Efficient Data Layout, Scheduling and Playout Control in MARS*

Milind M. Buddhikot and Gurudatta M. Parulkar

Washington University in St. Louis
Computer & Communications Research Center
Dept. of Computer Science
St. Louis, MO 63130-4899

Abstract. Large scale on-demand multimedia servers, that can provide independent and interactive access to a vast amount of multimedia information to a large number of concurrent clients, will be required for a wide spread deployment of exciting multimedia applications. Our project, called Massively-parallel And Real-time Storage (**MARS**), is aimed at prototype development of such a large scale server. This paper primarily focuses on the distributed data layout and scheduling techniques developed in this project. These techniques support a high degree of parallelism and concurrency, and efficiently implement various playout control operations, such as *fast forward, rewind, pause, resume, frame advance* and *random access.*

1 Introduction

The primary focus of this paper is on the distributed data layout, scheduling and playout control algorithms developed in conjunction with our project, called the *Massively-parallel And Real-time Storage* (MARS). This project is aimed at the design and prototyping of a high performance large scale multimedia server that will be an integral part of the future multimedia environment. The five main requirements of such servers are: support potentially thousands of concurrent customers all accessing the same or different data; support large capacity (in excess of terabytes) storage of various types; deliver storage and network throughput in excess of a few Gbps; provide deterministic or statistical *Quality Of Service* (QOS) guarantees in the form of bandwidth and latency bounds; and support a full spectrum of interactive stream playout control operations such as *fast forward (ff), rewind (rw), random access, slow play, slow rewind, frame advance, pause, stop-and-return and stop.* Our work aims to meet these requirements.

1.1 A Prototype Architecture

Figure 1 shows a prototype architecture of a MARS server. It consists of three basic building blocks: a cell switched ATM interconnect, storage nodes, and the

* This work was supported in part by ARPA, the National Science Foundation's National Challenges Award (NCA), and an industrial consortium of Ascom Timeplex, Bellcore, BNR, Goldstar, NEC, NTT, Southwestern Bell, Bay Networks, and Tektronix.

central manager. The ATM interconnect is based on a custom ASIC called *ATM Port Interconnect Controller* (APIC), currently being developed as a part of an ARPA sponsored gigabit local ATM testbed [4]. The APIC is designed to support a data rate of 1.2 Gbps in each direction. Each storage node provides a large amount of storage in one or more forms, such as large high-performance magnetic disks, large disk arrays or a high capacity fast optical storage. The nodes that use optical storage can be considered as off-line or near-line tertiary storage. The contents of such storage can be cached on the magnetic disks at the other nodes. Thus, the collective storage in the system can exceed a few tens of terabytes. Each storage node may also provide one or more of the resource management functions, such as media processing, file system support, scheduling support, and admission control.

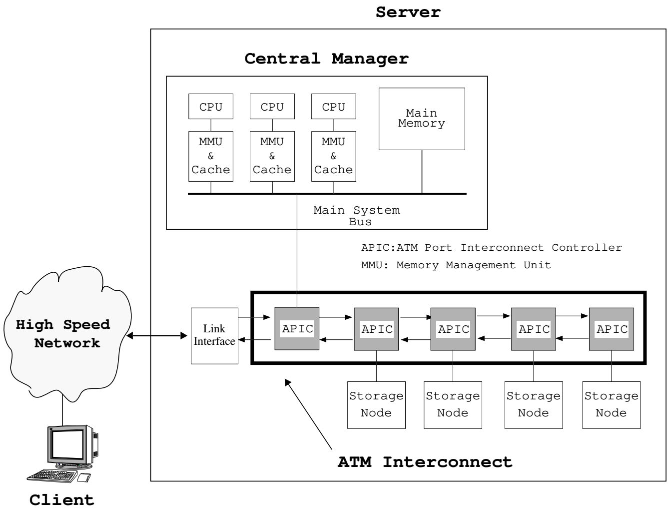

Fig. 1. A prototype architecture

The central manager shown in Figure 1 is responsible for managing the storage nodes and the APICs in the ATM interconnect. For every multimedia document,[2] it decides how to distribute the data over the storage nodes and manages 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/to 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

[2] A movie, a high quality audio file, an orchestrated presentation etc. are a few examples of a multimedia document.

not participate in actual data movement. This ensures a high bandwidth path between the storage nodes and the network.

