Distributed Data Layout, Scheduling and Playout Control in a Large Scale Multimedia Storage Server

Milind M. Buddhikot, Guru Parulkar

WUCS-94-33

July 94

Department of Computer Science
Campus Box 1045
Washington University
One Brookings Drive
St. Louis, MO 63130-4899

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

1. Introduction

Large scale multimedia storage servers will be an integral part of a ubiquitous distributed multimedia infrastructure that is taking shape. The users of this infrastructure will have access to exciting applications such as multimedia mail, orchestrated presentations, high quality on-demand audio and video, collaborative multimedia document editing, browsing remote multimedia archives and virtual reality environments. The recognition of the fact that the existing network based servers cannot meet the requirements of these new applications, has led to a flurry of research activity in the area of design of high performance network based multimedia storage servers. To comprehend the challenges in designing such servers, it is important to understand their requirements. These include : 1) support potentially thousands of concurrent customers all accessing the same or different data, 2) support large capacity (in excess of terabytes) storage of various types, 3) deliver storage and network throughput in excess of a few Gbps, and 4) provide deterministic or statistical QOS guarantees in the form of bandwidth and latency bounds.

Given the trends towards multi-participant and interactive applications, in addition to the four basic requirements, a storage server must support interactivity. Interactivity can be defined as the ability of a client to control a multimedia stream delivered by the server. The stream control can be of two types: stream playout control and the stream content control. The stream playout control allows the user to perform operations such as *fast forward (ff)*, *rewind (rw)*, *pause*, *frame advance*, *slow play*, and *stop-and-return* on a media stream. On the other hand, the stream content control allows the client to modify the stream content by performing appropriate media edit functions. For example, in case of audio, video and image streams, a client may want to enhance the stream and store it back to the server. Note that for most of the on-demand applications, interactivity in the form of stream playout control will be sufficient. In other words, a retrieval environment that supports data retrieval but no data edits, will be more prevalent.

One of the greatest hurdles in designing high performance storage servers that meet all 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 keep improving faster than that of on-line secondary storage [5]. The mismatch between the requirements of multimedia and the traditional operating system support for secondary storage devices aggravates this bottleneck even further.

Given these observations, addressing the challenge of high performance storage I/O will be critical to the design of multimedia clients and servers. Our project called the *Massively-parallel and Real-time Storage* (MARS) represents an effort in this direction. The salient feature of our architecture is 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 sever, to achieve a scalable architecture that transparently connects storage devices to an ATM based broadband network. The ATM interconnect within this architecture uses a custom ASIC called *ATM Port Interconnect Controller* (APIC), which will serve as the basic building block for a high bandwidth *networked-*I/O *subsystem*. The APIC is currently being developed in an ARPA sponsored gigabit LAN testbed, whereas the APIC based MARS storage server will be designed and prototyped as a part of the recently initiated NSF funded *Grand Challenges* project. In our opinion, this research effort in storage server design is the first one to propose a system architecture that is scalable in terms of throughput and number of clients, and take an integrated approach that collectively satisfies all the requirements.

In this paper, we will consider only a retrieval environment and primarily focus on the strong interaction between the architecture, data layout, data compression, and scheduling. In particular, we will present distributed multilevel data layout, scheduling and playout control schemes developed in conjunction with our architecture. These schemes allow all clients to access the same data without data replication and support both buffered as well as bufferless clients. Also, they provide strict deterministic guarantees to each active client during normal playout as well as a full spectrum of interactive stream control operations (namely, fast forward, rewind, frame advance, slow play, slow rewind, pause, stop-and-return and stop). Our implementation of the stream control operations requires no extra bandwidth reservation and provides acceptable operation latency of a few hundread milliseconds.

The rest of this paper is organized as follows: Various service models that are possible for a on-demand multimedia server are illustrated in Section 2. The basics of our prototype implementation of a large scale server are presented in Section 3. Section 4 describes the distributed and hierarchical data layout scheme. Next, our basic multilevel scheduling scheme is illustrated in Section 5. Various ways of implementing playout control operations and their implications on scheduling are described in Section 6. This section also presents modifications that must be made in the basic scheduling scheme to achieve smooth transition between normal playout and operations such as *ff* and *rw*. Section 7 presents some interesting analytical results on distributed data layout. These results guarantee that no extra BW reservation is required, at the storage nodes in the architecture, to allow each active client to independently do *ff* and *rw*. Section 8 gives a brief overview of related research efforts in high performance I/O and multimedia storage servers, reported in literature. Finally, the conclusions from the present status of the work and the expected results of the ongoing work are outlined in Section 9.

2. Service Models for a On-demand Multimedia Storage Server

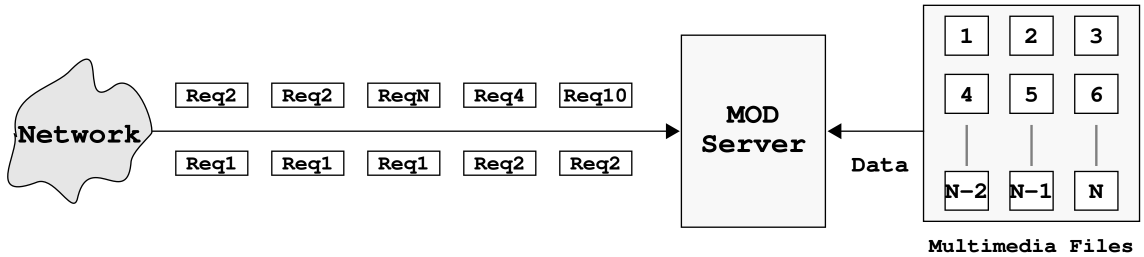

Figure 1: MOD server request-response model

Figure 1 shows the access model of a MOD server in a retrieval environment. Typically, the MOD sever will have N multimedia files, each of which has two attributes associated with it. The first attribute, duration, is the length of the time that would be required to play the file at normal playout rate, whereas the other attribute, demand, measures the number of simultaneous connections the server can support for the file. The clients can access the multimedia file (s) by sending requests to the server. The server performs admission control, which, based on the existing load and the resource usage, admits or rejects the client requests.

The request arrival process at a MOD server is likely to exhibit spatial and temporal locality. The spatial locality property refers to the fact that some of the data items are likely to be accessed more frequently than the others. For example, in case of an on-demand movie application, a large fraction of requests received are likely to be for popular and recently released movies. However, some applications may exhibit lesser spatial locality than the others, for example, in an on-demand digital library or a radiological image database, all documents are likely to be uniformly accessed.

The temporal locality property signifies that depending on the application, the request arrival process will be temporally clustered to various extent. For example, in case of on-demand movies, request rate will be higher during the evening time period of a weekend than on the weekdays, and more requests will be received during the 6 to 9 pm than any other period of the day.

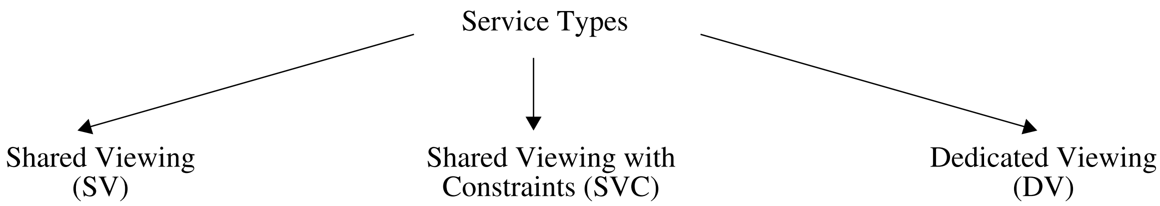

Figure 2: On-demand service types

Given such an access model and request arrival pattern, the extent to which various requirements outlined earlier in Section 1 are met by a storage server depends on the service model that underlies its design. Figure 2 illustrates various service models that are possible for an on-demand storage server. Of these, the shared viewing and dedicated viewing models have been discussed in literature, whereas the shared viewing with constraints model is a generalization of a service commonly known as "near-video" [14, 4]. Note that this classification is based on the degree of personalization and interactivity provided by a service.

Shared Viewing (SV) service:
In this type of service, the time of access for a multimedia document is decided by the server rather than the client. The MOD server multicasts the multimedia program at fixed times and the clients interested in receiving the program tune in to the server at those fixed times. Existing Pay-Per-View (PPV) channels on cable networks represent this service. Clearly, a client of such a service can not control the stream playout and also has minimal freedom as to when the multimedia stream will be available. In other words, the term "on-demand" in this model is quite a misnomer. Given that the existing shared media networks such FDDI support broadcast and that the future broadband networks, such as ATM, will support multicast, a MOD system offering SV service can be easily implemented by multi-casting data retrieved off a simple disk or even a digital tape. In short, implementing such a server presents hardly any technological challenges.

