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

Milind M. Buddhikot (milind@dworkin.wustl.edu),
Guru M. Parulkar (guru@flora.wustl.edu),
Jerome R. Cox, Jr. (jrc@hobbs.wustl.edu),
Department of Computer Science, Washington University in St. Louis, St. Louis 63130

Introduction

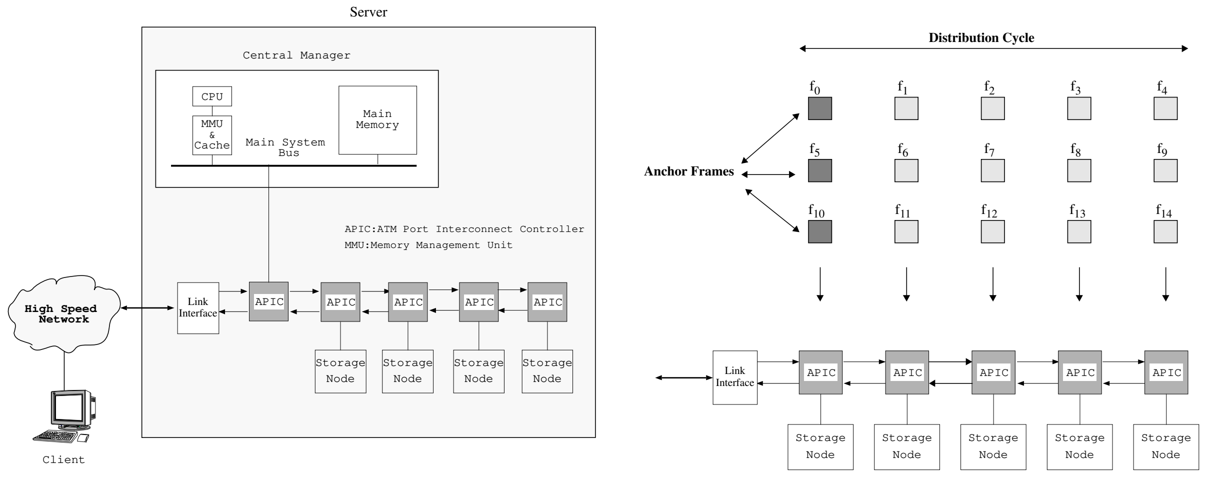
Large scale multimedia servers will be an integral part of future multimedia applications. The important requirements of such servers are: 1) support potentially thousands of concurrent customers all accessing the same or different data 2) support large capacity (in excess of terabytes) storage of various types 3) deliver storage and network throughput in excess of a few Gbps, 4) provide deterministic or statistical 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.*

The existing network based storage servers suffer from serious network and storage I/O bottlenecks and will not be able to meet the aforementioned requirements. Therefore, designing large scale high performance servers is a challenging task that requires significant architectural innovation. To this end, we have undertaken a project called the *Massively-parallel And Real-time Storage* (MARS). The motivation and the details of this architecture are presented in [1]. This paper focuses on the implications of playout control operations on the data layout and scheduling schemes.

A Prototype Architecture

Figure 1 shows a prototype architecture of a MARS server. It consists of two basic building blocks: the *ATM* based interconnect and the storage node. 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 central manager shown in the figure is responsible for managing the storage nodes and the APICs in the ATM interconnect. For every media document, it decides how to distribute the data over the storage nodes and manages the associated meta-data information. It receives the connection requests from the remote clients and based on the availability of resources and the QOS required, admits or rejects the requests. For every active connection, it also schedules the data read/write from the storage nodes by exchanging appropriate control information with the storage nodes. Note that the central manager only sets up the data flow between the storage devices and the network and does not participate in actual data movement. This ensures a high bandwidth path between storage and the network.