Using the prototype architecture as a building block (called a *"storage cluster"*) and a multicast ATM switch, a large scale server can be realized. Both these architectures can meet the demands of future multimedia storage servers and have been described in greater detail in [1].

Note that these architectures easily support various on-demand multimedia service models, such as Shared Viewing (SV), Shared Viewing-with-Constraints (SVC) or *"near-video"*, and Dedicated Viewing (DV). However, in this paper, we assume that the server supports a retrieval environment using the DV service. This service model is a natural paradigm for highly interactive multimedia applications, as it treats every client request independently and does not depend on spatial and temporal properties of the request arrivals. Also, in this paper, we assume that each storage node uses magnetic disks or a disk array, such as commercial Redundant Arrays of Inexpensive Disks (RAID).

2 Distributed Data Layout

A data layout scheme in a multimedia server should possess the following properties: 1) it should support maximal parallelism in the use of storage nodes and be scalable in terms of number of clients concurrently accessing the same or different document, 2) facilitate interactive control and random access, and 3) allow simple scheduling schemes that can ensure periodic retrieval and transmission of data from unsynchronized storage nodes.

We use the fact that the multimedia data is amenable to spatial striping to distribute it hierarchically over several autonomous storage nodes within the server. One of the possible layout schemes, called *Distributed Cyclic Layout* (DCL_k), is shown in Figure 2. The layout uses a basic unit called *"chunk"* consisting of k consecutive frames. All the chunks in a document are of the same size and thus, have a constant time length in terms of playout duration. Different documents may have different chunk sizes, ranging from $k = 1$ to $k = F_{max}$, where F_{max} is the maximum number of frames in a multimedia document. 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 2, 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), the stream is slowed down by a factor of D from the perspective of each storage node or the throughput required per stream from each storage node is reduced by a factor of D. This in turns helps in masking the large prefetch latencies introduced by very slow storage devices at each node.

