Efficient Data Layout, Scheduling and Playout Control in MARS*

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

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

November 1, 1995

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 as a part of 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.

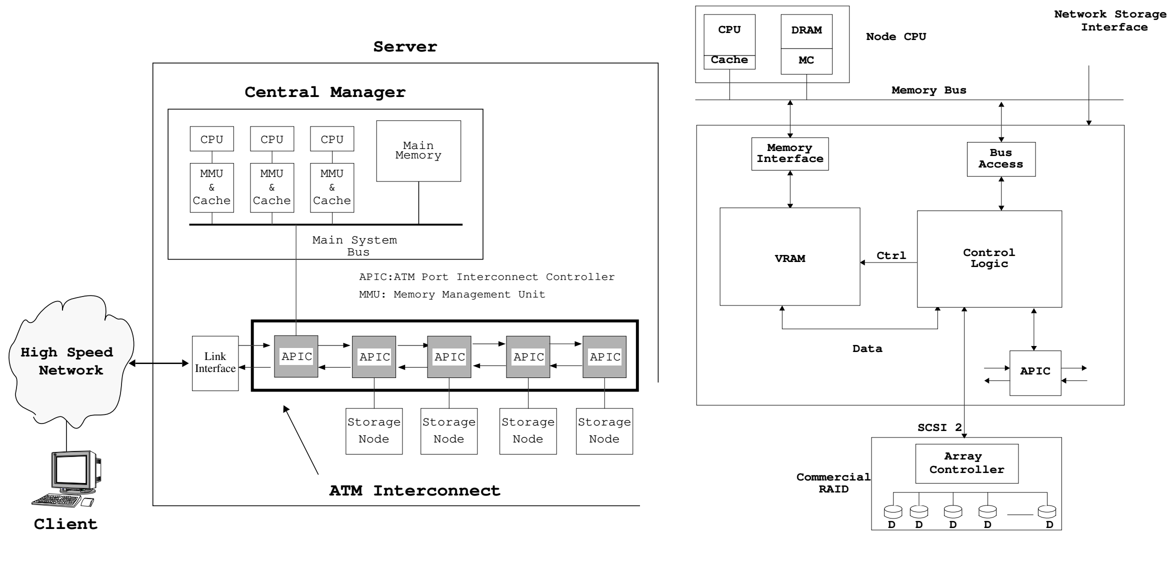
1.1 A Prototype Architecture

Figure 1 (a) 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

*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.

ASIC called *ATM Port Interconnect Controller* (APIC), currently being developed as a part of an ARPA sponsored gigabit local ATM testbed [15, 16]. The APIC is designed to support a data rate of 1.2 Gbps in each direction.

Each storage node in this architecture, illustrated in Figure 1 (b), is realized using a high performance processor subsystem with an APIC interface. One cost-effective and flexible way to realize the storage node is to use a Pentium PC with multiple SCSI interfaces to which various form of storage devices can be connected. The path from the local storage at the node to the APIC interconnect is optimized using a dual ported VRAM. The local storage at each node can be 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 runs a NETBSD UNIX operating system enhanced to handle multimedia data. Specifically, the filesystem and buffer management function in this OS have been modified to allow periodic retrieval of data and minimize buffer copying [13, 14]. Also, a new process scheduling technique called *Real Time Upcalls* (RTU) is employed to support periodic event processing required to provide QOS guarantees [25, 26, 27]. Each node may also provide other resource management functions, such as media processing and admission control.



(a) MARS architecture

(b) Example storage node design

Figure 1: A prototype architecture

The central manager shown in Figure 1 (a) is responsible for managing the storage nodes and the APICs in the ATM interconnect. For every multimedia 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

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

devices and the network and does 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 [3].

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).

The rest of this paper is organized as follows: Section 2 describes the basic two level data layout called *Distributed Chunked Layout* used in our architecture. It also describes performance metrics used to evaluate the layouts. Section 3 characterizes the load-balance properties of various distributed data layouts required for efficient implementation of interactive operations such as *ff, rewind,* and *fast search*. Section 4 describes a simple distributed scheduling scheme and the problems caused by *ff* and *rw*. It then also discusses prefetch and transmit options available to a storage node and then describes a scheduling framework that efficiently implements all playout control operations with minimal latency. Section 5 discusses the implications of MPEG compression on the design of the data layout and scheduling schemes. In Section 6 we present a cross section of related work in the area of media servers. Finally, Section 7 summarizes the conclusions of our work and discusses some of the on-going work.

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 the 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. In case of a Variable Bit Rate (VBR) video, a chunk therefore represents a Constant Time Length (CTL) but a variable data length unit. In case of a Constant Bit Rate (CBR) source, it also has constant size [8, 9, 20]. 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.

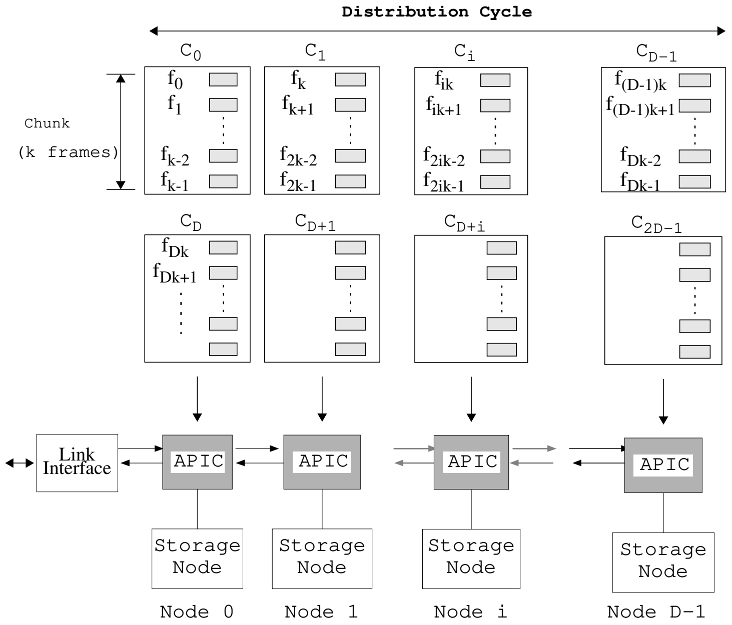


Figure 2: Distributed Chunked Layout

Note that each storage node stores the chunks assigned to it using a local storage policy. If the node uses a disk array, the way each chunk is stored or striped on the array depends on the type of the disk array. For example, a RAID-3 disk array will use byte striping, whereas a RAID-5 will use block striping. Other striping techniques such as [11] are also possible. Clearly, our data layout is hierarchical and consists of two levels: Level-1 which decides the distribution of data over storage nodes, and Level-2 which defines how the data assigned to each node by Level-1 layout will be stored on the local storage devices.

