing applications, requires very frequent but small data accesses [26]. However, measurements on the first RAID prototype at the University of California, Berkeley revealed poor performance and less than expected linear speedup for large data transfers [11]. The excessive memory copying overhead due to interaction of caching and DMA transfers, and restricted I/O interconnect (VME bus) bandwidth were cited to be the primary reasons of poor performance. Also, it is recognized now that large RAID disk arrays do not scale very well in terms of throughput.

The recent work on RAID-II at the University of California, Berkeley has attempted to use the lessons learned from the RAID prototype implementation to develop high bandwidth storage servers by interconnecting several disk arrays through a high speed *HIPPI network backplane* [21]. It's architecture is based on a custom board design called *Xbus Card* that acts as a multiple array controller and interfaces to HIPPI as well as to FDDI networks. Though the measurements on RAID-II have demonstrated good I/O performance for large transfers, the overall solution employs FDDI, Ethernet and HIPPI interconnects and is ad-hoc. Also, it has not been demonstrated to be suitable for real-time multimedia, where the application needs are different than the needs of supercomputer applications.

6.2 Multimedia Servers

A significant amount of research has attempted to integrate multimedia data into network based storage servers. However, most of it has addressed different dimensions of the problem such as operating system support, file systems design, storage architecture, metadata design, disk scheduling etc. in an isolated fashion. Here we will categorize and summarize some of the notable research efforts.

6.2.1 Multimedia File Systems

One of the early qualitative proposals for a on-demand video file system is reported in Sincoskie [30]. The work by Rangan et al. [28, 33] developed algorithms for constrained data allocation, multi-subscriber servicing and admission control for multimedia and HDTV servers. However, this work assumes an unrealistic single disk storage model for data layout. It is worth repeating that such a model is inappropriate, as the transfer speed of a single disk will be barely sufficient to support a single HDTV channel and is about three orders of magnitude lower than that required to support a thousand or more concurrent customers independently accessing the same data.

Some of the recent work by Vin et. al. [34, 35] focuses on developing statistical admission control algorithms for disk array based server capable of deterministic and/or statistical QOS guarantees.

6.2.2 Storage Server Architecture

Hsieh et al. [18] evaluate a high performance symmetric multiprocessor machine as a candidate architecture for a large scale multimedia server. They report measurements carried out on a fully configured Silicon Graphics high end symmetric multiprocessor workstation called SGI ONYX. They present two data layout techniques called *Logical Volume Striping* and *Application Level Striping* that use multiple parallel RAID 3 disk arrays, to increase concurrency and parallelism. The focus of their work until now has been on characterizing using measurements the upper limit in terms of maximum number of concurrent users in different scenarios, such as various type of memory interleaving, various type of striping schemes and multiple processes accessing a single file or different file.

Similar to our work, they propose a three level layout: the lowest level is the RAID level 3 byte striping. The second level called *Logical Volume striping* allows data to be striped over multiple RAIDs belonging to a logical volume. The last level, called the *Application Level Striping*, allows applications to stripe data over multiple logical volumes. The data layout reported in this paper uses data units of size 32 KB, based on the assumption that each video frame is of that size.

The platform used in their measurements featured: eight 20 MIPS processors, upto 768 Mbytes of multi-way (4-way or 8-way) interleaved RAM, and upto 30 RAID-3 disk arrays. The platform employs a very fast system bus (about 1.2 GBps) and an I/O bus (about 310 MBps). Thus, the prototype architecture to be used as a VOD server is an expensive high end parallel processing machine.

All the measurements reported in [18] assume that the clients are homogeneous and perform only normal playout. In other words, cases when a subset of active clients are performing interactive playout control or require different display rates have not been considered. Also, even though the authors recognize that they will require some real-time scheduling to optimize the disk head movements and retrievals from different disks, this particular paper does not report any details on this topic. Similarly, buffer management, scheduling, network interfacing and admission control have been mentioned to be important topics, but have not been investigated.