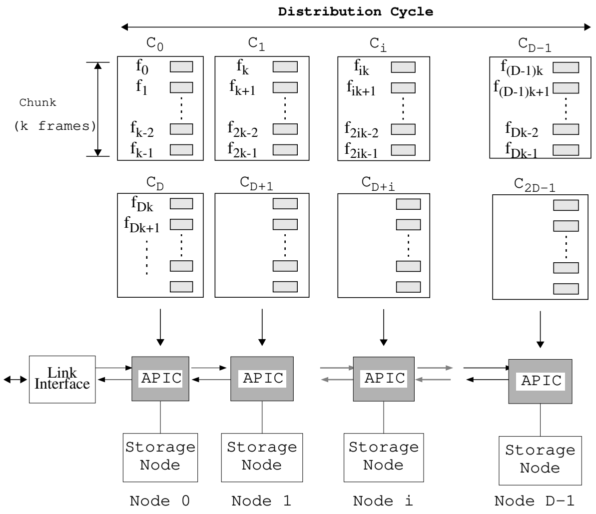

Fig. 2. Distributed Chunked Layout

2.1 Load Balance Property of Data Layouts

In the simple DCL_1 layout (recall chunk size $k = 1$), when a document is accessed in a normal playout mode, the frames are retrieved and transmitted in a linear (mod$D$) order. Thus, for a set S_f of any consecutive D frames (called "frame set"), the set of nodes S_n (called "node set") from which these frames are retrieved contains each node only once. Such a node set is called a balanced node set. A balanced node set indicates that the load on each node, measured in number of frames, is uniform[3]. However, when the document is accessed in an interactive mode, such as *ff* or rw, the load-balance condition may be violated. In our study, we implement *ff* and rw by keeping the display rate constant and skipping frames, where the number of frames to skip is determined by the *ff* rate and the data layout. Thus, *ff* may be realized by displaying every alternate frame, every 5^{th} frame, or every d^{th} frame in general. We define the fast forward (rewind) distance d_f (d_r) as the number of frames skipped in a fast forward (rewind) frame sequence. However, such an implementation has some implications for the load balance condition. Consider a connection in a system with $D = 6$ storage nodes, a DCL_1 layout, 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 $\{0, 2, 4, 6, 8, 10, \ldots\}$. This implies that in this example, the odd-numbered nodes are never visited for frame retrieval during *ff*. Clearly, it is desirable to know in advance what frame skipping

[3] The load variation caused by the non-uniform frame size in compressed media streams is compensated by adequate resource reservation and statistical multiplexing.

distances a layout can support without violating the load-balance condition. To this end, we state and prove the following theorem[4].

Theorem 1. *Given a* DCL$_1$ *layout over D storage nodes, the following holds true:*

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

Proof. We give a proof by **contradiction**. Let f be the 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:

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

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 layout, we can see that the set of frames assigned to the node n_p can be defined as follows:

$$F_p = \{\forall \ell \in \mathcal{N} : (p + \ell \times D) \mapsto n_p\} \tag{2}$$

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

$$\begin{aligned} (f+jd_f)-(f+id_f) &= kD \\ (j-i) &= k \times \frac{D}{d_f} \end{aligned} \tag{3}$$

Two cases that arise are as follows:

[4] 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.

- **Case 1: k is not a multiple of d_f:** If D and d_f are relatively prime[5], then, $\frac{D}{d_f}$ cannot be an integer. The left hand side (L.H.S) of Equation 3 being a difference of two integers, is also an integer. Hence, the left hand side and the right hand side of the above equation cannot be equal. Thus, the Equation 3 cannot be true, which is a contradiction.
- **Case 2: k is a multiple of d_f:** If this condition is true, then $(j - i) = k_1 \times D$, where $k_1 = \frac{k}{d_f}$. However, this contradicts our 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 claim 2 in the Theorem statement is also justified. The proof in the case of a rewind operation is similar and is not presented here.

Thus, as per this theorem, 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.

2.2 Staggered Distributed Cyclic Layouts

Now we will briefly describe a more general layout called *Staggered Distributed Cyclic Layout* (SDCL) and characterize the fast forward distances it can support without violating load balance. Figure 3 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 described earlier, the anchor node for successive distribution cycles is always fixed to the same node. On the other hand, for the layout in Figure 3, anchor nodes of the successive distribution cycles are staggered by one node, in a $(\bmod D)$ order. This is an example of a staggered layout with stagger factor of $k_s = 1$, and other staggered data layouts with non-unit stagger distance are possible. Note that DCL_1 is a special case of SDCL layout with the stagger distance of $k_s = 0$. In general, the SDCL can be looked upon as a family of distributed cyclic layouts, each with different stagger distance k_s and with different load-balance properties. The following theorem illustrates the load-balance properties of SDCL with $k_s = 1$. A similar result for the general case with non-unit stagger distance is not presented here.

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 f_a, with fast forward distance d_f, then the node set S_n is load-balanced, provided:*