2.1 Performance Metrics

We will evaluate Level-1 distributed layouts using two performance metrics. The first one called *parallelism* (P_f), is defined as the **number of storage nodes participating concurrently in supplying the data for a document f.** The second metric called *concurrency* (C_f) defines the **number of active clients that can simultaneously access the same document f.**

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

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

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

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

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

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

The parallelism $P_f(n)$ supported by the Level-2 layout at a node n decides for each active connection, how many of the local storage devices are accessed in parallel to retrieve the data assigned by Level-1 layouts. For example, in case of a RAID-3, all disks participate in transfer of every data access, whereas in case of a RAID-5 only a subset of all the disks may be accessed. Clearly, a Level-2 layout should support high parallelism, even distribution of load, and high disk utilization, and thus, in turn support good statistical multiplexing of large number of retrievals of VBR streams.

In this paper, we will discuss only Level-1 layout and will not concern ourselves with evaluation of various Level-2 layouts.

3 Load Balance Property of Data Layouts

In this section we will illustrate certain load balance properties of Level-1 data layouts that are essential for efficient implementation of interactive operations such as *ff* and *rw*. For the sake of ease of exposition, we will present all our discussion and results in terms of a DCL_1 layout with chunk size $k = 1$. However, as we point out later these results are equally useful for other layouts with non-unit chunk size.

Consider a simple DCL_1 layout (recall chunk size $k = 1$) described in Section 2. 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 that maximizes parallelism 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 (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*. Thus, when a connection is being serviced in *ff* mode, the load measured in terms of the number of frames retrieved doubles for even numbered nodes and reduces to zero for odd numbered nodes. In other words, the parallelism P_f is reduced from D to $D/2$ during *ff* of a connection and the concurrency must be proportionately reduced. Clearly, in presence of a large number of connections independently exhibiting interactivity, this can lead to occasional severe load imbalance in the system and can make it difficult to satisfy the QOS contract agreed upon with each client at the time of connection setup. Thus, if we can ensure that P_f and consequently, C_f are unaffected during *ff* or *rw*, we can guarantee load-balance situations. One way to do this, is to use only those frame skipping distances that do not affect P_f and C_f. We call such skipping distances as *"safe skipping distances"* (SSD). Thus, given a data layout, we want to know in advance all the SSDs a distributed data layout can support. In order to provide a rich choice of *ff* (and *rw*) speeds, the number of such SSDs should be maximized. To this end, we state and prove the following theorem[3].

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:

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

Without any loss of generality, assume that two frames $f + id_f$ and $f + jd_f$, are mapped to the same node n_p. Since any two frames mapped to the same node differ by an integral multiple of D, we have

$$(j - i) \quad = \quad k \times \frac{D}{d_f} \tag{1}$$

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

[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.

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, then, $\frac{D}{d_f}$ cannot be an integer. However, $(j-i)$ is an integer. Thus, the Equation 1 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. ∎

It is interesting to note that above theorem is in fact a special case of a basic theorem in abstract algebra which states "If a is a generator of a finite cyclic group G of order n, then the other generators of G are the elements of the form a^r where $gcd(r, n) = 1$" [17]. In the scenario described by Theorem 1 above, $n = D$ and the generator a is 1. Under the operation of addition, a^r is d_f. Thus, as per this basic theorem, 1 (normal playout) and d_f (*ff*/*rw*) generate a group (a set) of D nodes, such that all the nodes are covered once.

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. Also, given a value D, there are always some distances d_f such as when d_f is a multiple of D or has a common factor with D, that cannot be safely supported. Therefore, additional data layouts that can increase choice of such distances need to be explored.

3.1 Staggered Distributed Cyclic Layouts

Now we will describe a more general layout called *Staggered Distributed Cyclic Layout* (SDCL_k) 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 (mod$D$) 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. Clearly, DCL_k is a special case of SDCL_k layout with the stagger distance of $k_s = 0$. Also, note that SDCL_k is conceptually similar to right-symmetric parity placement in RAID-4/5 [10].

We will illustrate some of the special properties of this layout with an example. Let us consider example in Figure 3 and a *ff* implementation by skipping alternate frames (that is $d_f = 2$) starting from frame 0 . The original frame sequence $\{0, 1, 2, 3, 4, 5, 6, 7\}$ is then altered to $\{0, 2, 4, 6, 8, 10, 12, 14\}$. The node set for this new sequence is then altered from the balanced set $\{0, 1, 2, 3, 4, 5, 6, 7\}$ to $\{0, 2, 4, 6, 1, 3, 5, 7\}$. Clearly, this new node set is re-ordered but still is balanced. However, if $d_f = 3$, the similar node set is given as $\{0, 3, 6, 2, 5, 0, 4, 7\}$, which contains 0 twice and hence is unbalanced. It can be verified that cases $d_f = 4$, $d_f = 8$ and $d_f = m \times D$,

