Efficient Data Layout, Scheduling and Playout Control in a Multimedia Storage Server

Milind M. Buddhikot
milind@dworkin.wustl.edu
+1 314 935 4203

Gurudatta M. Parulkar
guru@flora.wustl.edu
+1 314 935 4621

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, currently being carried out as a part of NSF's National Challenge Award grant, is aimed at the design and prototyping of a high performance large scale multimedia server that will be an integral part of future multimedia environment. The five main requirements of such servers are: 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, 4) provide deterministic or statistical *Quality of Service* (QOS) guarantees in the form of bandwidth and latency bounds, and 5) support a full spectrum of interactive stream playout control operations such as *fast forward (ff), rewind (rw), slow play, slow rewind, frame advance, pause, stop-and-return and stop.* Our work aims to meet these requirements.

In order to make this extended abstract self-sufficient and motivate the new results, we have included some background material from our earlier publications [1, 2]. However, the full paper will use additional space to report the new results in greater detail. Also, due to space constraints, we have not included citations to a great body of related work in this field. However, the same have been extensively reported in [1, 2].

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 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.

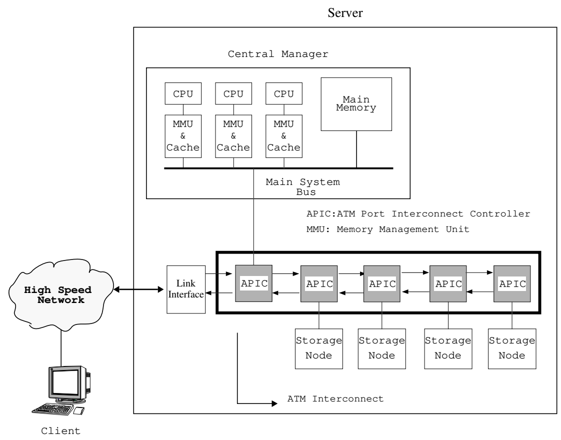

Figure 1: A prototype implementation

The central manager shown in Figure 1 is responsible for managing the storage nodes and the APICs in the ATM interconnect. For every media document [1], 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 not participate in actual data movement. This ensures a high bandwidth

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

path between storage and the network.

Using prototype architecture as a building block (called a "*storage cluster*") and a multicast ATM switch, a large scale server, illustrated in Figure 2, can be realized. Both these architectures can meet the demands of future multimedia storage servers and have been described in greater detail in [1].

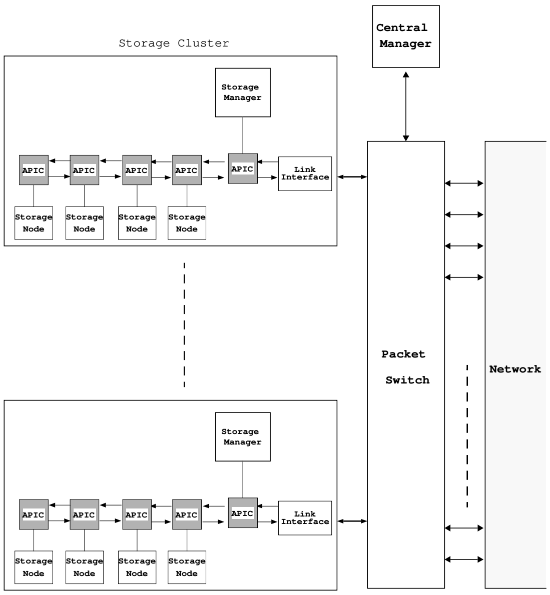

Figure 2: A scalable implementation

Note that these architectures easily support various on-demand multimedia service models, such as Shared Viewing (SV)(*"pay per view"*), Shared Viewing-with-Constraints (SVC) (also called *"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.

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 3. The layout uses a basic unit called *"chunk"* consisting of k consecutive frames. The chunk size can range from $k = 1$ to $k = F_{max}$, where F_{max} is the maximum number of frames in the multimedia document. For example, in case of MPEG compressed streams, the group-of-pictures (GOP) is one possible choice of chunk size. A chunk is always confined to one storage node. The successive chunks are distributed over storage nodes using a logical layout topology. For example, in Figure 3, the chunks have been laid out using a ring topology. Note that in this scheme, the two consecutive chunks at the same node are separated in time by kDT_f time units, D being the number of storage nodes and T_f the frame period for the stream. Thus, if the chunk size is one frame (DCL_1 layout), from the perspective of each storage node the stream is slowed down by a factor of D or the throughput required per stream from each storage node is reduced by a factor of D.

