Design of Large Scale Multimedia-On-Demand Storage Servers and Storage Hierarchies *

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

Srihari Sampath Kumar
srihari@cs.wustl.edu
+1 314 935 7541

Guru M. Parulkar
guru@arl.wustl.edu
+1 314 935 4621

P. Venkat Rangan
venkat@cs.ucsd.edu
+1 619 534 7029

1 Introduction

Rapid advances in communication, high speed packet switching, mobile data communications, media compression, and processor and memory design will make a distributed multimedia computing infrastructure akin to that shown in Figure 1 feasible in the near future. Exciting interactive applications such as multimedia mail, orchestrated presentations, high quality on-demand audio and video, collaborative multimedia document editing, large digital libraries or multimedia archives, and virtual reality environments will become widespread in this infrastructure. A user may want to access these applications from a variety of end-systems such as a Personal Digital Assistant (PDA), a set top box with a television set, a high end personal computer at home or a workstation in the workplace. The interactive nature of these applications requires that the user be able to control playout or content of multimedia streams such as video, audio and graphics. Specifically, a user may perform playout control commands such as *fast-forward, rewind, pause/resume, fast-search* and *random access search* or may request media editing functions such as *clip, enhance, filter, restore etc.*. Also, most of these applications handle multimedia information in stored or static form. Therefore, the performance of such applications directly depends on the performance of storage servers that store such multimedia information.

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

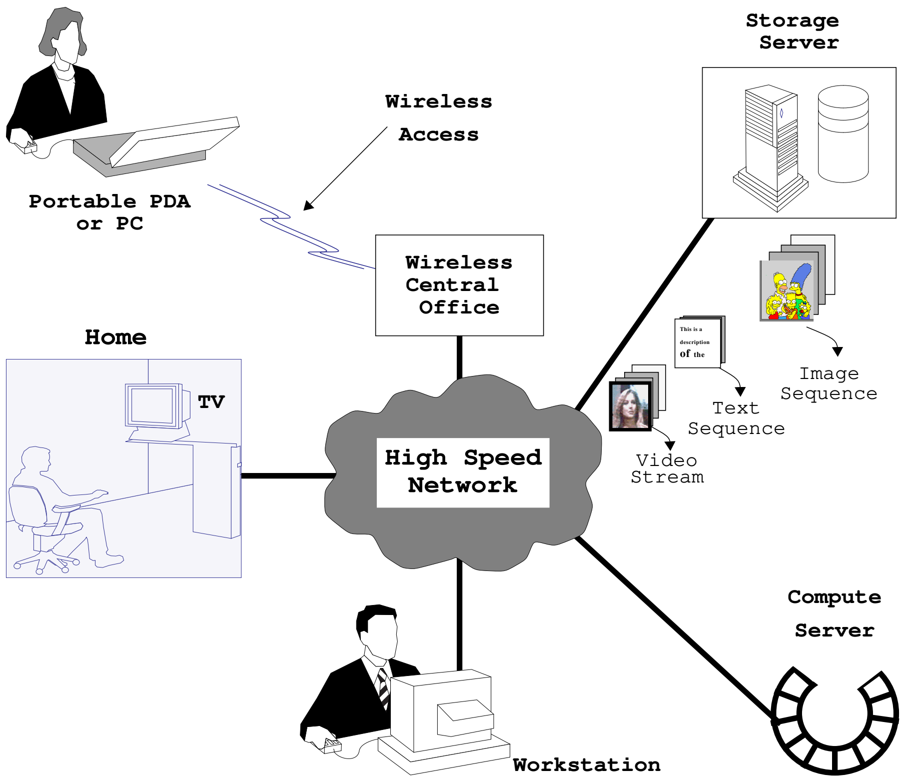

Figure 1. Future Distributed Multimedia Infrastructure

The requirements of such Multimedia-On-Demand (MOD) storage servers are uniquely different than those of the existing network based servers. First, the real-time nature of multimedia data requires QOS guarantees in the form guaranteed periodic retrieval and transmission of data. Such guarantees can be provided by performing admission control and resource reservation. The layout of data on storage devices used at the server and the algorithms used to schedule retrievals from the server must be designed to ensure that such resource reservation is valid throughout the duration of an admitted client session. Clearly, *admission control, data layout and storage access scheduling* are important design problems.

A MOD server that stores a large number of long multimedia documents must manage tera/peta bytes of storage systems that will be constructed in hierarchical fashion using storage devices of various types. The server must support potentially hundreds or thousands of concurrent clients independently. This, directly translates to a storage and network throughput requirement in excess of tens or hundreds of gigabits. Also, it is important that the server architecture be scalable in terms of throughput and number of clients.

The large system scale and expanse of the geographical distribution of the future multimedia computing infrastructure poses scalability problems. Specifically, such an infrastructure has to support millions of concurrent users that communicate with each other across the continents and access information streams originating at locations on the other side of the globe. Clearly, networking and storage systems that attempt meet such requirements must be hierarchical and distributed. In general, in such an hierarchy, guaranteed high performance accesses to data are inexpensive and easy to support only over short distances. Therefore, to alleviate cost and performance penalty incurred in very remote data accesses, information caching must be explored at all possible levels in the system.