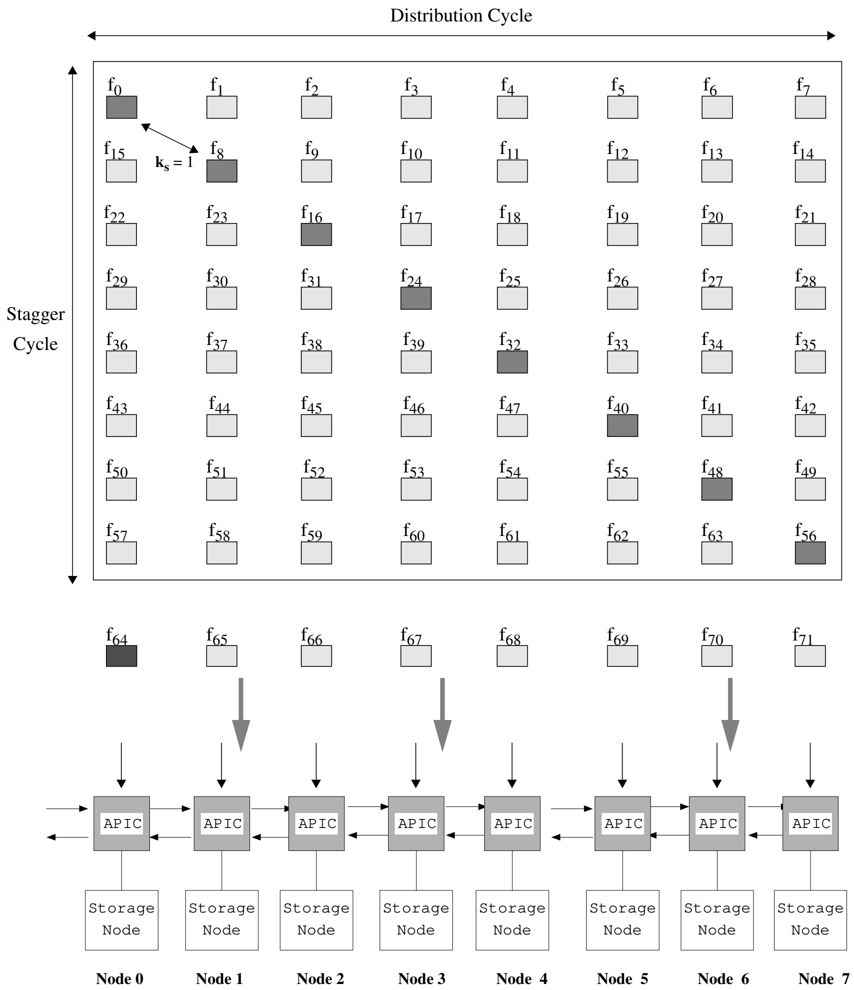


Figure 3: Staggered Distributed Cyclic Layout(SDCL$_1$) with $k_s = 1$

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.

The following theorem formalizes the load-balance properties of SDCL$_1$ with $k_s = 1$.

Theorem 2 *Given a* SDCL$_1$ *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:*

  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 [6]. An interesting thing to note here is that "most" of the distances that are unsafe for DCL_1 layout are safe for SDCL_1 layout. For example, if $D = 8$, $d_f = 2, 4, 8$ are unsafe for DCL_1 but safe for SDCL_1 layout.

3.2 Generalized Staggered Distributed Data Layouts

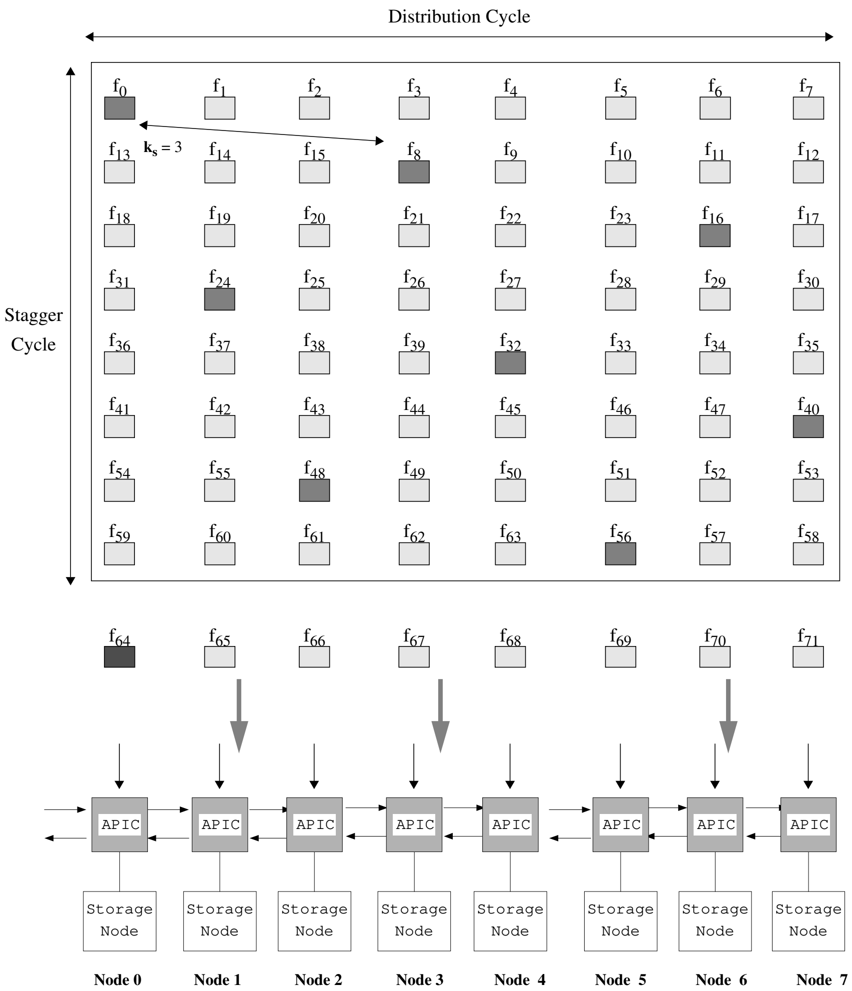


Figure 4: Generalized Staggered Distributed Layout with $k_s = 3$

In the previous section, we described a SDCL_k layout with stagger distance of 1 i.e $k_s = 1$. This distributed layout can be treated as a special case of a Generalized SDCL_k (G-SDCL_k) with non-unit stagger distance. In general, the G-SDCL_k can be looked upon as a family of distributed cyclic layouts, each with different stagger distance k_s and with different load-balance properties. To study load balance properties of these generalized layouts, consider a G-SDCL$_1$ with eight nodes ($D = 8$) and a stagger distance of 3 ($k_s = 3$) illustrated in Figure 4. Let the fast forward distance d_f be four frames ($d_f = 4$), which is a factor of $D = 8$. Assume that the fast forward starts from the frame $f = 8$. Then the fast forward frame set is $S_f = \{8, 12, 16, 20, 24, 28, 32, 36\}$. The set of nodes from which these frames are retrieved is given as $S_n = \{3, 7, 6, 2, 1, 5, 4, 0\}$ which is a balanced node set. On the contrary, a fast forward starting at the same frame with a distance $d_f = 3$ produces a node set $S_n = \{3, 6, 1, 7, 2, 5, 3, 6\}$ which is unbalanced. Similarly, it can be verified that $d_f = 2, 8$ produce balanced node sets, but $d_f = 5, 6, 7$ do not.