Shared Viewing with Constraints (SVC):
In this type of service, the client accesses the multimedia documents at any time it desires by sending request to the MOD server. The server processes and accepts requests in groups to exploit the clustering property of the arrival process. Typically, a new client request may face a variable admission latency, after which the client is assigned to a multicast group, all members of which are connected to the server by a single multicast network connection. The SVC represents incremental improvement over the SV service, as it allows user to access multimedia documents at nearly arbitrary instances, unlike the fixed instances in SV service. This service may make sense in case of certain applications, such as on-demand movies, where request arrivals are likely to exhibit significant temporal and spatial clustering. However, supporting interactive behavior is difficult in this service and hence, it will be unsuitable for applications such as digital library or database browsing, where request clustering is difficult to achieve and interactivity is important.

Dedicated Viewing (DV):
In this type of service, illustrated in Figure 3, the client accesses the multimedia documents by sending requests to the server. Each such request, from the same or different client, is treated independently at the server. For example, in Figure 3, 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

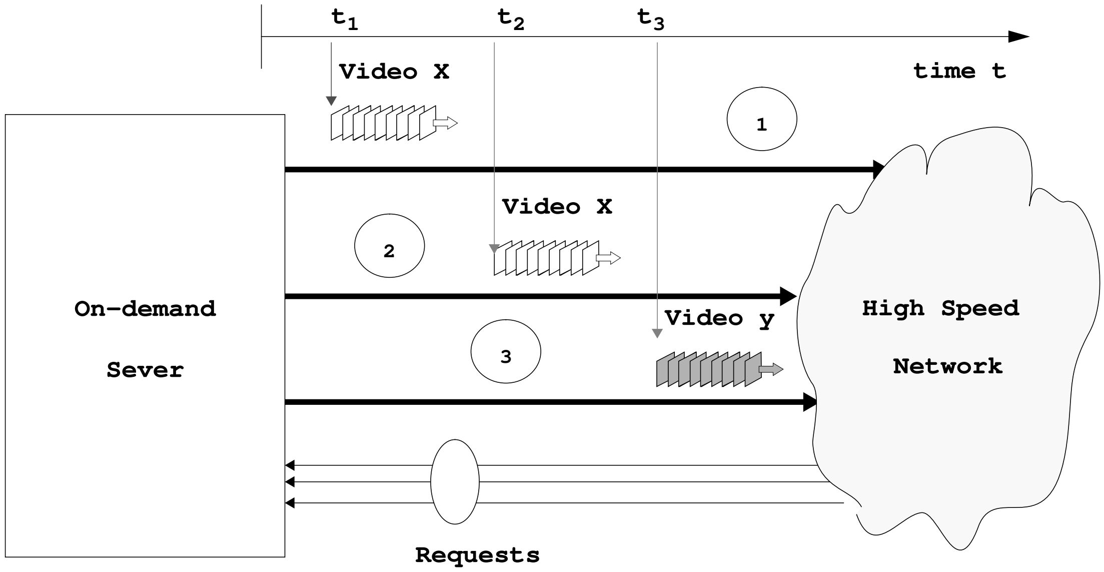

Figure 3: MOD server for a Dedicated Viewing service (DV)

separate retrieval and delivery. Thus, in an on-demand movie server supporting maximum 1000 users with DV service, all 1000 users, in 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 presence of other clients that are active concurrently. The primary advantage of this service is that it is a natural paradigm for personalized, interactive multimedia delivery.

We believe that in future, aforementioned three services will co-exist and cater to different classes of applications. The DV services are not possible at present, but will be important for a large class of exciting new applications that are inherently interactive. However, designing a server which offers DV services and meets all the requirements mentioned earlier is a challenging task. Our (MARS) project uses DV service as an underlying model and attempts to meet this challenge. The basic MARS architecture has been detailed in [2] and here, we will briefly present a prototype implementation that is relevant for the rest of the paper.

3. A Prototype Architecture

Figure 4 shows a prototype architecture of a MARS server. 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) currently being developed as a part of an ARPA sponsored gigabit local ATM testbed. The APIC is designed to support a data rate of 1.2 Gbps in each direction[9].

The central manager 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

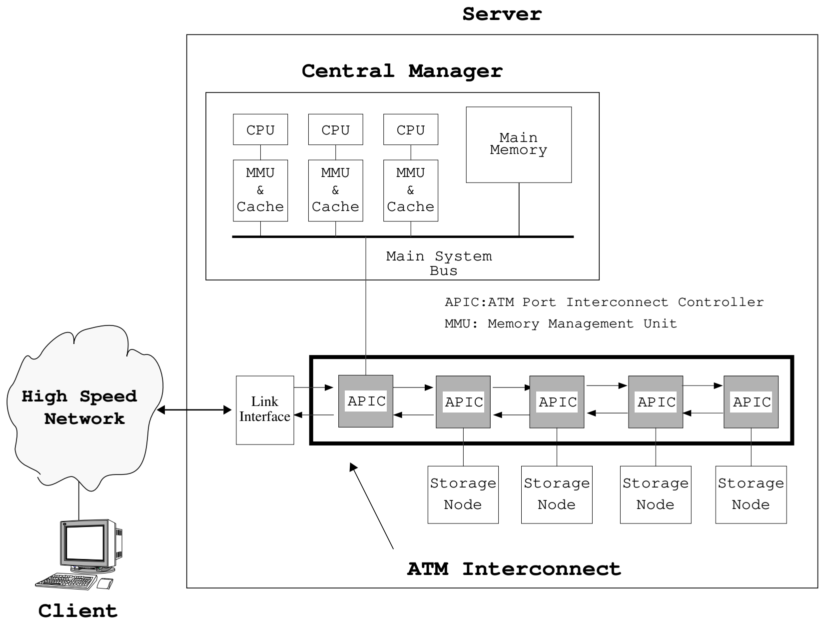

Figure 4: A prototype architecture

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.

The architecture of the storage node described here assumes the storage at the node to be a RAID, however it can be easily extended to accommodate other forms of storage such as a set of independent high capacity optical disks or tertiary storage in the form of optical or magnetic tapes. A RAID storage node is illustrated in Figure 5. The disk array at the node is constructed out of a set of *Small Computer System Interconnect* (SCSI) strings, with a fixed number of disks per string. The multiple SCSI strings are controlled by an array controller, which interfaces to the APIC through a dual ported memory, such as a video RAM (VRAM).

In a read-only environment, the array controller asynchronously writes the data in the VRAM and the APIC consumes it to transfer it to the network. The VRAM is shared by a set of buffers such as: circular buffers for periodic streams, buffers for non-real time tasks (for streams such as still images, text, and progressive image transmissions), request buffers used by the central manager to write control commands for read and write schedule management, and positional meta-data buffers.

The array controller is responsible for managing the local storage and providing one or more of the resource management functions, such as file system operations, scheduling, and compute support. The control block, the heart of the array controller, can be a finite state machine or an embedded processor that runs a small real-time executive (or a real-time OS). Additional details on the storage node and a scalable extension of the prototype architecture can be found in [2].

4. Hierarchical Data Layout

The motivation for our hierarchical data layout schemes results from following two observations:

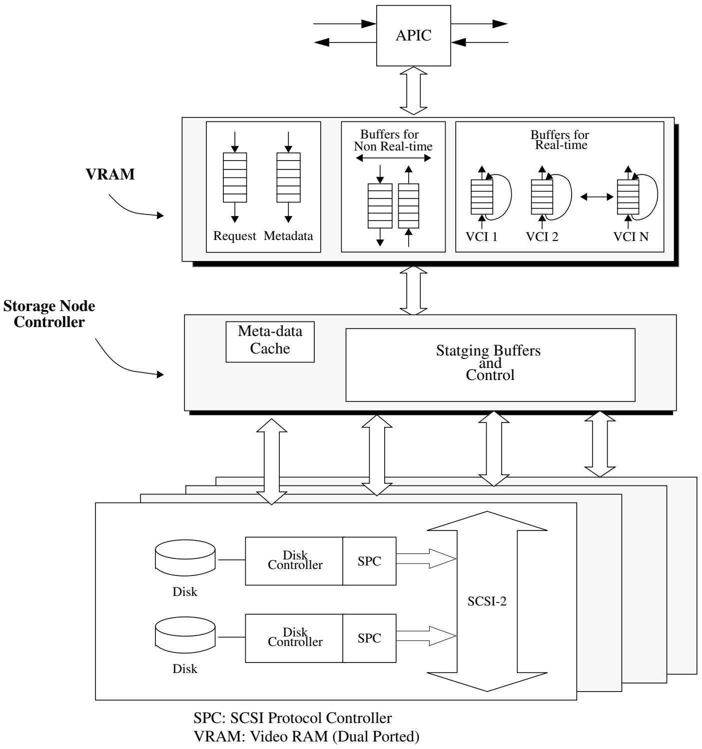

Figure 5: Storage node architecture

- **Disk arrays have limited scalability:** The disk arrays and the associated data striping schemes introduce parallelism to overcome the storage throughput bottleneck. However, though these ideas have been demonstrated to be useful, disk arrays will not be able to support a large number of clients with aggregate throughput of the order of few Gbps [15]. Also, though very high throughputs have been demonstrated in disk arrays developed for supercomputers and mainframes [3, 19], they are expensive custom solutions designed for applications very different from multimedia applications.