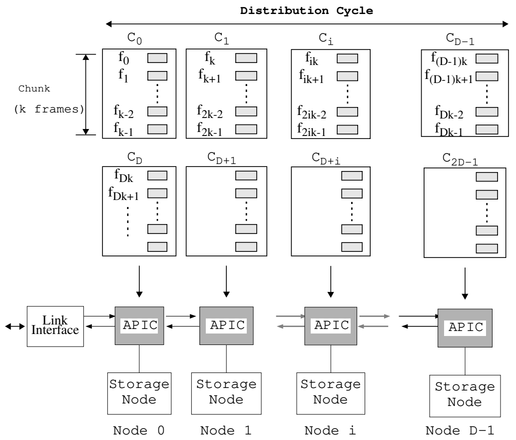

Figure 3: Simple Distributed Chunked data layout scheme

A chunk assigned to a storage node is stored on its storage devices, such as a disk array, as per a node-specific policy. For example, in the case 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. 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 the following factors: 1) amount of per-connection buffer available at the server, 2) type of network service used for data transport between the client and the server, 3) size of the buffer available at the client. For this paper, we assume that the client is **Bufferless**, that is, it has a few frames worth of buffer required by the decompression hardware. For example, a typical MPEG decoder may have a frame buffer and a buffer for at least three frames - for I, P and B frames. We are also exploring an alternate scenario of **Buffered Client**, which can store 100 to 200 frames.

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 of any consecutive D frames S_f (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 [2]. 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 frame sequence. However, such an implementation has some implications on 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 distances a layout can support without violating the load-balance condition. To this end, we state the following theorem[3].

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

- *If the fast forward (rewind) distance* d_f *is relatively prime to 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.*

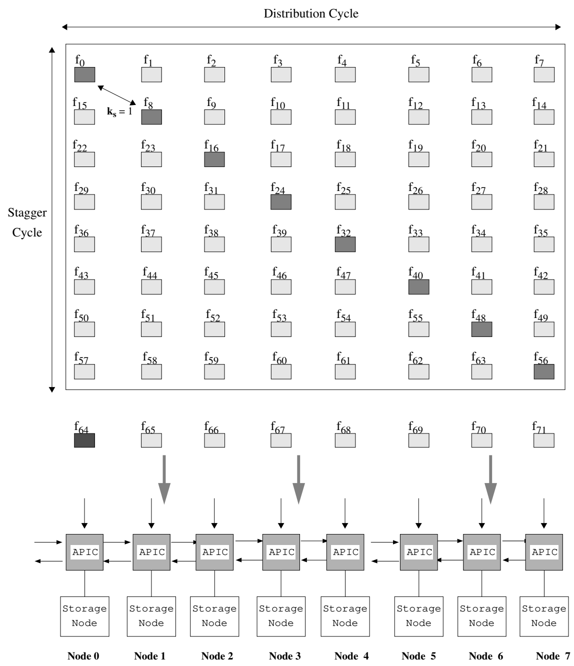

Figure 4: Staggered Distributed Cyclic Layout(SDCL) with $k_s = 1$

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.

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 4 illustrates an example of such a layout. We define a distribution cycle in a layout as a set of D frames, in which the first frame number is an integral multiple of D. The starting frame in such a cycle is called an anchor frame and the node to which it is assigned is called an anchor node. In the case of a DCL_1 layout, the anchor node for successive distribution cycles is always fixed to node 0. On the other hand, for the layout in Figure 4, anchor node of the successive distribution cycles is staggered by one node, in a $(\bmod D)$ order. This is an example of a staggered layout with stagger factor of $k_s = 1$. Other SDCL layouts with a non-unit stagger distance are possible. The following theorem illustrates the load-balance properties of SDCL with $k_s = 1$ [3] and the proof for more general case is being worked out. Note that DCL_1 is a special case of SDCL layout with the stagger distance of $k_s = 0$.

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

[3] 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.

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

- **Load balance condition for fast forward:** *If the fast forward starts from an anchor frame with a distance* d_f*, then the node set* S_n *consisting of nodes from which* D *frames in the fast forward frame set* S_f *are retrieved is load-balanced, provided:*
  1. $d_f = d_i$ *(where* $1 \leq i \leq p$*) 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.*

Note that these two theorems together allow the server to provide clients with 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.

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. 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 also creates or updates appropriate data structures such as *Slot Use Table*, *Connection State Block*, *Transmission Map*, etc. and informs the storage nodes of this new connection. Each storage node updates its scheduling information and allocates appropriate buffers. If an active client wants to effect a playout control operation, it sends a request to the server. The central manager receives it, and in response, changes the transmission and prefetch schedule. This can add, in the worst case, a latency of one cycle [4].