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

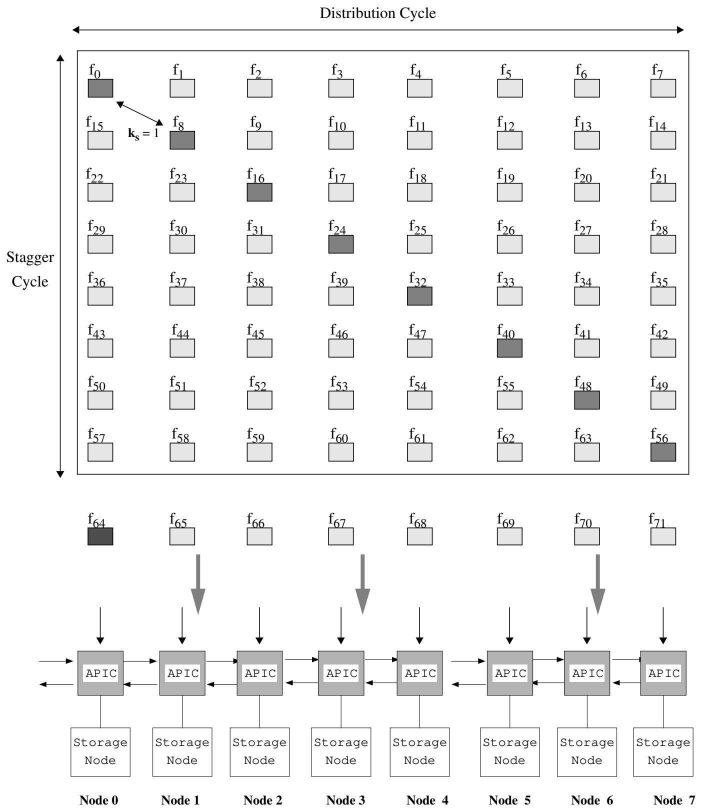

Fig. 3. Staggered Distributed Cyclic Layout(SDCL) with $k_s = 1$

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. $d_f = d_i + kD^2$ $(k > 0)$

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

A detailed proof of this theorem can be found in the current version of [2, 3]. The aforementioned two theorems together allow the server to provide clients a rich choice of *ff* (rw) speeds. Also, both these theorems are equally valid for a chunked layout with non-unit chunk size, if the *ff* (rw) is implemented by skipping chunks instead of frames. These results are useful in implementing *ff* (rw) on MPEG streams that introduce interframe dependencies and make it difficult to realize arbitrary frame skipping distances.

3 Distributed Scheduling

In this section, we illustrate the basic scheme and data structure 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.

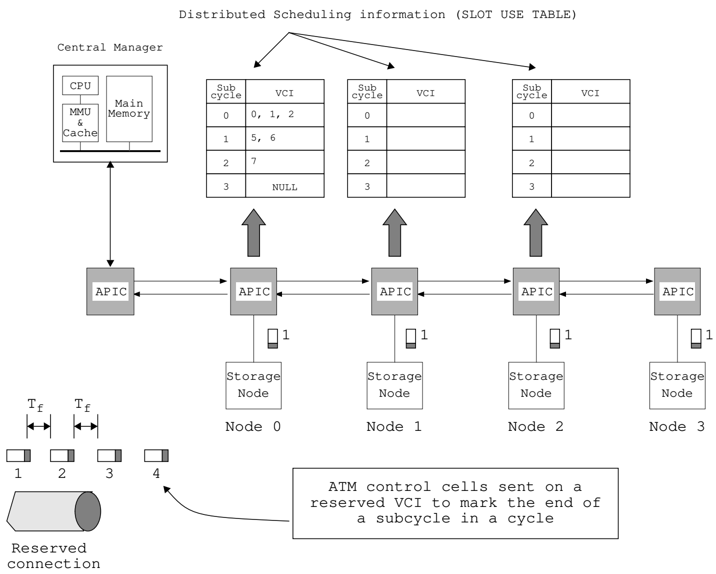

Fig. 4. Distributed scheduling implementation

In a typical scenario, a client sends a request to the server to access a multimedia document at the server. This request is received and processed by the central manager at the server, shown in Figure 1. 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, a network connection to the client, with appropriate QOS, is established. The central manager informs the storage nodes of this new connection, which in response create or update appropriate data structures such as *Slot Use Table*, *Connection State Block*, *Transmission Map*, etc. and allocate sufficient buffers. If an active client wants to effect a playout control operation, it sends a request to the server. The central manager receives it, and in response, instructs the storage nodes to change the transmission and prefetch schedule. Such a change can add, in the worst case, a latency of one scheduling cycle[6].