- **Multimedia data is amenable to striping:** The periodic nature of multimedia data is well suited to spatial distribution or striping. 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 the parts of it can be physically distributed on different storage devices and accessed in parallel.

Thus, key to getting a large throughput when accessing multimedia documents, is to distribute the contents to a greater degree on autonomous, co-operating storage nodes, such as the one's in our architecture discussed earlier. With this prelude, we will now present our basic distributed layout scheme.

4.1. Distributed Chunked Layout

Figure 6 illustrates an example layout scheme, called as *Distributed Cyclic Chunked Layout* (DCCL), that uses a logical unit consisting of k successive frames as a basic layout unit. We call such

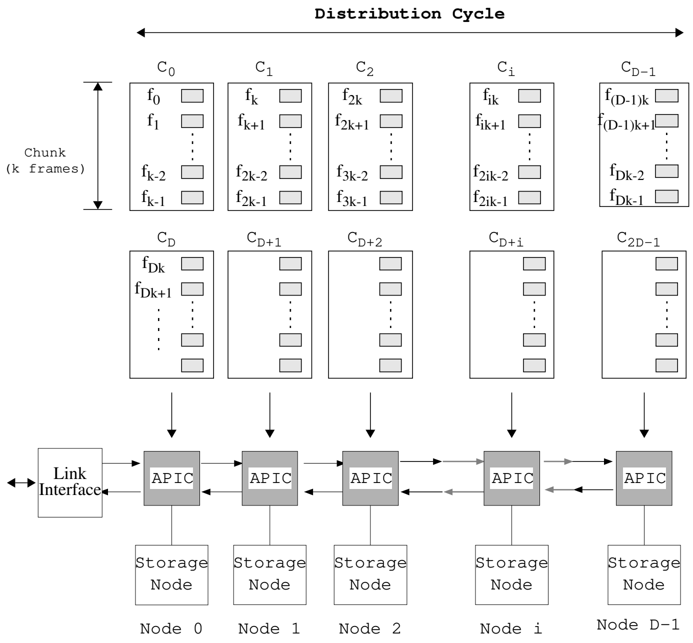

Figure 6: A chunked layout scheme

a unit, a chunk. The frames in a single chunk are confined to one storage node. The chunk size can be any number of frames ranging from, $k = 1$ – one frame per chunk, to $k = F_{max} =$ Maximum number of frames in the stream, which is the entire duration of the media stream. Also, the chunk size may be different for different multimedia documents. The successive chunks of a stream are distributed on the storage nodes as per a certain logical layout topology. For example, as shown in Figure 6, the chunks are laid out following a ring topology. Note that in this topology, the data layout consists of a repetition of a basic pattern, called *Distribution Cycle*, consisting of D consecutive chunks. The first chunk in such a cycle is called the **anchor chunk** and the node to which it is assigned is called **anchor node**. In the case when the chunk size is one, the **anchor chunk** is termed as an **anchor frame** and the resulting layout is called *Distributed Cyclic Layout* (DCL). Also, unlike some of the other layouts presented in subsequent sections, in case of DCCL (as well DCL), the location of the anchor node is always fixed and is the same for all distribution cycles.

In this simple scheme, the time separation between consecutive chunks of the same stream assigned to a storage node is $D \times T_f \times k$. This amount, in turn, represents time between two consecutive pre-fetches, if the entire chunk is fetched as a single unit. Similarly, in case of DCL, two consecutive frames of a stream at a storage node are separated by D frames, thus, increasing the effective period of the stream from T_f to $T_f \times D$. In other words, the stream is slower by a factor of D from the perspective of each node.

Each node stores the chunks assigned to it on the local storage devices as per a node-specific storage policy. For example, in case of storage in the form of a disk array, various possible options are: 1) Store the chunk contiguously on the same disk, 2) Store every frame in the chunk on a separate disk, and 3) Stripe each frame in the chunk on the disk array. The detailed discussion of several tradeoffs associated with each of these options is beyond the scope of this paper.

A storage node may retrieve the chunks assigned to it from its local storage as a single unit or in parts (frames). The tradeoffs involved in the choice of the option stem from following factors: 1) size of buffer available at the client, 2) amount of per-connection buffer available at the server, and

3) type of network service used for data transport between the client and the server.

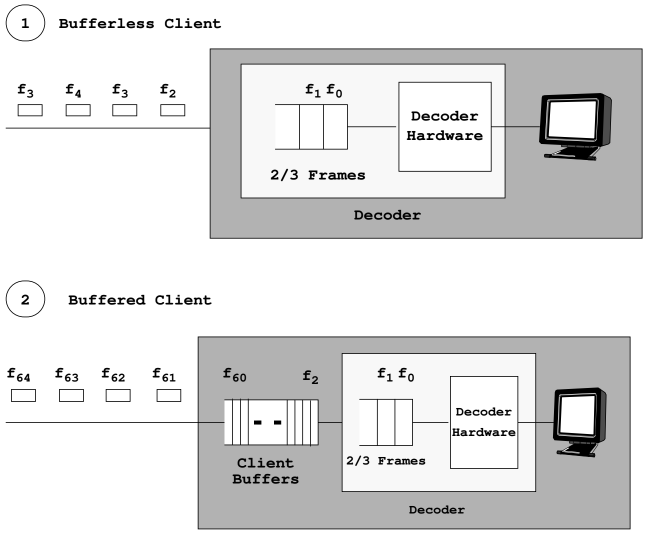

Figure 7: Buffer scenarios at the client

Figure 7 illustrates the two possible buffer scenarios at the client. Typically, a **Bufferless Client** has a few frames worth of buffers required by the decompression hardware. For example, a typical MPEG decoder may have buffer for at least three frames - for I and P anchor frames required to decode B frames and the frame buffer. On the other hand, a **Buffered Client**, in addition to the decoder buffer, has a buffer large enough to store several (100 to 200 - approx. few seconds worth) frames. Availability of such buffer makes network delay and jitter a non-issue, but requires buffer management on the part of the client.

Next, assuming a DCL layout, we will present our hierarchical scheduling schemes.

5. Hierarchical Scheduling

Given a chunked layout, a storage node in our architecture has following three options as to how it schedules prefetch and transmission of chunks:

1. *Fetch and transmit the chunk frame-by-frame*
2. *Fetch the chunk as a single unit and transmit it as a single burst*
3. *Fetch chunk as a unit and transmit it frame-by-frame*

In each of these options, in order to support guaranteed periodic retrieval of data for all active connections, we use a 3-level hierarchical scheduling scheme. The highest level of this scheme consists of a scheduling policy that schedules data retrievals from the unsynchronized storage nodes. The second level consists of a scheme for scheduling the storage devices at a node (e.g.: scheduling reads from the disks in the disk array, scheduling disk head movement etc.). The last level guarantees periodic pacing of data by each APIC associated with a storage node, on to the APIC interconnect and subsequently to the network, as per a rate specification. The complexity of scheduling algorithms used at all levels depends on various factors such as, *size of the buffer available at the client side, type of data transport service provided by the network, size of per connection buffer at the server, design of the data layout, and the extent of support for sophisticated interactive playout control.* In following subsections, we will discuss the first option in detail and briefly describe the rest of the options.

5.1. Scheme 1: Fetch and transmit the chunk frame-by-frame

As shown in this subsection and subsequent sections, using simple scheduling schemes, this approach can provide strict deterministic QOS guarantees to all active clients during normal playout as well as stream control operations. It minimizes buffer required at the client and the server, offers the best operation latency, and allows efficient implementation of playout control operations. At first thought, it may seem that fetching a chunk frame-by-frame increases seek overhead at the disks. However, we believe that under near-full load (i.e when number of active clients and hence, request per disk are large in number), an appropriate disk head scheduling policy that minimizes rotational and seek latency by ordering disk requests, in conjunction with smallest chunk size, will provide good disk utilization and allow the disk arrays at the storage nodes to be used in a load-balanced fashion.