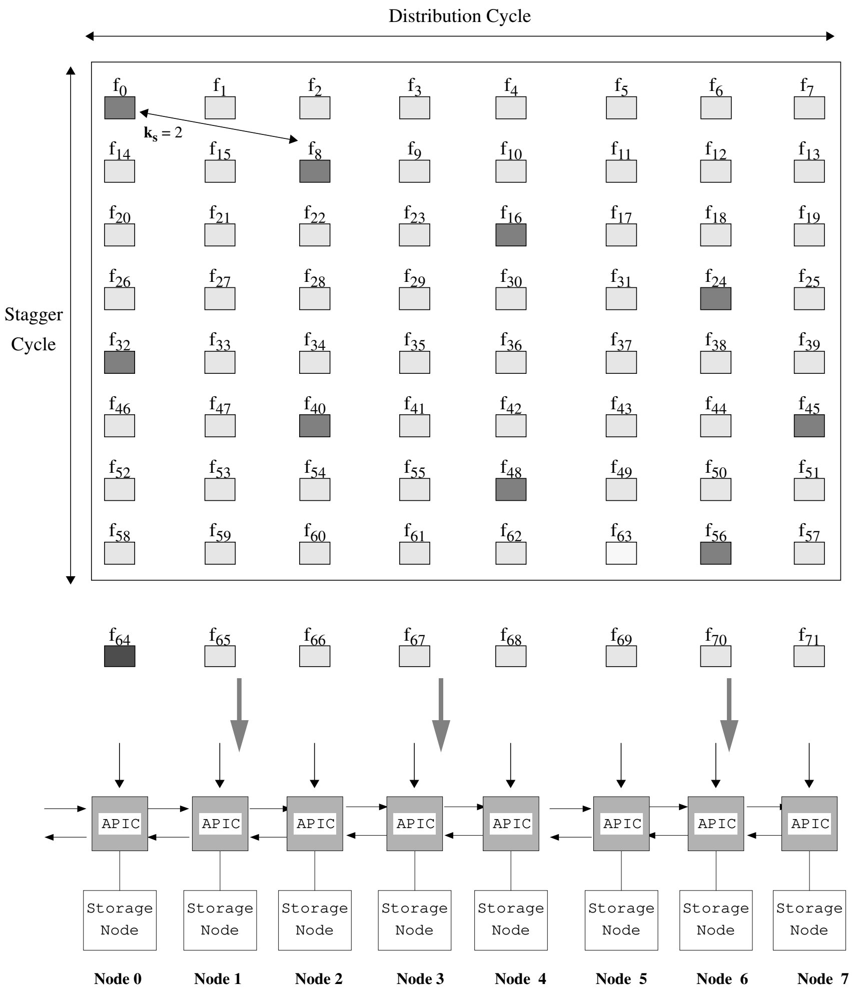


Figure 5: Generalized Staggered Distributed Layout with $k_s = 2$

Now let us consider a G-SDCL$_1$ with $k_s = 2$ shown in Figure 5. If the fast forward begins at frame $f = 16$ with a distance $d_f = 2$, the corresponding node set $\{4, 6, 0, 2, 6, 0, 2, 4\}$ is unbalanced. Similarly, distances $d_f = 3, 4, 5, 6, 7, 8$ produce unbalanced node sets.

From these examples we conjecture that if the the stagger distance k_s is relatively prime with the number of nodes then a result similar to the one mentioned in Theorem 1 is possible. Specifically, we conjecture that the following result can be proved.

Conjecture 1 *Given a* SDCL$_1$ *layout with a stagger distance k_s over D storage nodes and the numbers $d_1, d_2, d_3, \ldots d_p$ that the are factors of D, the following holds true.*

1. **Load balance condition for fast-forward:** *If the fast forward always starts from an anchor frame, with a fast forward distance d_f, the node mapping set S_n is load-balanced, provided:*
   (a) *k_s and D are relatively prime* **and**
   (b) $d_f = d_i$ $(1 \leq i \leq p-1)$ **or**
   (c) *$d_f = m \times D$, where m and D are relatively prime* **or**
   (d) $d_f = d_i + kD^2$ $(k > 0)$
2. **Load balance condition for rewind:** *A similar constraint defined in terms of anchor frame value, k_s and D holds for rewind operation.*

Thus, using one or more of G-SDCL$_k$ layouts, the server can provide a rich choice of *ff* and *rw* speeds to its clients. Note that all along we have assumed that frame skipping required for *ff* and *rw* operations can be implemented efficiently. This however may not be the case. We defer the discussion of this issue till Section 4.1.

4 Distributed Scheduling

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

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

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

As an example, consider a prototype with 15 storage nodes ($D = 15$), each with a disk array capable of providing sustained effective storage throughput of 5 MBps. Since the data layout described above allows seek and rotational latencies to be effectively masked, we can operate each disk array at its full effective bandwidth. Also, the simple scheduling scheme above allows each APIC to operate at full 1.2 Gbps. Thus, the aggregate storage and network throughput from the server is 1.2 Gbps, which can support as many as ≈ 54 compressed HDTV clients or ≈ 220 standard NTSC quality identical MPEG-2 clients.