[6] A cycle is typically a few hundreds of milliseconds duration.

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. A ping-pong buffering scheme facilitates such overlapped prefetch and transmission. 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.

Table 1. Prefetch information at a node

VCI	No. of Frames	Frame IDs	Frame Address	Buffer Descriptor
VCI_1	1	8	$addr_8$	$bufdescr_1$
VCI_2	2	4,5	$addr_{4,5}$	$bufdescr_{4,5}$
VCI_3	1	1000	$addr_{1000}$	$bufdescr_{1000}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
VCI_{100}	1	8500	$addr_{8500}$	$bufdescr_{8500}$

Each storage node has its own independent prefetch cycle, in which it uses the prefetch information, illustrated in Table 1, for each active connection to retrieve the data. Specifically, the prefetch information consists of the following basic items: 1) Number of frames to be prefetched in the current cycle, 2) identification (ID) numbers of the frames to be fetched, 3) metadata required to locate the data on the storage devices at the storage node, and 4) the buffer descriptors that describe the buffers into which the data retrieved in the current cycle will be stored. Thus, for the example of Table 1, for $VCI = 2$, two frames $f = 4, 5$ need to be fetched using addresses $addr_4$ and $addr_5$ into the buffer described by $bufdescr_{4,5}$. Typically, the buffer descriptors and the buffers will be allocated dynamically in each cycle.

The 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. As shown in Figure 4, 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. The storage node counts these cells to know the current sub-cycle number and the start/end of the cycle.

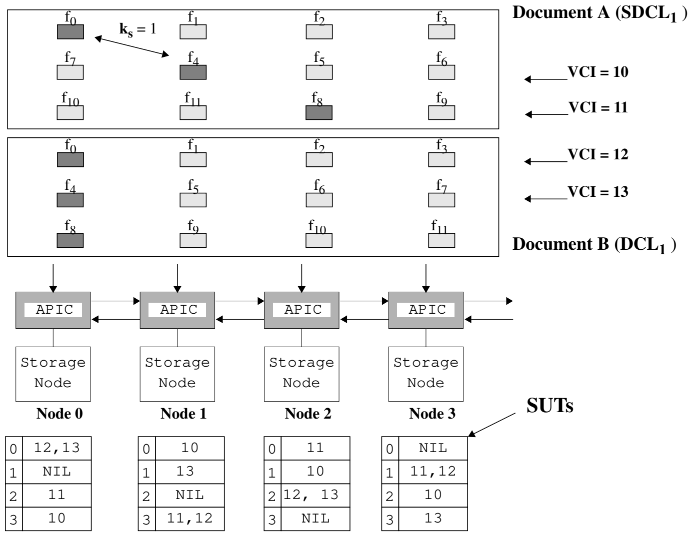

Fig. 5. An example of connections in different playout states

One of the main data structures used by each node to do transmission scheduling is the **Slot Use Table** (SUT). The i^{th} entry in this table lists the set of VCIs for which data will be transmitted in the i^{th} sub-cycle. This table is computed by the storage node or the central manager at the start of each cycle, using simple modulo arithmetic and load-balance conditions described earlier.

Table 2. Frame and node sets for all connections

VCI	Frame set S_f	Node set S_n
10	$4, 5, 6, 7$	$1, 2, 3, 0$
11	$8, 9, 10, 11$	$2, 3, 0, 1$
12	$0, 3, 6, 9$	$0, 3, 2, 1$
13	$4, 5, 6, 7$	$0, 1, 2, 3$