Due to space constraints, here we will present only one scheme for scheduling data reads over multiple, unsynchronized storage nodes. First, we will describe a simple scheduling scheme that efficiently supports independent normal playout of same or different document for all active connections. In the next section, we will illustrate the modifications to this scheme to support playout control operations. The salient feature of these modifications is that they that require no extra BW reservation and offer tolerable operation latency at the cost of doubling per connection buffer requirement.

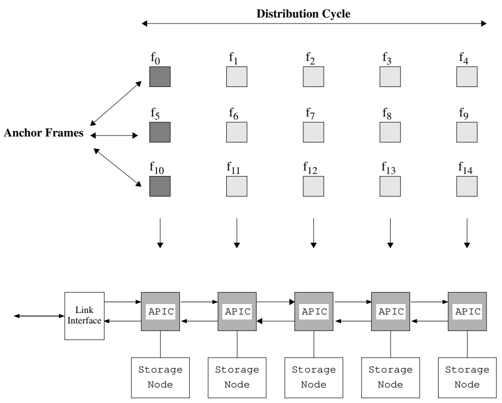

Figure 8: A simple layout example

Figure 8 and Figure 9 together illustrate a simple data layout and a scheme for scheduling data retrieval from storage nodes when clients are bufferless. In this scheme, each storage node maintains C_a buffers, one for each active connection. In a retrieval environment, 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. Each cycle consists of three phases: *data transmit*, *handover* and *data pre-fetch*. During the *data transmit phase* (T_{Tx}), the

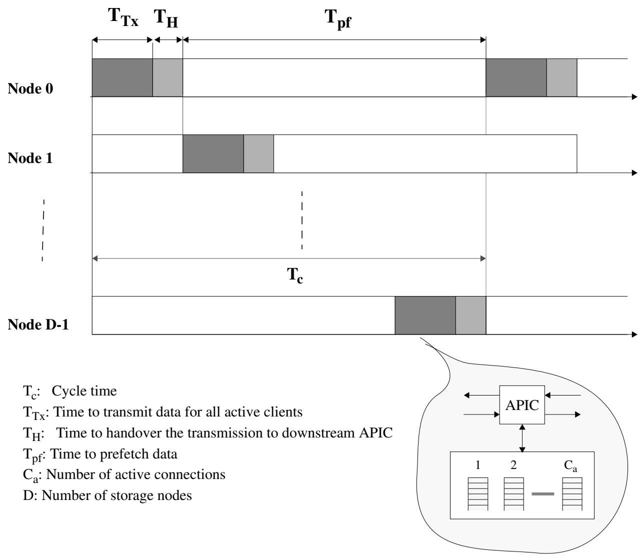

Figure 9: A simple scheduling example

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. The last phase in the cycle, namely the *data pre-fetch phase* (T_{pf}) is used by the storage node to pre-fetch data for each connection that will be consumed in the next cycle.

As an example, consider a prototype with 15 storage nodes ($D = 15$) servicing a fixed number of video connections which have frame period of $T_f = 33$ msecs (ms). If the APIC interconnect bandwidth is 622 Mbps, the effective data bandwidth available is 563 Mbps, excluding the bandwidth lost due to the ATM header overhead in each cell. Using these values, if the cycle time is set to $T_c = 33 \times 15 \approx 495$ ms, the length of the transmit phase can be at most $T_{Tx} = 33$ ms. Thus, each storage node has to pre-fetch a frame worth of data for each connection every $((495 - 33) - T_H) = 462 - T_H$ ms. Assuming that the T_H is of the order of 1 ms, the pre-fetch time is 461 ms. A simple RAID constructed using disks with maximum rotational and seek latencies of 10 ms can deliver 5 MBps. Thus, a storage node with such a RAID can pre-fetch 2.3 megabytes of data in 461 ms. A buffer of this size can store approximately 27 frames, each of 84 KB - the average size of a MPEG encoded 20 Mbps HDTV video stream. Since each frame belongs to an independent connection, ≈ 27 independent HDTV connections are possible. The total bandwidth requirement of these connections is 27×20 Mbps $= 540$ Mbps, which is less than the effective interconnect bandwidth of 563 Mbps. Thus, ≈ 27 compressed HDTV clients can be supported simultaneously. Similar calculations show that approximately 110 standard NTSC quality MPEG clients can be supported in this setup.

5.2. Scheme 2: Fetch the chunk as a single unit and transmit it as a single burst

Fetching the entire chunk as a single unit has the advantage of amortizing the seek overhead at the disks over large transfers. This may improve the disk utilization, but will increase the per-connection buffer requirements at the server.

In the case of a buffered client, the server can transmit the prefetched chunk as a single burst. Figure 10 illustrates the basic idea behind scheduling transmissions in this fashion. In this example,

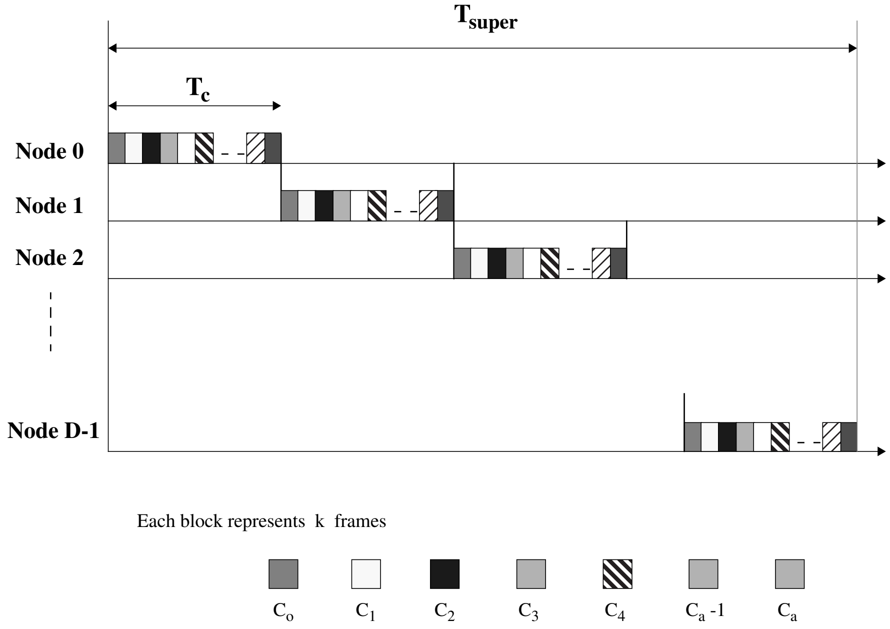

Figure 10: A simple scheduling that transmits chunks as single bursts

the global schedule is a recurring super-cycle that consists of D cycles. One such cycle, each of time length $T_c = T_f \times k$, is assigned to every storage node. In each cycle, the APIC associated with the node transmits the prefetched chunk of every active connection as a single burst. At the end of its assigned cycle, the node hands over transmission to the successor node in the linear (modulo D) order. Thus, the server assigns the entire interconnect and the network link BW to a single connection and sends long burst of data into the network. However, this requires that the network support a transport service that allows reliable transmission of such large bursts of data at link rates. Also, such burst transmission has to occur periodically (every $D \times k \times T_f$) for each connection, with minimum burst loss and/or blocking probability. Supporting a large number of such active connections is a non-trivial task for a network designer.

Another interesting observation is that to ensure QOS to a large number of concurrent clients, between any two consecutive chunk transmissions for a stream, the server has to successfully transmit the chunks from the rest of the clients. The time between consecutive chunk transmissions for a stream depends on the amount of buffer at the client side. For example, a smaller client buffer needs to be filled more often by frequent chunk transmissions. This implies that a client with the smallest buffer dictates the periodicity of chunk transmissions for all connections. Thus, even if some of the admitted clients have large buffers, server has to treat all clients to have buffer of only one size - the one which is smallest among all. In other words, all clients must have the same buffer size equal to the chunk size.

Consider, a scenario in which all clients are identical (have same amount of buffer) and the network provides necessary burst transmission service. Let the number of storage nodes be $D = 15$ and the chunk size $k = 30$ frames. This means that a node must prefetch and transmit a chunk every $33 \times 30 \times 15 = 14.85$ secs for each active connection. Given that the average frame size of an MPEG compressed NTSC quality video is approximately 20 KB, the chunk size is approximately 600 KBs. In order to serve 100 independent, identical clients, 100 such chunks must be retrieved in a cycle time of 14.85 secs. Therefore, the average throughput of the storage device, such as a disk array, at the storage node must be approximately 4.1 MBps. If the chunk size is reduced to 2 frames ($k = 2$), the disk array throughput requirement is unchanged, but the cycle time is now reduced to mere 990 msecs. This cycle time affects the latency for playout control operations. Also, unrestricted playout control is problematic with large chunk sizes (large cycle lengths) and requires over-engineering the architecture by reserving extra BW at each node for certain playout control operations.