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

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

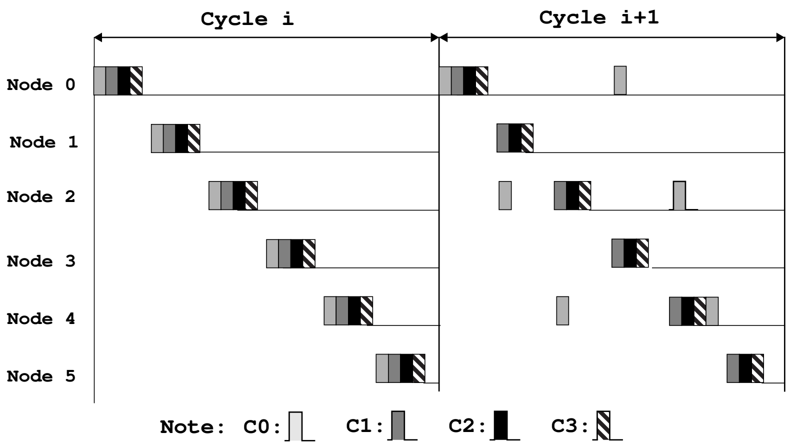
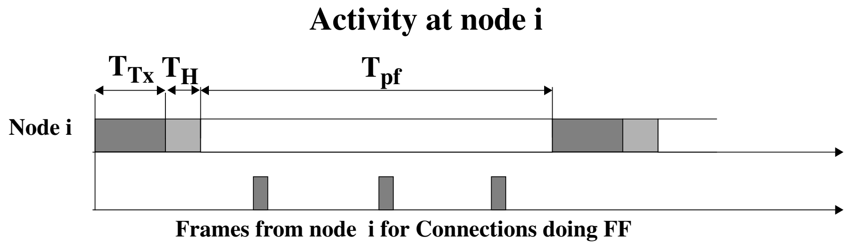


Figure 6: Revised schedule when C_0 performs fast forward

Figure 6 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 $S_{play} = \{0, 1, 2, 3, 4, 5\}$, which is identical for all connections. When the *ff* request for connection C_0 received in the i^{th} cycle becomes effective, the transmission order for it is altered to the ordered node set $S_f = \{0, 2, 4, 0 \ldots\}$ in the $(i+1)^{th}$ cycle. The transmission order for other connections is unchanged.



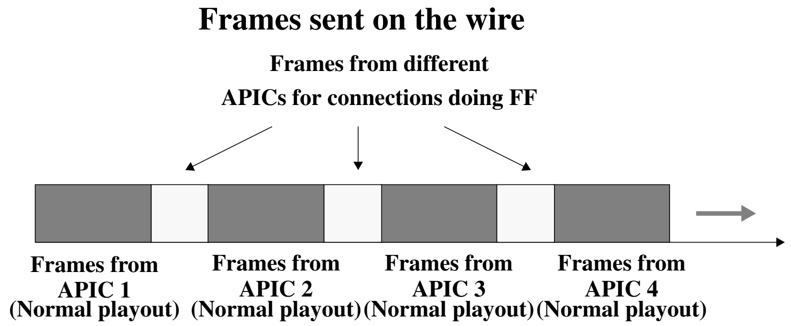


Figure 7: General case of M out of C_a connections doing fast forward

Figure 7 illustrates the transmission activity at node i and on the APIC interconnect when M out of C_a active connections are performing fast forward. At a typical node i the transmission occurs in multiple phases, one of which is for connections performing normal playout and the rest are for connections performing fast forward. These phases cannot be combined into a single phase as the transmission order of all the M connections performing fast forward is not identical. The sequence of frames appearing on the wire consists of two sequences: a sequence of frames transmitted from an APIC followed by frames transmitted from possibly all APICs for connections performing fast forward. It must however be noted that at any time, only one APIC transmits frames for a connection.

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

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

Clearly, for a scheduling scheme that synchronizes multiple storage nodes to satisfy above requirements, the prefetch and transmit operations for all active connections must be statistically multiplexed at each node. Since the connections are typically VBR in nature, the efficiency of such statistical multiplexing depends on the amount of data prefetched and transmitted per connection. The smaller the amount of data, higher is the burstiness in the storage retrievals and network transmissions, which in turn makes statistical multiplexing inefficient. So before we describe our scheduling scheme, we will look at various prefetch and transmit options available to a storage node and implications of *ff*/*rw* operations on them.

4.1 Granularity of Prefetch and Transmission at a Node

The prefetch and transmit options available to a node are described in terms of prefetch granularity F_g, defined as the amount of data prefetched per connection as a single unit in a scheduling cycle and transmit granularity T_g, defined as the amount of data transmitted per connection in a single cycle. Both F_g and T_g are specified in terms of the number of frames and strongly depend on per-connection buffer available at the server, type of network service used for data transport between the server and the client, buffer available at the client, and the design of the data layout. Since the prefetch and transmit follow a producer-consumer relation, the prefetch granularity (F_g) must always be greater than or equal to the transmit granularity (T_g) to ensure correct operation. Given this, the storage node has the following three options:

Option 1: Fetch and transmit frame at a time ($F_g = T_g = 1$)

In this option, irrespective of the chunk size used by the Level-1 layout, the storage node prefetches the data a frame at a time from the storage devices. Similarly, the APIC transmits a frame at a time. Clearly, this option minimizes buffer required at the client as well as the server. Since the length of the scheduling cycle is directly proportional to F_g, and T_g, this option leads to small scheduling cycles. This, in turn, guarantees that the worst case admission latency and response time for playout control operations is proportionately small.

However, there are several reasons this option may be undesirable: First, a chunk size more than one frame makes the retrieval pattern at a node bursty and requires stringent QOS guarantees during the time a storage node is servicing a connection. In other words, this option is suitable only for chunk size of one frame.

In case of MPEG streams, transmission on a frame-by-frame basis results in large peak to average bandwidth variation, as the bandwidth requirements vary drastically for I, P and B frames. Such variations are undesirable from the perspective of network transport.

Last, fetching data frame-by-frame increases seek overhead at the disks, as the disk head reads small amounts of data between consecutive fetches. Under near-full load (i.e when number of active clients and hence, requests per disk are large in number), this may require a non-trivial disk head scheduling policy to minimize rotational and seek latency by ordering disk requests. In short, this option is suitable for bufferless clients, DCL_1 layouts and streams with very high bandwidth (such as HDTV) and low burstiness.