Lougher et al. have reported a design of a small scale *Continuous Media Storage Server* (CMSS) that employs append-only log storage, disk striping and hard real-time disk scheduling [24]. Their transputer based implementation handles very few simultaneous customers and supports small network bandwidth. This implementation is clearly not scalable for large number of users and for high bandwidth streams such as HDTV.

Tobagi et al. [31] report a small scale PC-AT and RAID based video server. A commercial continuous video server called *Shark* [17] is currently being marketed by IBM. Similarly, Kandlur et al. [19, 36], describe a disk array based storage server and present a disk scheduling policy called Grouped Sweeped Scheduling to satisfy periodic requirements of multimedia data. In a related paper [10], they discuss load balance issues in implementation of *ff* and *rw* in such a server. However, all these server proposals are PC based and thus not scalable. Also they do not support high parallelism and concurrency.

To summarize, all the proposals mentioned above address the problems of scheduling, data layout, and meta-data design in an isolated fashion and none describe a system architecture that is scalable in terms of throughput and large number of clients. We believe that our research is the first to address the problem of large scale server design by taking an integrated approach that collectively addresses all these issues.

7 Conclusions

We presented an efficient scalable architecture for a high performance large scale multimedia server. We also described distributed data layout and scheduling schemes that are aimed at supporting normal playout as well as interactive playout operations with tolerable latency for a large number of clients. We highlighted several research questions associated with these schemes that need to be explored through simulation and/or analytical modeling. Also, we identified the design of hardware and software systems in the storage node in our architecture as a key component of our research.

The primary outcome of our proposed research will be design and detailed evaluation of distributed data layout, scheduling, interactive playout control, and statistical admission control techniques for high performance large scale servers such as MARS. The hardware and software design component of the proposed work will also lead to a detailed blue-print for prototyping of a large scale multimedia on-demand server that will be deployed in the local gigabit ATM testbed.

References

[1] Almeroth, K., and Ammar, M., "The Role of Multicast Communication in Provision of Scalable and Interactive Video-on-Demand Service," *Proceedings of* 5^{th} *International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'95)*, pp. 267-270, Durham, New Hampshire, April 19-21, 1995.

[2] Buddhikot, M., Parulkar, G., and Cox, Jerome, R. Jr., "Design of a Large Scale Multimedia Storage Server," invited for publication in the *Journal of Computer Networks and ISDN Systems*, Elsevier's North Holland Publishers. Shorter version appeared in the *Proceedings of INET'94/JENC5, Conference of the Internet Society and the Joint European Networking Conference,* Prague, June, 1994.

[3] Buddhikot, M., Parulkar, G. and Cox, J., "Distributed Scheduling, Data Layout and Playout Control in a Large Scale Multimedia Storage Server," Technical Report WUCS94-33, Department of Computer Science, Washington University in St. Louis, Sept. 1994.

[4] Buddhikot, M., Parulkar, G. and Cox, J., "Efficient Data Layout, Scheduling and Playout Control in a Large Scale Multimedia Storage Server," *Proceedings of* 5^{th} *International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'95)*, pp. 339-350, Durham, New Hampshire, April 19-21, 1995.

[5] Cao, Pei, et al., "The TickerTAIP Parallel RAID Architecture," *Proceedings of the 1993 International Symposium on Computer Architecture*, May 1993.

[6] Chang, Y., et al., "An Open Systems Approach to Video On-demand," *IEEE Communications Magazine*, pp. 68-80, May, 1994.

[7] Chang, Ed, and Zakhor, A., "Admissions Control and Data Placement for VBR Video Servers," Proceedings of the First International Conference on Image Processing, November, 1994.

[8] Chang, Ed, and Zakhor, A., "Variable Bit Rate MPEG Video Storage on Parallel Disk Arrays," First International Workshop on Community Networking, San Francisco, July 1994.