Figure 5 illustrates distributed scheduling with an example. This example shows two documents: Document A stored using the SDCL layout and Document B stored using DCL_1 data layout. Of the four active connections indicated, the connections with $VCI = 10, 11$ are accessing the document A and the connection with $VCI = 13$ is accessing document B in a normal play mode. On the other hand, $VCI = 12$ is accessing the document B in *ff* mode by skipping every third frame. Table 2 illustrates for each connection, the transmission frame set and ordered set of nodes from which it is transmitted during current transmission cycle. Using this table, the SUT at each node can be constructed. For example, node 0 transmits for connections $12, 13$ in slot 0, for connections 11 in slot 2, for connection 10 in slot 3, and remains idle during slots 1. The SUT at node 0 in Figure 5 records this information. Also, note that the SUTs at all the nodes contain exactly four non-zero entries and one `NIL` entry per cycle, indicating that the load is balanced over all the nodes. In this example, the documents have the same chunk size, however, the case when documents have different chunk sizes can be easily accommodated.

4 Implications of MPEG

Empirical evidence shows that in a typical MPEG stream, depending upon the scene content, I to P frame variability is about 3:1, whereas P to B frame variability is about 2:1. Thus, the MPEG stream is inherently variable bit rate. Clearly, when retrieving MPEG stream, the load on a node varies depending upon the granularity of retrieval. If a node is performing a frame-by-frame retrieval, the load on a node retrieving an I frame is 6–8 times that on a node retrieving a B frame. Hence, it is necessary to ensure that certain nodes do not always fetch I frames and others fetch only B frames. The variability of load at the GOP level may be much less than at the frame level, and hence selecting appropriate data layout and retrieval unit is crucial. In presence of concurrent clients, it is likely that each storage node can occasionally suffer overload, forcing the prefetch deadlines to be missed and thus, requiring explicit communication between the storage nodes to notify such events. The modifications to the basic scheduling required to realize this are not described here. Also, note that if the disk arrays used at the storage nodes are commercial RAIDs, the scheme for striping the data at each node is in effect fixed and may be inefficient for the statistical multiplexing of MPEG streams.

Another problem posed by MPEG like compression techniques is that they introduce inter-frame dependencies and thus, do not allow, frame skipping at arbitrary rate. This in effect means that *ff* by frame skipping can realize only a few rates. For example, the only valid fast forward frame sequences are IPP$\cdots$IPP$\cdots$ or [IIII. . .]. However, both these sequences increase the network and storage BW requirement enormously during *ff* and *rw*. The two ways to rectify this problem are as follows:

- **Store a intra-coded version of the movie along with a interframe coded version:** This option offers unlimited fast forward/rewind speeds, however, it increases 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.

The tradeoffs associated with these two options are currently being evaluated.

5 Conclusions

In this paper, we described data layout and scheduling options in our prototype architecture for a large scale multimedia server currently being investigated in the project **MARS**. We illustrated a family of hierarchical, distributed layouts called *Staggered Distributed Data Layouts* (SDCLs), that use constant time length logical units called chunks. For some of these layouts, we defined and proved a load-balance property that is required for efficient implementation of playout control operations such as fast-forward and rewind. Finally, we illustrated a distributed scheduling framework that guarantees orderly transmission of data for all active connections in any arbitrary playout state and addresses the implications of MPEG. These schemes and the associated tradeoffs are currently being evaluated using simulations.

References

1. Buddhikot, M., Parulkar, G., Cox, Jerome, R. Jr.: Design of a Large Scale Multimedia Storage Server. *Journal of Computer Networks and ISDN Systems (1995) 504-517.*
2. Buddhikot, M., Parulkar, G.: Scheduling, Data Layout and Playout Control in a Large Scale Multimedia Storage Server. Technical Report **WUCS-94-33**, Department of Computer Science, Washington University in St. Louis (1994).
3. Buddhikot, M., Parulkar, G.: Load Balance Properties of Distributed Data Layouts in MARS. Technical Report (in preparation), Department of Computer Science, Washington University in St. Louis. (1995).
4. Dittia, Z., Cox., J., Parulkar, G.: Design of the APIC: A High Performance ATM Host Network Interface. Proceedings of IEEE INFOCOM'95 (to appear) Boston (1995).

This article was processed using the LaTeX macro package with LLNCS style