5.3. Scheme 3: Fetch chunk as a unit and transmit it frame-by-frame

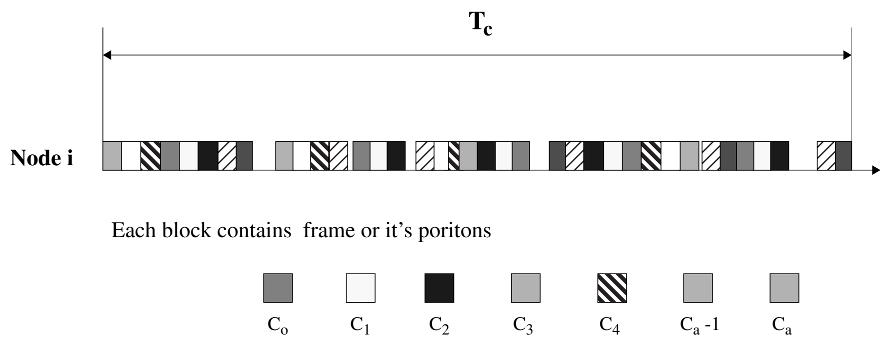

Figure 11: Interleaving transmissions of all connections in frame-by-frame transmission

At the client, this approach minimizes the buffer requirement, but at the server, the advantages as well as the disadvantages of fetching the chunk as a single unit are the still the same as in Scheme 2.

The frame-by-frame transmission in this scheme allows the server to use the efficient VBR transport services that will be provided by future ATM networks. Thus, in this case, the APIC associated with each node would interleave transmission of data for all connections over the interconnect and subsequently to the network. Figure 11 illustrates such interleaving in a single cycle assigned to a typical Node i.

Note that in presence of delay jitter introduced by network congestion and instantaneous sever overloads, frame-by-frame transmission used in Scheme 1 and 3 may cause data to be delivered to the client beyond required deadlines. However, standard techniques, such as a small playout buffer at the client, can easily mask this effect.

To summarize, Scheme 1 and 3 are two promising approaches available to a server to fetch and transmit the data. In the rest of the paper, we will concern ourselves with special case of Scheme 1 that uses chunk size of one frame.

6. Scheduling Playout Control Operations

As discussed earlier, future on-demand multimedia applications will require interactivity in the form of stream playout control that allows 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*

Table 1: Classification of playout control operations

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

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 same group.

In order to understand implications of these operations on data layout and scheduling, it is necessary to first understand various ways of implementing the playout control operations.

6.1. Semantics of Playout Control Operations

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 analyzed in [8].
- **Sequence variation (SVS) scheme:** 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 side is unaltered but the retrieval rate and the transmission rate 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 **SVS**, irrespective of whether a video stream is in normal play mode or *ff* mode, 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, every d^{th} frame (d = fast forward distance).

However, the **RVS** implementation of *ff* and *rw* has following significant drawbacks:

1. **Increased network and storage BW requirement: RVS** approach increases the resource requirements at the server in the form of increased buffer and storage and network BW. Since the interactive behavior is typically unpredictable, any deterministic guarantees to interactive operations would require the server to be highly over-engineered. Also, no matter how the server makes n (fast forward factor) times the normal playout BW available for fast forward, the network may not be able support such increases in BW requirement without high cost and/or significant blocking.
2. **Inappropriate for real-time decoders:** Most of the decompression engines at the client handle real-time decoding of incoming data at a rate smaller than or equal to a maximum frame rate. Thus, a MPEG decoder hardware can decode at the most 30 fps. Therefore, any attempt to increase the decoding/display rate by increasing the data rate will not work. In other words, it does not help to use complex (presumably intelligent) algorithms at the server to send data at higher rate, if the client cannot handle it.

3. **Increased buffer requirement at the client:** Given the cost factor, a typical real-time MPEG decoder has minimal (3 frames worth) buffers. This implies that data coming in at n times the normal playout rate cannot be buffered at the client and will be dropped, wasting all the "good" work server and network did to transport it to the client.

Due to these compelling reasons, we choose to implement *ff* and *rw* using **SVS** approach. We distinguish *fast-play* (*fp*) and *ff* as two different operations and implement *fp* using **RVS** approach. 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 in **RVS**. 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.

6.2. Scheduling for FF and RW is problematic

The simple data layout and scheduling scheme described in Section 5 works for normal stream playout but not for *ff* and *rw*. Consider a connection in an example system with number of storage nodes $D = 6$ and a fast forward implementation by skipping alternate frames. The frame sequence for normal playout is $\{0, 1, 2, 3, 4, 5, \ldots\}$, whereas for the fast forward the same sequence is altered to $\{0, 2, 4, 6, 8, 10, \ldots\}$. The set of nodes from which the frames are retrieved in the normal playout is $\{0, 1, 2, 3, 4, 5, \ldots\}$. Upon *ff* this node set is altered to $\{0, 2, 4, 0, 2, 4, \ldots\}$. Clearly, in this example, during *ff* the odd-numbered nodes are never visited for frame retrieval. This means that for every active connection performing fast forward or rewind, even numbered nodes have to fetch frames twice as often. Another serious implication of this is that as the display rate is constant, node 2 for example, must retrieve and transmit data in a time position which is otherwise allocated to node 1 in normal playout.

Thus, there are two main problems: first, the stream control alters the sequence of node-visits from the normal linear (modulo D) sequence. In other words, 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. Secondly, it forces some nodes to retrieve and transmit more often, creating "hot-spots" and, in turn, requiring bandwidth to be reserved at each node to deal with the overloads. Such additional bandwidth reservation will lead to conservative admission control and poor utilization. If no such bandwidth reservation is made, QOS may be violated during overloads.

The first step in fixing these problems is to decide the order of transmissions for different nodes on a per-connection basis. Figure 12 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. Note that the during the time duration assigned to each node, transmissions for connections C_0, C_1, C_2 and C_3 need not be in strict sequential order and can be interleaved at the cell level.

When the *ff* request for connection C_0 received in 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 13 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 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

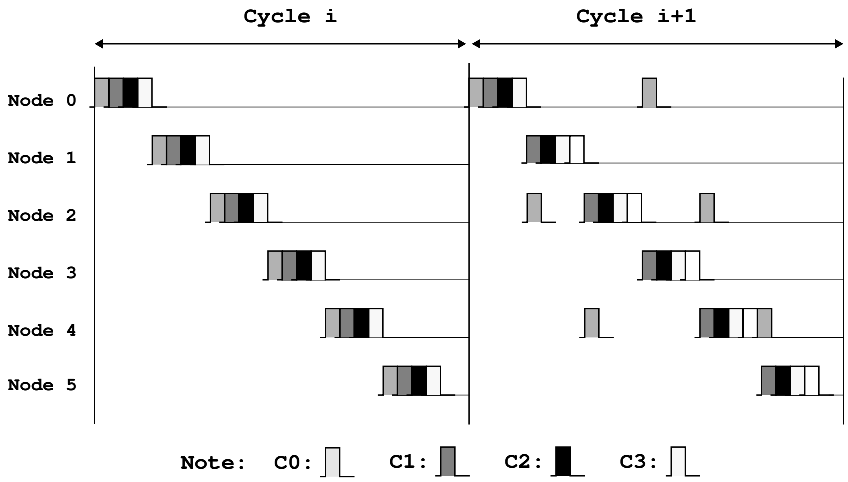

Figure 12: Revised schedule when C_0 performs fast forward

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 now 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 to guarantee that data to be transmitted has been prefetched into the buffers. Hence, to achieve a smooth transition from normal playout to fast forward without violating QOS guarantees, we provide two buffers per connection: one buffer used to store the data being prefetched and the other used to transmit the previously prefetched data. This effectively decouples transmission and prefetching and allows transmission order to be modified on a per cycle basis.

When a *ff* request is received, the next D frames in fast forward frame sequence and sequence of nodes to which they correspond are computed. These frames are retrieved and buffered in the existing prefetch cycle. The ordered set of nodes from which the frames are read represents the transmission order for the next transmission cycle. Note that the central manager in our architecture receives the *ff* (or *rw*) requests and computes the new transmission order for each connection doing fast forward or rewind at the start of each prefetch cycle. Since cycle length is typically a few hundread msecs, and computation of transmission (prefetch) order involves simple modulo arithmetic, the overhead incurred is minimal. Note that in this scheme, the operation latency for the *ff* or *rw* is a maximum of one cycle. Consider the example, of $D = 15$ and frame period is $T_f = 33msec$ in the Section 5. In this example, the *ff* request from any (potentially all) of the 110 clients can be realized in 1 cycle time of 495 msecs. The *ff* and other playout control operations increase the scheduling overhead, but we believe that it is insignificant.