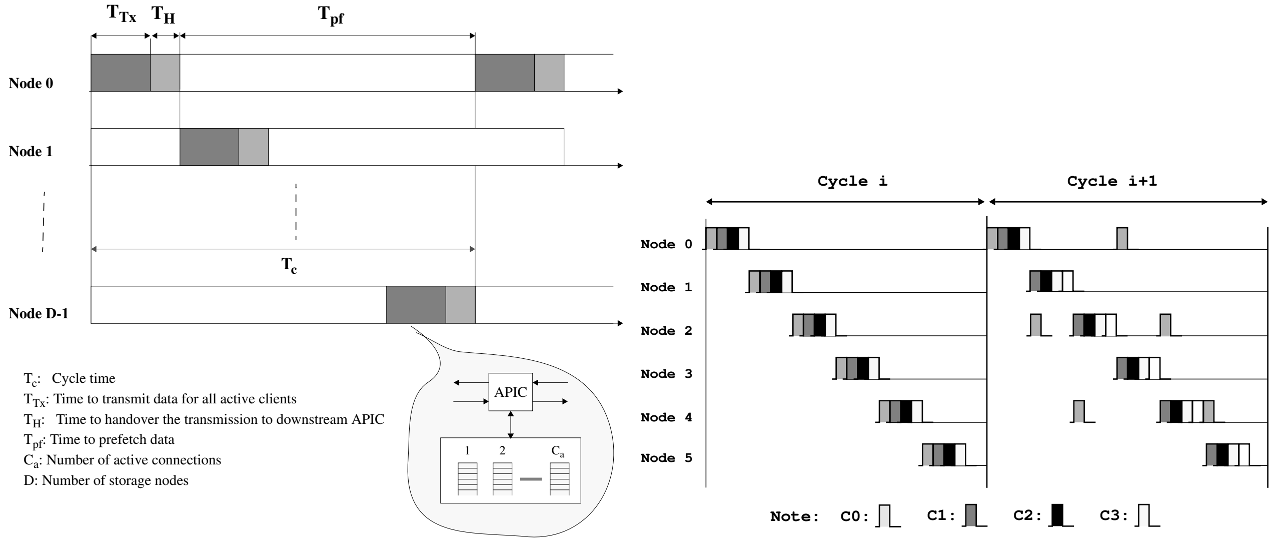
(a) A prototype implementation of a MARS server

(b) Layout example

Figure 1: A prototype architecture and an example layout

Data Layout and Scheduling

The periodic nature of multimedia data is well suited to spatial distribution or striping. For example, a logical unit for video can be a single frame or a collection of frames. Each such logical unit or the parts of it can be physically distributed on different storage devices and accessed in parallel. In our architecture, we use this property of multimedia data to stripe it in a hierarchical fashion. We outline only one of the many possible data layout schemes that satisfy real-time playout requirements and allow a large number of concurrent accesses to the same or different data in a retrieval environment. It must be noted that the architecture, data layout, data compression, and scheduling interact very strongly, a detailed discussion of which can be found in [3].



(a) Scheduling Example (b) Revised schedule when C_0 fast forwards

Figure 2: Scheduling: normal playout and fast forward

Figure 1 (b) illustrates an example system with five storage nodes numbered 0 to 4 from left to right. The frames f_0, f_1, f_2, f_3, f_4 are assigned to nodes D_0, D_1, D_2, D_3, D_4 respectively. The frame f_5 is again assigned to node D_0, thus following a frame layout topology that looks like a ring. Given that there are D storage nodes, this Distributed Cyclic Layout (DCL) has the property that at any given node, the time separation between the successive frames of the stream is $D \times$ *Inter-frame time*. Thus, the effective period of each stream seen by a storage node is D times longer, which facilitates prefetching of data to mask the high rotational and seek latencies of the magnetic disk storage. Note that the granularity of data striping over the storage nodes can be in units of multiple frames called a *chunk* [3]. Also, depending on the nature of the storage at the node, the frames assigned to it can be stored in multiple ways. For example, in the case of a RAID, the blocks of a frame can be further striped on the disks of the RAID.

Next, we will present a simple scheme for scheduling data retrieval from storage nodes when clients are assumed to be bufferless. In this scheme, shown in Figure 2 (a), 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 APIC corresponding to a storage node reads the C_a buffers and transmits them over the interconnect to the network. Once this phase is over, the APIC sends control information (a 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 transmitted in the next cycle.

As explained in [1], with the number of storage nodes D=15 and a simple disk array capable of delivering sustained 5 MBps throughput at each node, as many as 110 standard MPEG streams or 27 HDTV compressed streams can be supported. It must be emphasized here that our scheme allows all the clients to concurrently and independently access the same stream or different streams.

Playout Control

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. 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). However, such playout control operations have strong implications on the data layout and scheduling in the server. For example, *fast forward* of a video stream can be implemented in two ways: **1) Rate Variation (RV)** approach, which achieves *ff* by increasing the display rate. This leads to an increase in the retrieval and transmission rate, or **2) Sequence Variation (SV)** scheme which maintains the display rate constant but skips frames. Given that the **RV** scheme requires extra resources in the form of storage and network bandwidth, we consider it to be inappropriate and have decided to explore the **SV** approach. Note that certain stream control functions such as *slow play*, *slow rewind* can only be implemented by the **RV** scheme. However, these operations reduce resource usage and therefore, are easier to realize. In this paper, we will restrict our attention to *ff* or *rw* operations and describe implications of the **SV** semantics on data layout and scheduling.