Option 2: Fetch data in chunks and transmit it as a burst ($F_g = k, T_g = k, k > 1$)

In case of a G-SDCL_k data layout, when using option a storage node will fetch entire chunk of k frames as a single unit, whereas in case of a G-SDCL_1 layout it will fetch k consecutive frames assigned to it as a single unit. Clearly, such k frames can be stored contiguously on the local disk devices and thus, the seek and rotation overhead can be amortized over large transfers. This improves the disk utilization, but increases 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 at high rate such as link rate. However, this requires that the network support a transport service that allows reliable and periodic transmission of such large bursts of data, 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. Also, this approach will not work with heterogeneous client population in which some clients are bufferless and therefore, cannot buffer the chunk before it is played out.

Option 3: Fetch data in chunks and transmit it frame-by-frame ($F_g = k, T_g = 1$)

The implications of fetching data in units of chunks are identical to Option 2. In this case, the frame-by-frame transmission may use the rate computed over the duration of the entire chunk or a fraction of the chunk to minimize the peak to average bandwidth variation - a technique commonly called lossless smoothing [21]. Such smoothing is subject to availability of a small smoothing or playout buffer at the client side. At the client, this approach minimizes the buffer requirement. Clearly, this approach does not require and exploit any additional buffers that may be available at the client side.

Of these three options, third option shows the promise of providing good disk utilization for reasonable amount of per connection buffers at the client and server sides. However, the explanation of our scheduling scheme that follows uses a DCL_1 layout and $F_g = 1$, $T_g = 1$, because it is conceptually very easy to understand.

4.2 Implication of FF/RW Implementation

The discussion of load balance properties of data layouts in Section 3 assumes that frame skipping required to implement *ff* and *rw* operations can be performed efficiently. This however may not be true. For example, consider, $F_g = k$, that is Options 2 and 3. When the storage node serves a connection in normal playout mode, it can minimize the seek and rotational latency by fetching large chunks in single seek operation.

However, during the *ff* and *rw* implemented by frame skipping, individual frames must be read, which requires repositioning the disk head after every frame retrieval. For large skipping distances and small frames sizes that are common in compressed streams, each such read will suffer severe seek and rotation penalty. Such penalties can be minimized under heavy load, if prefetch load over multiple connection is randomly distributed over each disk and efficient disk scheduling algorithms such as those reported in [34] are used. However, under low or moderate loads, frame skipping may lead to poor disk utilization and cycle overflows. Also, if the Level-1 layout uses a non-unit chunk size, frame skipping causes load-imbalance. In other words, frame skipping is suitable for G-SDCL_1 Level-1 layouts.

An alternate approach to dealing with this problem is to always use a Level-1 layout with a chunk size of k frames and implement *ff* and *rw* by increasing the granularity of skipping to chunks. This kind of chunk skipping is analogous to segment skipping discussed in [11]. It has the advantage that during normal playout as well as *ff* and *rw*, chunks are read from the disk in much the same way without any additional seek/rotation penalties. Also, all the results mentioned in Section 3 for frame skipping on G-SDCL_1 layouts with any stagger distance apply to chunk skipping over G-SDCL$_k$ layouts. However, the visual quality of such chunk skipping is likely to be unacceptable at large chunk sizes.

The tradeoffs involved in these choices are currently under investigation. In our following discussion of scheduling scheme, we will assume **Choice 2**, as it is conceptually simpler and easier to describe. However, it is easily extended to **Choice 1**.

4.3 Scheduling Scheme

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 distributed scheduling, each storage node has to schedule reads from the disks in the disk array and optimize disk head movements. However, discussion of these local scheduling algorithms is beyond the scope of this paper.

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 (a). 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 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[4].

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.

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 per connection prefetch information consists of the following basic items stored in a data structure called **Prefetch Information Table** (PIT): 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

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

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	$addr_{8500}$	$bufdescr_{8500}$

described by $bufdescr_{4,5}$. Typically, the buffer descriptors and the buffers will be allocated dynamically in each cycle.

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

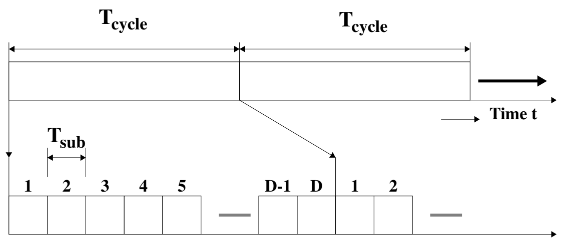


Figure 8: Cycle and Subcycle

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 at the start of each cycle, using connection state information and the load balance conditions described earlier in Theorem 1 and Theorem 2 computed using simple modulo arithmetic.

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 10 illustrates distributed scheduling with an example. This example shows two documents: Document A stored using the SDCL_1 layout and Document B stored using DCL_1 data layout. Of the four active connections

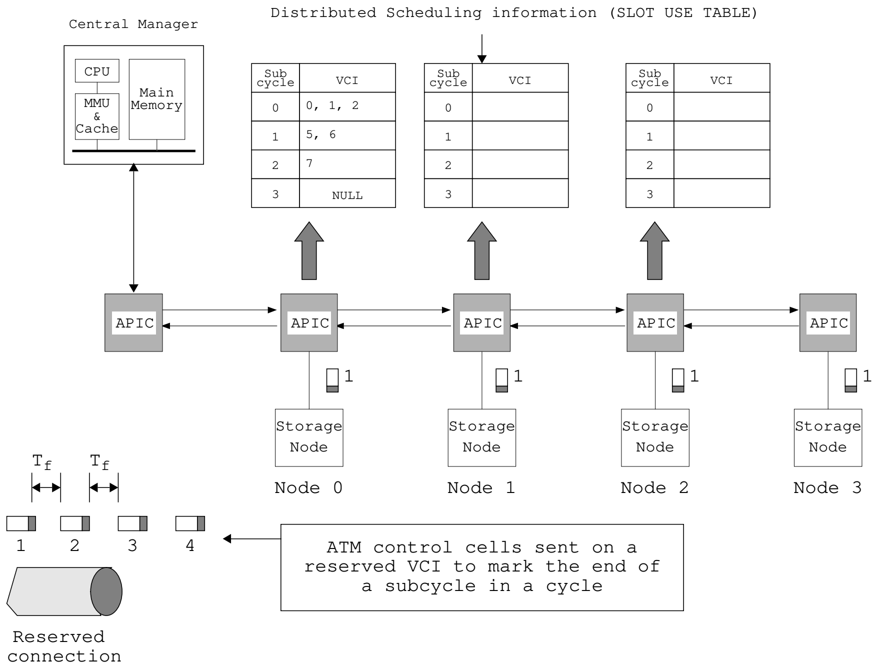


Figure 9: Distributed scheduling implementation

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 10 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.