As can be seen in the above example, the load distribution is still unbalanced. Load balance is ensured, if we can guarantee that each storage node fetches and transmits a fixed number of frames per connection in each prefetch and transmission cycle irrespective of whether a connection is performing normal playout or other playout control operations. This is achieved by either constraining the data layout or modifying the data layout. We will revisit this topic again in Section 7.

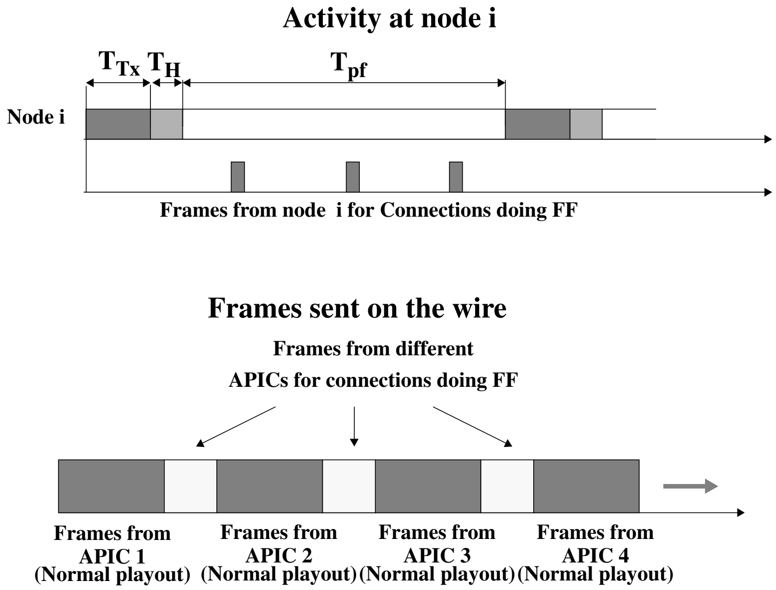

Figure 13: General case of M out of C_a connections doing fast forward

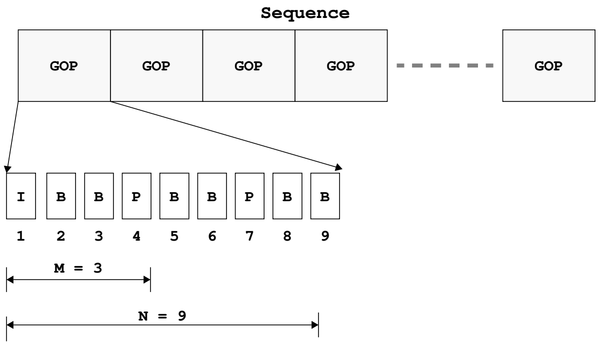

Figure 14: Structure of MPEG stream

6.3. Implications of MPEG

The structure of MPEG compressed streams has implications for the data layout, scheduling and playout control. In its simplest form an MPEG stream, illustrated in Figure 14 consists of a succession of group-of-pictures (GOP), each of which for example, has a structure [IBBPBBPBB]. The I frames are coded using a Discrete Cosine Transform(DCT) without reference to any other pictures, and hence, can be decoded independently. Thus, they can be treated as anchor frames in the stream from which decoding can begin. However, intra-coding achieves moderate compression and hence, I frames are very content intensive. The P frames are coded using motion compensated prediction from a past I or a P frame and are normally used for further prediction. Clearly, P frames cannot be transmitted independently. The B frames achieve the highest compression and hence, have the smallest content. They are encoded using non-causal prediction from both past and future reference frames (i.e I and/or P frames). Clearly, B frames cannot be transmitted independently and cannot be used as a reference frame. Empirical evidence shows that depending upon the scene content, I to

P frame variability is about 3:1, whereas P to B frame variability is about 2:1 [23].

These properties have serious implications for *ff* [1]. First, a *ff* sequence obtained by skipping 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 [IPIPIP...] or [IIII...]. This implies that the set of fast forward distances (d_f) for a **SVS** implementation of *ff* on a MPEG stream will be a function of M and N parameters in Figure 14.

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 presence of many such connections, network will not be able easily meet such dramatic increases in BW demand Clearly, reserving such BW at the connection setup will be immensely wasteful. Second 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. There are three possible solutions to these problems:

1. **Reduce frame display rate:** One way to reduce the transmission rate at the server and the network connection BW is to do "temporal down-sampling", which reduces the display rate. For example, one may expect that in a fast forward implementation by skipping $(N-1) = 8$ frames, if the display rate is reduced from 30 fps to 15 fps, the perceived *ff* rate will be 4 times normal playout rate. However, reducing display rate may cause perception of jerkiness.

2. **Reduce the content of the I and P frames:** Clearly, reducing the number of bits in each I and P frame will reduce the bandwidth requirement. In case of live sources, MPEG encoder can accomplish this by reducing the quantization factors used for coding the I and P frames. However, in case of stored compressed streams ("canned") streams, as in case of a multimedia server, such reduction will be possible only if the versions of these frames corresponding to reduced quantization factors are also stored. However, reducing quantization factor unduly can deteriorate detail and result in poor subjective quality.

3. **Reduce the frame resolution:** Another way of reducing the content of the frame is to reduce the resolution of the frames i.e perform spatial domain down-sampling. Thus, when the stream is stored on the server, the I and P frames are stored in multiple resolution. On a *ff*, the server would retrieve and transmit I and P frames with reduced resolution. However, to keep the resolution unchanged at the client, up-sampling hardware (such as [1]) needs to be used at the client side.

We believe that in order to keep the network BW requirements during *ff* under control, a combination of these techniques will have to be employed.

MPEG does have implications on the data layout schemes used at the server. Specifically, the data layout must ensure that the I frames in all GOPS are not assigned to the same node. This is important, as the nodes handling (retrieving and transmitting) I frames will be loaded more than those responsible for P and I frames. It is desirable that the server be able to exploit this VBR characteristics to avoid always reserving peak storage BW.

We claim that our data layout schemes compensate for the load-imbalance inherent in distributing frames for an MPEG stream. Also, our dynamic cycle-by-cycle scheduling allows the VBR characteristics of such streams to be exploited.

[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.

7. Load Balancing

In the previous section, we discussed the two main problems that playout control operations create for the distributed scheduling of data reads over multiple storage nodes. The reader may recall that fast forwarding (or rewinding) by an arbitrary distance modifies the normal linear (modulo D) sequence of node visits to a new sequence in which some of the nodes might be visited more often and thus, create "hot-spots". This requires extra BW to be reserved at each node to meet overloads and hence, will lead to conservative admission control and poor utilization. The key to avoid such conditions is to ensure that irrespective of the state of the connections (whether they are doing normal playout or other playout control operations), each node handles the same number of frames in each pre-fetch/ transmission cycle. This condition can be achieved in two ways: either by constraining the DCL data layout and or by designing a new layout scheme. In the rest of this section, we will present some interesting results on both these approaches.

7.1. Load Balance with a Distributed Layout

In the discussion here on, it is important to understand the definition of load-balance. 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 implication of distributed data layout in conjunction with periodic scheduling on load-balance is that *ff* and *rw* implementation by skipping frames can not realize arbitrary fast forward/rewind rates without violating the load-balance condition. This can be stated formally as follows.

THEOREM 1. *Given that the number of storage nodes is finite, no distributed (multimedia) data layout scheme will support fast forward (rewind) of arbitrary skipping distance without violating the load balance condition.*

Proof: Let the number of storage nodes be D and the set of frames in the stream be $\{f_0, f_1, f_2, f_3, \ldots, f_{F_{max,d}}\}$. Since $D \ll F_{max,d} \ll \infty$, if a distributed data layout is followed, any given storage node will have multiple frames of the stream. In particular, let frames at storage node p be $\{f_{k_1}, f_{k_2}, f_{k_3}, \ldots, f_{k_{max}}\}$ (where $0 \leq k_i \leq F_{max,d}$). Clearly, if the fast forward starts from frame number f_{k_1} with a distance of $k_2 - k_1$, node p will be visited consecutively, before any other node is visited. This violates the load balance condition. Hence, given a distributed layout scheme, fast forward (rewind) of arbitrary distances is not possible. ∎

7.2. Load Balance in a Distributed Cyclic Layout

In order to make the VOD service attractive it is desirable that the on-demand multimedia server offer a rich choice of *ff* (*rw*) speeds to the user of on-demand multimedia. In light of this requirement and the above theorem, a distributed data layout scheme should allow as many fast forward distances as possible without violating the load balance. One of the important factors that will decide extent of such choice is the number of storage nodes D on which the data is distributed. We conjecture that there will be a relationship between the fast forward (d_f) [2] distance and D[3].