The simple data layout and scheduling scheme described above works for normal stream playout but not for *ff* and *rw*. Consider a connection in an example system with $D = 6$ 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 $\{0, 2, 4, 6, 8, 10, \ldots\}$. This implies that in this example, the odd-numbered nodes are never visited for frame retrieval during *ff*. Also, 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 2 (b) 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 $N_{play} = \langle D_0, D_1, D_2, D_3, \ldots D_5 \rangle$, which is identical for all connections. 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 $N_{ff} = \langle D_0, D_2, D_4, D_0 \ldots \rangle$ in the $(i+1)^{th}$ cycle. The transmission order for the rest of the connections is unchanged. Figure 3 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 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 pre-fetch 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 pre-fetched into the buffers. Hence, to achieve a smooth transition from normal playout to fast forward, we provide two buffers per connection: one buffer used to store the data being pre-fetched and the other used to transmit the previously pre-fetched data. This effectively decouples transmission and pre-fetching and allows transmission order to be modified on a per cycle basis.

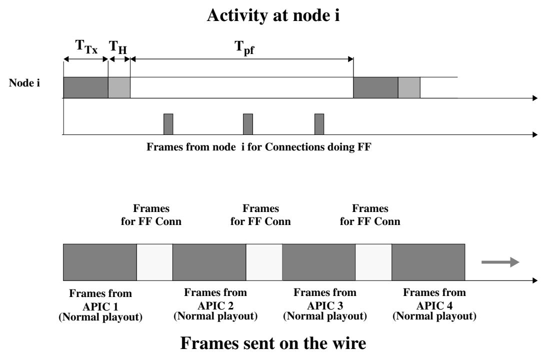


Figure 3: General case of M out of C_a connections doing fast forward

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 pre-fetch 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 pre-fetch cycle. Since cycle length is typically a few hundread msecs, and computation of transmission (pre-fetch) 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. 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 pre-fetch 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 have proved that if number of storage nodes is constrained to be a prime number and the fast forward distance by which frames are skipped is not a multiple of D, then the node set for fast forwarded frames is balanced. This can be seen in Figure 1 (b). The details of the theorem and it's proof are in [3].

Implications of MPEG

The structure of MPEG compressed streams has implications on the data layout, the scheduling and the playout control. In its simplest form an MPEG stream consists of a succession of group-of-pictures (GOP), each of which for example has a structure [IBBPBBPBB]. Some important properties of the three different types of frames in this stream are: 1) I frames are the most content intensive and hence the largest in size. 2) B frames have the smallest content and require both I and P frames to be decoded. 3) Since, the I frames are intra-coded, they can be transmitted independent of any other frames.

Given these facts, the simplest *ff* frame sequences for such a stream are [IPIPIP...] or [IIII...]. However, transmitting only I frames increases the network and storage BW requirement by at least a factor 2. Also, if the standard display rate is maintained, skipping all the frames between consecutive I frames, may make the effective fast forward rate unreasonably high. (e.g.: if I-to-I separation is 9 frames, perceived fast forward rate will be 9 times the normal playout rate.) Hence, it is desirable to reduce the display rate at the client. This in turn will reduce the transmission rate at the server and the network connection BW. However, this option needs to be further investigated.

Also, layout of an MPEG stream 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. Similarly, the transmission rate required for I is higher than that for P and B frames. It is desirable that the server be able to exploit this VBR characteristics to avoid always reserving peak storage and network 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. The details regarding this can be found in [3].

Work in Progress

In our ongoing work, we are developing and analyzing the distributed, multilevel real-time scheduling, data layout and admission control algorithms to guarantee QOS to a large number of clients during normal playout as well as a full spectrum of playout control operations. We plan to demonstrate our 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.

References

[1] Buddhikot, M., Parulkar, G., and Cox, Jerome, R. Jr., "Design of a Large Scale Multimedia Storage Server," *Proceedings of the INET'94/JENC5, Conference of the Internet Society and the Joint European Networking Conference,* Prague, June, 1994.

[2] 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, July, 1994.

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

Acknowledgements. This research was supported in part by ARPA, National Science Foundation, and an industrial consortium of ascom Timeplex, Bellcore, BNR, Goldstar, NEC, NTT, Southwestern Bell, SynOptics and Tektronix. In particular, we would like thank Dr. Arif Merchant from NEC USA, Inc., Princeton for many productive discussions. Also, we are grateful to NEC USA, Inc. for the opportunity to conduct some of the reported work at their research center in Princeton, during the summer internship of the first author.