Clearly, the various algorithms that process the interactive control requests such as *ff*, *rw*, *pause* etc. use the knowledge of the data layout to update the PIT and and the SUT data structures. The discussion of these algorithms is however beyond the scope of this paper.

5 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

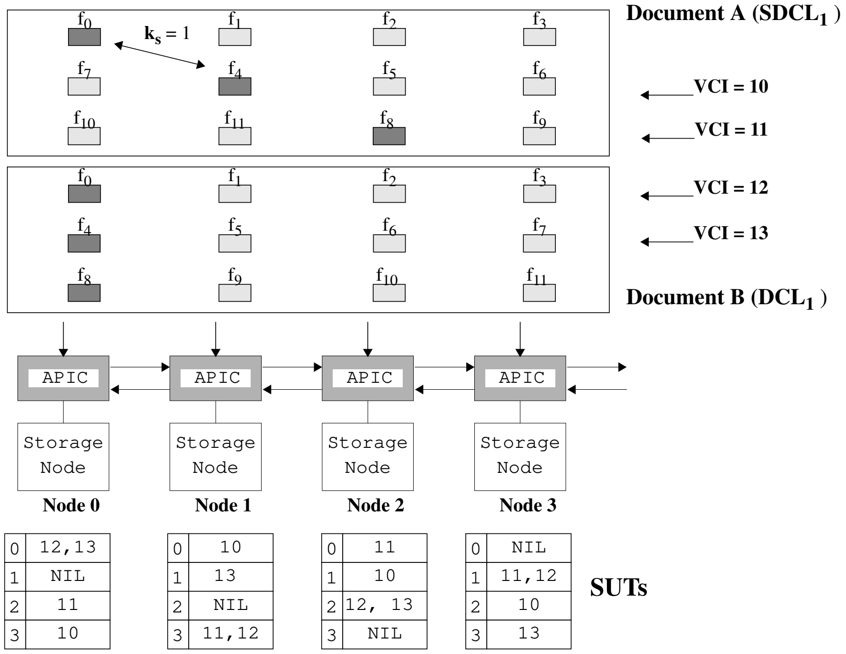


Figure 10: An example of connections in different playout states

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.

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 be realized only at a few rates. For example, the only valid fast forward frame sequences are [IPP···IPP···] or [IIII...]. There are two problems with sending only I frames on *ff*. First, it increases the network and storage bandwidth requirement at least three to four times. For example, a MPEG compressed NTSC video connection that requires 5 Mbps on an average during playout would require approximately 15 Mbps for the entire duration of the *ff*, if only I frames are transmitted. In the presence of many such connections, the network will not be able to easily meet such dramatic increases in bandwidth demand. Another problem is that if the standard display rate is maintained, skipping all 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. The two ways to rectify this problem are as follows:

- **Store a intra-coded version of the movie along with an 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. For example, a chunk can be a a GOP.

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

6 Related Work

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

6.1 High Bandwidth Disk I/O for Supercomputers

Salem et al. [29] represents some of the early ideas on using disk arrays and associated data striping schemes to improve effective storage throughput. Observing that large disk arrays have poor reliability and that small disks outperform expensive high-performance disks in price vs. performance, Patterson et al. [24] 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 [24]. 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 [12]. 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* [22]. Its architecture is based on a custom board design called *Xbus Card* that acts as a multiple array controller and interfaces to HIPPI as well as to FDDI networks. Though the measurements on RAID-II have demonstrated good I/O performance for large transfers, the overall solution employs FDDI, Ethernet and HIPPI interconnects and is ad-hoc. Also, it has not been demonstrated to be suitable for real-time multimedia, where the application needs are different than the needs of supercomputer applications.

6.2 Multimedia Servers

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

6.2.1 Multimedia File Systems

One of the early qualitative proposals for a on-demand video file system is reported in Sincoskie [30]. The work by Rangan et al. [28, 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.

Some of the recent work by Vin et. al. [33, 34] focuses on developing statistical admission control algorithms for a disk array based server capable of deterministic and/or statistical QOS guarantees.

Keeton et. al. discuss schemes for placement of sub-band encoded video data in units of constant playout length on a two dimensional disk array [20]. They report simulation results which conclude that storage of multi-resolution video permits service to more concurrent clients than storage of single resolution video. Similarly, Zakhor et. al. report design of schemes for placing scalable sub-band encoded video data on a disk array. They focus only on the path from the disk devices to the memory and evaluate using simulation, layouts that use CDL and CTL units mentioned earlier in Section 2 [8, 9]. However, none of these papers address issues in the implementation of interactive operations.

Chen et. al. report data placement and retrieval schemes for an efficient implementation of *ff* and *rw* operations in a disk array based video server [11]. Our work is completely independent and concurrent this work and has similarities and differences [4, 5]. Chen et. al.'s paper assumes a disk array based small scale server whereas our work assumes a large scale server with multiple storage nodes, each with a disk array. They define a MPEG specific data layout unit called a segment, which is a collection of frames between two consecutive I frames. Our definition of a chunk is a data unit which requires a constant playout time at a given frames/sec rate. So a segment Chen et. al.'s segment is a special case of our chunk. Chen et. al. discuss two schemes for segment placement: in the first scheme, the segments are distributed on disks in a round robin fashion, in much the same way as our DCL_k layout over multiple storage nodes. For *ff*/*rw* operation, they employ a segment selection method which ensures that over a set of retrieval cycles, each disk is visited only once. Thus, here the load balance is achieved over multiple retrieval cycles. In our case, the retrieval model is different. Our scheduling requires that load balance to be achieved in each prefetch cycle.

In second segment placement scheme, the segments are placed on the disk array in such a way that for certain fast forward rates, the retrieval pattern for each round contains each disk only once. Our SDCL_k layout over storage nodes with stagger distance of one is similar to this second segment placement scheme. However, our result is more general, as it characterizes many more safe skipping rates for *ff* and gives a condition for safe implementation of rewind operation. Also, we have extended the SDCL_k to G-SDCL_k which has been briefly discussed in Section 3.2.