To this end, let us take a closer look at the example DCL shown in Figure 8. Assume that the fast forward starts from frame f_0 with a fast forward distance d_f of 2. First $D = 5$ frames

[2] Since fast forward and rewind are complementary operations, here on we will only talk of the former.

[3] This result was first suggested by Dr. Arif Merchant in a different form during the first author's summer internship at NEC C&C Research Labs, in Princeton, New Jersey.

in the fast forward sequence are $\{f_0, f_2, f_4, f_6, f_8\}$, which are retrieved from a balanced node set $\{D_0, D_2, D_4, D_1, D_3\}$. If the fast forward distance is 3, the node set if altered to ordered set $\{D_0, D_3, D_1, D_4, D_2\}$, which is still balanced. It can be easily verified that the node set is balanced when $d_f = 4$, but is unbalanced when $d_f = D = 5$ or $d_f =$ integral multiple of D. With this background, we present the following theorem that relates D and d_f explicitly for a DCL layout.

THEOREM 2. *Given a* DCL *distributed layout over D storage nodes, the following holds true:*

- *If the fast forward (rewind) distance d_f is relatively prime to D, then*
  1. *The set of nodes S_n from which consecutive D frames in fast forward frame set S_f are retrieved is load-balanced.*
  2. *The fast forward can start from any arbitrary frame (node) number.*

Proof: We give a proof by **contradiction**. Let f be number of the arbitrary frame from which the fast forward is started. The D frames in the transmission cycle are then given as:

$$\{f, f + d_f, f + 2d_f, f + 3d_f, \ldots, f + id_f, \ldots f + jd_f + \ldots f + (D-1)d_f\}$$

If the frame f is mapped to node n_f, the set of nodes from which these D frames are retrieved is as follows:

$$\begin{aligned} f &\mapsto n_f, \\ f + d_f &\mapsto (n_f + d_f) \bmod D, \\ &\vdots \\ f + id_f &\mapsto (n_f + id_f) \bmod D, \\ &\vdots \\ f + (D-1)d_f &\mapsto (n_f + (D-1)d_f) \bmod D, \end{aligned}$$

Without any loss of generality, assume n_p to be one of the D storage nodes that appears at least more than once in this node set. This means two frames, say $f + id_f$ and $f + jd_f$ are mapped to the same node n_p. If we carefully study the DCL we can see that the set of frames assigned to the node n_p can be defined as follows:

$$F_p = \left\{ \begin{array}{lllll} p, & p+D, & p+2D, & P+3D, & P+4D \\ \cdots, & \cdots, & p+iD, & p+(i+1)D, & p+(i+2)D \end{array} \right\}$$

that is

$$F_p = \{\forall \ell \in \mathcal{N} : (p + \ell \times D) \mapsto n_p\} \tag{1}$$

Clearly, any two frames mapped to the same node differ by an integral multiple of D. Hence,

$$\begin{aligned} (f + jd_f) - (f + id_f) &= kD \\ (j - i) &= k \times \frac{D}{d_f} \end{aligned} \tag{2}$$

Two cases that arise are as follows:

- **Case 1: k is not a multiple of d_f:** If D and d_f are relatively prime [4], then, $\frac{D}{d_f}$ can not be a integer. The L.H.S, being a difference of two integers, is an integer. Hence L.H.S $\neq$ R.H.S. A contradiction to our assumption.
- **Case 2: k is a multiple of d_f:** Then $(j - i) = k_1 \times D$, where $k_1 = \frac{k}{d_f}$. But this contradicts the assumption that the two selected frames are in the set which has only D frames and hence can differ at the most $D - 1$ in their ordinality.

Since, the frame f from which fast-forward begins is selected arbitrarily, the last Claim 2 in the Theorem statement is also justified. The proof in case of rw is similar to above proof and is not presented here. ∎

We state the following corollary to this theorem.

COROLLARY 1. *Given a* DCL *distributed layout with cycle length D, such that D is prime, following holds true:*

- *If the fast forward (rewind) distance d_f is not a multiple of D, then*
  1. *The set of nodes S_n from which D frames in a transmission cycle are retrieved is load-balanced*
  2. *The fast forward can start from any arbitrary frame (node) number.*

This follows immediately from Theorem 2.

7.3. Staggered Distributed Cyclic Layout (SDCL)

As explained in Theorem 2, the DCL works only when d_f is relatively prime to the number of storage nodes D. Also, the proof of this theorem illustrated that if fast forwarding distance d_f is a multiple of D or has a common factors with D, the load is not balanced across the nodes. In order to fix this problem, we have designed a new data layout scheme called as the *Staggered Distributed Cyclic Layout* (SDCL).

The key idea in this scheme is that the anchor frame in a frame distribution cycle is not always assigned to the same node. Instead, the anchor node of successive distribution cycles is staggered by a distance of k_s and hence the term *Staggered Distributed Cyclic Layout*. Figure 15 illustrates the layout for $D = 8$ and $k_s = 1$. As seen in this figure, the layout consists of a pattern called *"stagger cycle"* composed of D (frame) distribution cycles and D^2 frames. The anchor frame of the $i^{th}(0 \leq i \leq D-1)$ distribution cycle in the stagger cycle is assigned to the i^{th} node. Thus, the location of anchor frames in two consecutive distribution cycles is staggered by 1 node. Like the DCL layout, the frames within the distribution cycle are distributed following a rotated (modulo D) sequence.

We will illustrate some of the special properties of this layout with an example. Let us consider *ff* implementation by skipping alternate frames (that is $d_f = 2$) starting from frame 0. The original frame sequence $\langle f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7\rangle$ is then altered to $\langle f_0, f_2, f_4, f_6, f_8, f_{10}, f_{12}, f_{14}\rangle$. The node set for this new sequence is then altered from the balanced set $\langle D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7\rangle$ to $\langle D_0, D_2, D_4, D_6, D_1, D_3, D_5, D_7\rangle$. Clearly, this new node set is re-ordered but still is balanced. However, if $d_f = 3$, the similar node set is given as $\langle D_0, D_3, D_6, D_2, D_5, D_0, D_4, D_7\rangle$, which contains D_0 twice and hence is unbalanced. It can be verified that cases $d_f = 4$, $d_f = 8$ and $d_f = m \times D$,

[4] If two numbers p and q are relatively prime, then their greatest common divisor is 1.

where m is relatively prime to D produce balanced nodes sets as well. Note here that $2, 4$ are factors of $D = 8$, but 3 is not.

Another interesting property of SDCL is that in order to produce a balanced node set on *ff*, the fast forward must always begin at an anchor frame. For example, if the fast forward begins at f_{30}, a non-anchor frame and $d_f = 4$, then the 7^{th} frame in the fast forward sequence - f_{58} has to be retrieved from the same node (node 1) as frame first frame in the sequence - f_{30}, producing a unbalanced node set.

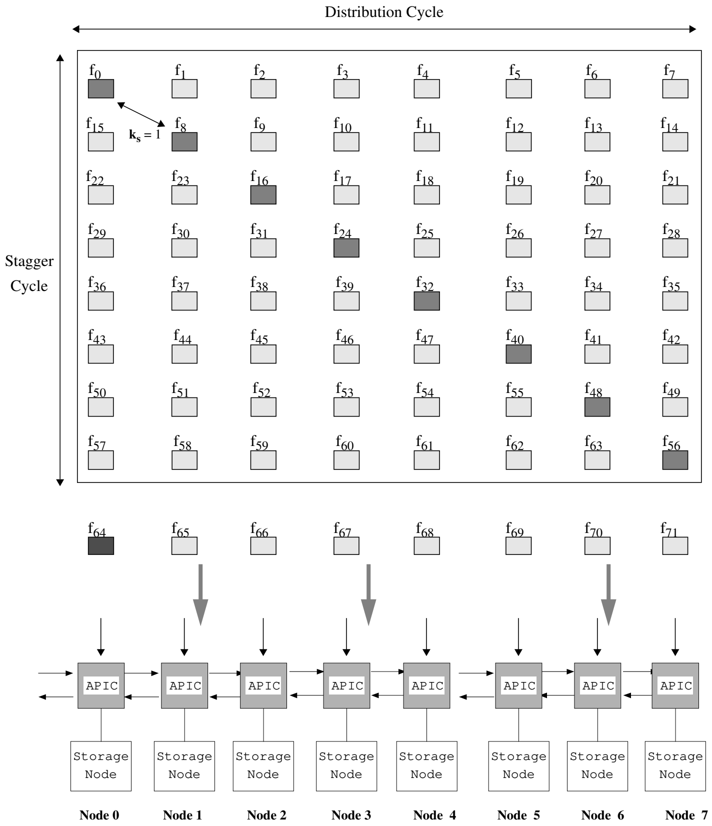

Figure 15: Staggered Distributed Layout Scheme (SDCL)

Thus, we see that a SDCL with $k_s = 1$ guarantees that if the fast forward (rewind) operation begins from a anchor frame and if the fast-forward (rewind) distance d_f (d_r) satisfies certain condition, then the node set from which D frames in a transmission cycle are retrieved is balanced. We will now state the following theorem which formalizes this property.