[9] Chen, P., et al., "RAID: High-performance, Reliable Secondary Storage," *ACM Computing Surveys*, Vol. 26, No. 2, pp. 145-185, June 1994.

[10] Chen, M., Kandlur, D., and Yu, S. P., "Support for Fully Interactive Playout in a Disk-Array-Based Video Server," *Proceedings of Second International Conference on Multimedia, ACM Multimedia'94*, 1994.

[11] Chervenak, A, "Performance Measurements of the First RAID Prototype," Technical Report, Department of Computer Science, University of California, Berkeley, 1990.

[12] Cranor, C., Parulkar, G., "Universal Continuous Media I/O: Design and Implementation," Washington University in St. Louis, Department of Computer Science, Technical Report WUCS94-34, December 1994.

[13] Dey, J, et al., "Providing VCR Capabilities in Large-Scale Video Servers," To appear in *Proceeding of ACM Multimedia'94*, San Francisco, Sept, 1994.

[14] Dittia, Z., Cox., J., and Parulkar, G., "Catching up with the networks: Host I/O at gigabit rates," Technical Report WUCS-94-11, Department of Computer Science, Washington University in St. Louis, Apr. 1994.

[15] Dittia, Z., Cox., J., and Parulkar, G., "Design of the APIC: A High Performance ATM Host-Network Interface Chip," Proceedings of the IEEE INFOCOM'95, pp. 179-187, Boston, Apr. 1995.

[16] Dittia, Z., Cox., J., and Parulkar, G., "A Gigabit ATM Desk-Area Network: System Architecture Document for the ATM Port Interconnect Controller (APIC)," Draft Version 1.1, Washington University in St. Louis, July 1995.

[17] Haskins, R., "The *Shark* Continuous-Media File Server," *Proceedings of IEEE Compcon'93*, San Francisco CA, Feb. 1993.

[18] Hsieh, J., et al., "Performance of a Mass Storage System for Video-On-Demand," To appear in the *Proceedings of INFOCOM'95*, April, 1995.

[19] Kandlur, D., Chen, M., and Shae, Z., "Design of a Multimedia Storage Server," *Proceedings of SPIE Conference on High-speed Networking and Multimedia Computing*, Vol. 2188, San Jose, pp. 164-178, Feb., 1994.

[20] Lee, E., "Software and Performance Issues in the Implementation of RAID Prototype," Technical Report UCB/CSD 90/573, Department of Computer Science, University of California at Berkeley, May, 1990.

[21] Lee, E., et al., " RAID-II: A Scalable Storage Architecture for High-Bandwidth Network File Service," Technical Report UCB//CSD-92-672, Department of Computer Science, University of California at Berkeley, Oct., 1992.

[22] Venkatesh, D., and Little, T., D., C., "Dynamic Service Aggregation for Efficient Use of Resources in Interactive Video Delivery," *Proceedings of* 5^{th} *International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'95)*, pp. 119-122 , Durham, New Hampshire, April 19-21, 1995.

[23] Lam, S., Chow, S, and Yau, K., Y., D., "An algorithm for lossless smoothing of MPEG video," Proceedings of ACM SIGCOMM '94, London, August 1994.

[24] Lougher, P., and Shepherd, D., "The Design of a Storage Server for Continuous Media," *The Computer Journal*, Vol. 36, No. 1, pp. 32-42, 1993.

[25] Liu, J., Du, D., and Schnepf, "Supporting Random Access on Real-time Retrieval of Digital Continuous Media," To appear in the *Proceedings of INFOCOM'95*, April, 1995

[26] Patteson, D., et al., "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *Proceedings of the 1988 ACM Conference on Management of Data (SIGMOD)*, Chicago IL, pp. 109-116, June 1988.

[27] Raman, G., and Parulkar, G., "Real Time Signals; A Mechanism to Provide Real-time Processing Guarantees," Washington University in St. Louis, Department of Computer Science, Technical Report WUCS95-06, March, 1995.