6.2.2 Storage Server Architecture

Hsieh et al. evaluate a high performance symmetric multiprocessor machine as a candidate architecture for a large scale multimedia server [18]. They report measurements carried out on a fully configured Silicon Graphics high end symmetric multiprocessor workstation called SGI ONYX. They present two data layout techniques called *Logical Volume Striping* and *Application Level Striping* that use multiple parallel RAID3 disk arrays to increase concurrency and parallelism. The focus of their work until now has been on using measurements to characterize the upper limit in terms of maximum number of concurrent users in different scenarios, such as various type of memory interleaving, various data striping schemes, and multiple processes accessing a single file or different files.

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

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

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

Biersack et. al. have recently proposed an architecture very similar to ours, called *Server Array* [1, 2]. This architecture uses multiple geographically distributed storage nodes interconnected by external network such as a LAN working together as a single video server. Also, it proposes use of Forward Error Correction (FEC) codes to improve reliability. However, loosely-coupled distributed server approach in this work will make it difficult to support interactive operations with sub-second latency for large number of concurrent clients.

Lougher et al. have reported a design of a small scale *Continuous Media Storage Server* (CMSS) that employs append-only log storage, disk striping and hard real-time disk scheduling [23]. Their transputer based implementation handles very few simultaneous customers and supports small network bandwidth. This implementation is clearly not scalable for large number of users and for high bandwidth streams such as HDTV.

Tobagi et al. [31] report a small scale PC-AT and RAID based video server. Similarly, Kandlur et al. [19, 35], describe a disk array based storage server and present a disk scheduling policy called Grouped Sweeped Scheduling (GSS) to satisfy periodic requirements of multimedia data. However, all these server proposals are PC based and thus not scalable. Also they do not support high parallelism and concurrency.

7 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 *Generalized Staggered Distributed Data Layouts* (G-SDCL$_k$), 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 distributed data layouts and scheduling algorithms are currently being implemented in a testbed consisting of several Pentium PCs equipped with commercial disk arrays, and interconnected using an ATM switch. Experimental measurements will be used to evaluate our data layouts, scheduling schemes and and various node level layouts and scheduling options.

Acknowledgements

We will like to thank the three anonymous reviewers for their insightful comments. Specifically, we thank the reviewer who pointed out that our result on DCL$_1$ layouts (Theorem 1) is in fact a special case of an old theorem in abstract algebra [17]. Some of the work reported in this paper was conducted during the first author's summer research internship at the Computer and Communications (C&C) Research Laboratory at NEC Research Institute, Princeton, New Jersey. We will therefore like to thank Dr. Kojiro Watanabe and Dr. Deepankar Raychaudhuri for this invaluable opportunity. We would also like to thank Dr. Arif Merchant and Daniel Reiniger, both at the NEC C&C Research Laboratory for many useful discussions.

References

[1] Bernhardt, C., and Biersack, E., "A Scalable Video Server: Architecture, Design and Implementation," In *Proceedings of the Realtime Systems Conference*, pp. 63-72, Paris, France, Jan. 1995.

[2] Bernhardt, C., and Biersack, E., "The Server Array: A Scalable Video Server Architecture," To appear in *High-Speed Networks for Multimedia Applications*, editors, Danthine, A., Ferrari, D., Spaniol, O., and Effelsberg, W., Kluwer Academic Press, 1996.

[3] Buddhikot, M., Parulkar, G., and Cox, J., R., Jr., "Design of a Large Scale Multimedia Storage Server," *Journal of Computer Networks and ISDN Systems, pp. 504-517, Dec. 1994.*

[4] Buddhikot, M., Parulkar, G., M., and Cox, J., R., Jr., "Distributed Layout, Scheduling, and Playout Control in a Multimedia Storage Server," *Proceedings of the Sixth International Workshop on Packet Video*, Portland, Oregon, pp. C1.1 to C1.4, Sept. 26-27, 1994.

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

[6] Buddhikot, M., and Parulkar, G., M., "Load Balance Properties of Distributed Data Layouts in MARS," Technical Report (in preparation), Department of Computer Science, Washington University in St. Louis. 1995.

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

[8] Chang, Ed, and Zakhor, A., "Scalable Video Placement on Parallel Disk Arrays," Image and Video Databases II, IS&T/SPIE International Symposium on Electronic Imaging: Science and Technology, San Jose, Feb. 1994.

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

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

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

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

[13] Cranor, C., and Parulkar, G., " Universal Continuous Media I/O: Design and Implementation," Washington University Department of Computer Science, Technical Report 94-34, December 1994.

[14] Cranor, C., and Parulkar, G., "Design of a Universal Multimedia I/O System," *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, April 1995.

[15] 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, April, 1994.

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

[17] Fraleigh, B., J., "A First Course in Abstract Algebra," Addison-Wesley Publishing Company, pp. 52-53, 1967.

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

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

[20] Keeton, K., and Katz, R., "The Evaluation of Video Layout Strategies on a High Bandwidth File Server," *Proceedings of International Workshop on Network and Operating Support for Digital Audio and Video (NOSSDAV'93)*, Lancaster, U. K., Nov., 1993.

[21] Lam, S., Chow, S, and Yau, K., Y., D., "An Algorithm for Lossless Smoothing of MPEG Video," *Proceedings of ACM SIGCOMM '94*, London, August 1994.

[22] 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.

[23] 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.

[24] Patterson, 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.

[25] Raman, G., and Parulkar, G., "A Real-time Upcall Facility for Protocol Processing with QOS Guarantees," *SOSP Symposium on Operating Systems Principles (Poster Presentation)*, Dec. 1995.

[26] Raman, G., and Parulkar, G., "Quality of Service Support for Protocol Processing Within End-systems," *High-Speed Networking for Multimedia Applications*, Wolfgang Effelsberg, Otto Spaniol, Andre Danthine, Domenico Ferrari (Editors),

[27] Raman, G., and Parulkar, G., "Real-time Upcalls: A mechanism to provide Processing Guarantees," Department of Computer Science, Technical Report, (WUCS-95-06), Washington University in St. Louis.

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

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

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

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

[32] 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 Multimedia '94*, San Francisco, October 1994.

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

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