THEOREM 3. *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* $1 \leq i \leq p$*) 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.*

The proof of this theorem has not been included here due to space constraints. Note that the Theorem 2 and 3 allow the server to know, at the time a multimedia document is laid out, the set of fast forward distances that can be supported safely without requiring any bandwidth reservation. Also, if the server stores a multimedia document using both the DCL and SDCL layout schemes, the choice of fast forward distances can be increased significantly. For example, if $D = 16$, the DCL layout will produce load-balanced node sets for $d_f = 3, 5, 7, 9, 11, 13, 15$ which are relatively prime to D, where as SDCL will produce load-balanced node sets for $d_f = 2, 4, 8, 16$.

8. Related Work

High performance I/O has been 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 will summarize some notable efforts.

Salem et al. [28] 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. [22] 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 [22]. 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 [6]. 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* [16]. 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.

Several recent proposals for filesystems and servers have attempted to address the special requirements of multimedia streams. In particular, [29] represents one of the early qualitative proposals for a on-demand video file system. The work by Rangan et al. [24, 32] 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.

Lougher et al. report 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 [18]. 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. [30] report a small scale PC-AT and RAID based video server. A commercial continuous video server called *Shark* [11] is currently being marketed by IBM. However, it is aimed at the PC based clients and is not very scalable.

Little et al. [17] develop a workstation based non-scalable video server. Their main emphasis has been on metadata database and a video browser. Similarly, Miller et al. [21] at NYNEX are working with Dow Jones to develop a Multimedia News On-Demand Service. Again, the emphasis of their work is on developing news information models, browsing interfaces and authoring systems. A very similar project at the University of California at Berkeley [27] is aimed at developing metadata indices and efficient user interfaces for their large distributed video databases and servers.

Daigle et al. [7], Yu et al. [34], Reddy et al. [25] report work exclusively on disk scheduling aimed at supporting retrieval of multimedia streams from single disks.

To summarize, all the proposals mentioned above address the problems of scheduling, data layout, and metadata 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 A taking an integrated approach that collectively addresses all the issues.

9. Conclusions and On-going Work

Design of high performance large scale multimedia servers will be critical to wide deployment of exciting new multimedia applications. In addition to the requirements of supporting large number of clients with QOS guarantees, and delivering a large network and storage throughput, a multimedia server must be able to support full range of interactive stream control operations. Given the modest rate of improvement in storage technologies, designing large scale servers is a challenging task. Our project *Massively-parallel and Real-time Storage* architecture addresses this challenge.

In this paper, we focussed on some of the distributed data layout, scheduling and playout control techniques that have been developed in conjunction with our architecture. Specifically, we presented a general distributed chunk layout scheme and discussed tradeoffs associated with it. We also illustrated a simple scheduling scheme for supporting large number of clients independently accessing the same or different multimedia documents.

In order to address the requirement of support for interactive operations, we showed that implementing *ff* and *rw* operations creates problems for distributed scheduling and presented a two fold solution: one, modify the scheduling scheme and the second, constrain or modify the data layout scheme. Our modifications to the basic scheduling, such as a providing two buffers per connection and performing transmissions on a per-connection basis, allow smooth transition from normal playout to *ff* and *rw* with minimal overheads and tolerable latency of few hundread msecs.

We argued that unrestricted fast forward is not possible with distributed data layout. Recognizing this fact, we provided analytical results that defined the set of fast forwarding distances for a given number of nodes and required no extra storage and network BW reservation. For the case when number of nodes and fast forward distance are not relatively prime, we designed a new data layout scheme called *Staggered Distributed Data Layout* and presented an analytical result that provides conditions on fast forward distances to guarantee no extra BW is required.

To summarize, we presented data layout, scheduling and playout control techniques that collectively minimize buffer requirements per client, allow a large number of independent, concurrent accesses to the same or different documents, support a full spectrum of playout control operations with minimal latency and without extra BW reservation, and support buffered as well as bufferless clients.

In our ongoing work, we are refining our multilevel data layout, real-time scheduling algorithms and developing efficient admission control procedures. We will demonstrate these ideas in an implementation of our prototype architecture. We also plan to extend our ideas to a server with a hierarchical storage system that comprises of a large on-line and near-line (robotically controlled optical juke box) storage.

Acknowledgements

Some of the work reported here was conducted during the first author's summer internship at the NEC USA, Inc., C&C Research Labs, Princeton. We are grateful to NEC USA for this opportunity and in particular thank Dr. Dipankar Raychaudhuri, Head of the Systems Lab and Dr. Kojiro Watanabe, General Manager of C&C labs. We also thank Dr. Arif Merchant and Daniel Reiniger at the NEC C&C labs for very useful discussions.The first author would also like to thank Dr. Andy Fingerhut, from Applied Research Labs (ARL) at the Washington University in St. Louis, for useful suggestions during the early stages of our work.

/

References

[1] —-, "Desktop Video Data Handbook," *Philips Semiconductors*, pp. 3-162-190, May 1993.

[2] Buddhikot, M., Parulkar, G., and Cox, Jerome, R. Jr., "Design of a Large Scale Multimedia Storage Server," invited for publication in *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] Cao, Pei, et al., "The TickerTAIP Parallel RAID Architecture," *Proceedings of the 1993 International Symposium on Computer Architecture*, May 1993.

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

[5] Chen, P., et al., "RAID: High-performance, Reliable Secondary Storage," submitted to *ACM Computing Surveys.*

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

[7] Daigle, S., "Disk Scheduling for Continuous Media Data Streams," Master's Thesis, Carnegie Mellon University, Dec. 1992.

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

[9] 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.

[10] Guttag, K., Gove, R., and Aken, V., "A Single-Chip Multiprocessor for Multimedia: The MVP," *IEEE Computer Graphics and Applications*, pp. 53-64, Nov. 1992.

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

[12] Hylton, Todd., Coffey., K., Parker, A., and Kent, H., "AdStaR Scientists Detect Giant Magnetoresistance in Small Magnetic Fields, Using Easy to Make Sensor," *SCIENCE*, August, 1993.

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

[14] 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.

[15] Keeton, K., et al., "Storage Alternatives for Video Service," *Proceedings of the* 13^{th} *IEEE Symposium on Mass Storage Systems*, pp. 100-105, Annecy, France, June 12-16, 1994.

[16] 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.

[17] Little, T., D., et al., "A Digital On-demand Video Service Supporting Content-based Queries," *Proceedings of ACM Multimedia'93*, Anaheim, CA, pp 427-436, Aug. 1993.

[18] 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.

[19] LoVerso, S., Isman, M., et al. "sfs: A Parallel File System for the CM-5," *Proceedings of USENIX Summer Conference*, June, 1993.

[20] 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

[21] Miller, G., Baber, G., and Gilliland, M., "News On-Demand for Multimedia Networks," *Proceedings of ACM Multimedia'93*, Anaheim, CA, pp. 383-392, Aug. 1993.

[22] 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.

[23] Pancha, P., and Zarki, M., "MPEG Coding for Variable Bit Rate Video Coding," *IEEE Communications Magazine*, pp. 54-66, May, 1994.

[24] 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.

[25] Reddy, N. and Wyllie, J., "Disk Scheduling in a Multimedia I/O System," *Proceedings of ACM Multimedia'93*, Anaheim, CA, pp. 225-233, Aug. 1993.

[26] Reininger, D., and Raychaudhuri, D., "Bit-Rate Characteristics of a VBR MPEG Encoder for ATM Networks," *Proceedings of IEEE ICC'93*, Geneva, Switzerland, May, 1993.

[27] Rowe, L., Boreczky, J., and Eadds, C., "Indexes for User Access to Large Video Databases," *IS & / SPIE Symposium on Electronic Imaging Science and Technology*, Conference 2185, 1994.

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

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

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

[35] Zou, W., Y., "Digital HDTV Compression Techniques for Terrestrial Broadcasting," *High Definition (HD) World Review*, Vol. 3, No. 3, pp. 4-10, 1992.