[28] Rangan, V., and Vin, H., "Designing File Systems for Digital Video and Audio," *Proceedings of the* 13^{th} *Symposium on Operating System Principles*, Operating Systems Review, pp. 81-94, Oct. 1991.

[29] Salem, K., and Garcia-Molina, H., "Disk Striping," *IEEE International Conference on Data Engineering*, 1986.

[30] Sincoskie, W., "System Architecture for a Large Scale Video on Demand Service," *Computer Networks and ISDN Systems, North Holland*, Vol. 22, pp. 155-162, 1991.

[31] Tobagi, F., Pang, J., Baird, R., and Gang, M., " Streaming RAID - A Disk Array Management System for Video Files," *Proceedings of ACM Multimedia'93*, Anaheim, CA, pp. 393-400, Aug. 1993.

[32] Venkatramani, C., and Chiueh, T., "Survey of Near-Line Storage Technologies: Devices and Systems," Experimental Computer Systems Laboratory, Technical Report #2, Department of Computer Science, SUNY Stony Brook, NY, Oct. 1993.

[33] Vin, H., and Rangan, V., "Design of a Multi-user HDTV Storage Server," *IEEE Journal on Selected Areas in Communication*, Special issue on High Definition Television and Digital Video Communication, Vol. 11, No. 1, Jan. 1993.

[34] Vin, H., et al., "A Statistical Admission Control Algorithm for Multimedia Servers," *Proceedings of the ACM Multimedia '94*, San Francisco, October 1994.

[35] Vin, H., et al., "An Observation-Based Admission Control Algorithm for Multimedia Servers," *Proceedings of the IEEE International Conference on Multimedia Computing Systems (ICMCS'94)*, Boston, pp. 234-243, May 1994.

[36] Yu, P., Chen., M., and Kandlur, D., "Grouped Sweeping Scheduling for DASD based Storage Management," *Multimedia Systems*, Springer-Verlag, pp. 99-109, Dec. 1993.

APPENDIX I
MARSSim: A Simulation Tool

We are developing a simulator called *MARSSim* based on the *raidSim* simulator from University of California, Berkeley [20]. The *raidSim* is an event-driven simulator developed primarily to study performance of various RAID levels for mix of I/O loads commonly observed in transaction processing and supercomputing applications. The simulator consists of three main parts: a module for implementing variety of RAID levels specific data layouts, a module for modeling disk behavior (such as seek and rotational latencies and a module to generate I/O requests load. The simulator does not model disk controller and any system overhead. We plan to modify the simulator in following ways:

- **Additional striping schemes:** We will model non-redundant disk arrays with user specifiable striping schemes.
- **Modeling multiple disk arrays:** Since, the data layout involves multiple disk arrays at different storage nodes co-operating to supply a multimedia document, we will model several disk arrays simultaneously.
- **Modeling MPEG data storage:** We will use the MPEG trace data available in the public domain on the internet and the (approximate) statistical models for MPEG sources to model storage of data on disk arrays.
- **Scheduling:** We will add capabilities that will allow us model various scheduling algorithms for reading data from disk arrays and individual disks. Also, scheduling scheme for co-ordinating reads (writes) over the disk arrays will be modeled.

Since analytical modeling of MPEG streams as well as disk arrays is difficult, we plan to use this tool to study statistical multiplexing of MPEG streams at the storage node. Specifically, we will be able to answer questions such as:

- Given a mix of representative MPEG documents, a disk array with N disks and a node level striping scheme, how does the overload probability vary with the number of admitted clients and the chunk size.?
- How does the storage and network throughput requirement vary with different chunk sizes when performing *ff* or *rw* by chunk skipping.?
- Under various load conditions, what kind of delay guarantees are possible for non real-time data access?
- To what extent are the probabilistic guarantees made by the statistical admission control criteria met under various load conditions.?

The *MARSSim* will also help us verify correctness of our distributed scheduling scheme and implementation of various playout control operations.