[4] 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. The buffering scheme illustrated in Figure 5 facilitates such overlapped pre-fetch 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.

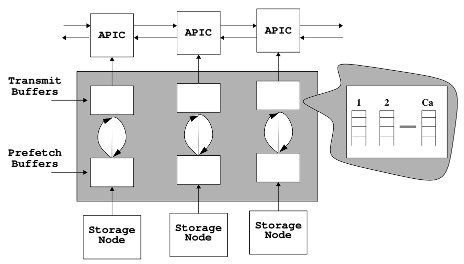

Figure 5: Distributed buffering

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 6, 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.

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 central manager at the start of each cycle and transmitted to each storage node. The computation uses simple modulo arithmetic and load-balance conditions described earlier.

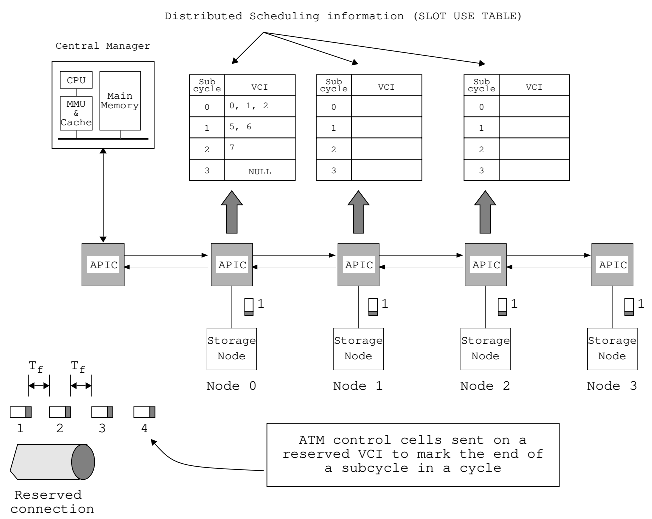

Figure 6: Distributed scheduling

4 Summary of Longer Version

Some of the details on the data layout, scheduling and playout control schemes have been reported in [3]. These results have not been published in our earlier research papers [1, 2]. If accepted, the longer version of our paper for NOSSDAV will contain the following:

1. **Distributed data layouts:** We will describe the distributed data layouts in greater detail and present the proofs of the Theorems stated earlier. Note that the size of the chunk has implications on the latency of playout control operations and resource utilization. Specifically, a smaller chunk size increases number of seeks at each disk of the storage node and thus, reduces the disk utilization. Larger chunk sizes can improve disk utilization, but increase worst case latency for playout control operations such as *ff* and *rw*. Also, *ff* implementation by chunk skipping may become visually unappealing for large chunk sizes. We will illustrate these tradeoffs in greater detail in our paper.
2. **Distributed scheduling and playout control:** The distributed scheduling will be described in greater detail and associated overheads in terms of APIC interconnect BW and buffer requirements will be shown to be minimal. We will illustrate the strong interaction between the distributed data layout and scheduling and detail the hardware and software support required at each storage node for efficient realization of such scheduling. Specifically, we will present the detailed algorithms and the data structures for realizing full spectrum of interactive operations for homogeneous (i.e. all connections require the same display rate) as well as non-homogeneous connections (connections require different display rates). Also, we will show that the worst case latency experienced by any playout control operation is of the order of a few hundred milliseconds.
3. **Coping with MPEG like compression:** 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 don't 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 unit is crucial. Also, MPEG like compression techniques introduce inter-frame dependencies and thus, don't 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$\cdots$IPP...] or [IIII...]. However, both these sequences increase the network and storage BW requirement enormously during *ff* and *rw*. We will illustrate our solutions, again within the distributed data layout and scheduling framework, to problems posed by MPEG.

References

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

[2] Buddhikot M., Parulkar, G., and Cox, Jerome, "Distributed Data Layout, Scheduling and Playout Control in a Large Scale Storage Server," *Proceedings of the Sixth International Workshop on Packet Video*, Portland, Oregon, 4 pages, Sept 26-27, 1994.

[3] Buddhikot, M., and 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, Dec. 1994.

[4] Dittia, Z., Cox., J., and Parulkar, G., "Catching up with the networks: Host I/O at gigabit rates," *To appear in Proceedings of INFOCOM'95*, Also, *Technical Report WUCS-94-11,* Washington University in St. Louis, July, 1994.