In this chapter, we present design of scalable storage server architectures and their hierarchies. First, we present essential background material required to understand rest of the chapter (Section 2). We then describe various storage server architectures. Specifically, we cover three different architectures: one, the distributed storage server architecture called of the Massively-parallel And Real-time Storage (MARS) server currently under development at Washington University [7] and the alternate architectures based on MIMD and SIMD multiprocessor machines (Section 3). The data layouts for large scale servers, specifically layouts in clustered MOD servers and layouts on storage devices such as disks are presented (Section 4). The admission control problem and possible solutions to it in the context of storage server design are also covered. (Section 5). In the end, we describe the problem of *information* or *program caching* in a storage server hierarchy and present near-optimal algorithm for computing caching schedule for interactive movies-on-demand application.

2 Essential Background

We will first present some essential background to help the reader understand the research issues in design of large scale MOD systems and services.

2.1 Application Scenarios

The two common emerging scenarios in which the future applications will be run are illustrated in Figures 2 and 3. In the first scenario commonly

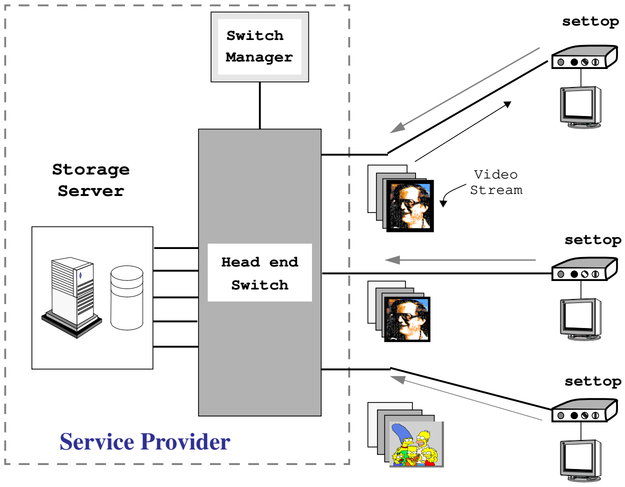

Figure 2. A typical Video-on-demand (MOD) application

termed as **Video-on-demand**, as conceived by telephone companies, cable operators, and content providers like Time-Warner, the client accesses the multimedia data using a set-top box which replaces the VCR and a remote control device used to communicate control commands. This infrastructure primarily suites on-demand entertainment program, news, home shopping, travel information, and network based game applications.

In the second scenario which is popularly known as **World Wide Web** millions of computers interconnected into a tangled web by the ever increasing *internet*, act as information servers and/or clients. The client is typically a browser program such as *Netscape, Xmosaic* with a point-and-click interface. These clients access multimedia information in the form of text, images, audio, video from the web servers in the internet via a special protocol called *HyperText Transfer Protocol* (HTTP) that employs the TCP/IP protocol stack for communication.

The client end-systems in these two scenarios differ in capabilities such as amount of local buffers, compute power, and availability of specialized media processing. For example, a hand-held PDA, or a set-top box will have small local buffers and minimal compute power compared to workstations or PCs. A storage server must provide interactive access to real-time and non-real-time data stored in multitudes of formats to

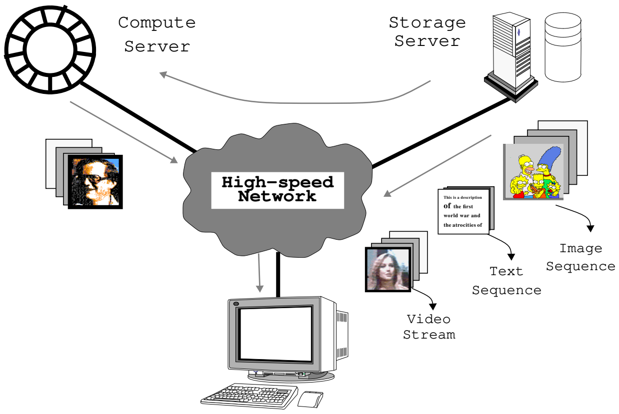

Figure 3. A World-Wide-Web based Multimedia-on-demand (MOD) application

such heterogeneous clients. To accomplish this, a clear definition of the service model that will be used by the server is essential.

2.2 Properties of Request Arrival Process

Figure 4 shows the model of a MOD server in a MOD retrieval environment. Each MOD server has N multimedia files, each of which has two attributes associated with it. The first attribute, duration D_k for the k^{th} multimedia file, is the length of the time that would be required to play the file at standard playout rate, whereas the demand Q_k for the k^{th} multimedia file measures the number of simultaneous connections the server can support for the file. The clients can access the multimedia file(s) by sending requests to the server. The server performs an admission control, which, based on the existing load and the resource usage admits or rejects the client request. The request arrival process at the MOD server may exhibit spatial and temporal locality as described below.

- **Spatial Locality:** The spatial locality property signifies that some of the data items are likely to accessed more frequently than the others. If the multimedia documents, indexed 1 to N_{max}, are ordered in decreasing order of demand (i.e if $i < j$, then $Q_i > Q_j$), the probability that the incoming request is for document p is higher than the probability that it is for document q, if $p < q$. For example, in case of an on-demand movie application, a large fraction of requests received are likely to be for certain popular and recently re-

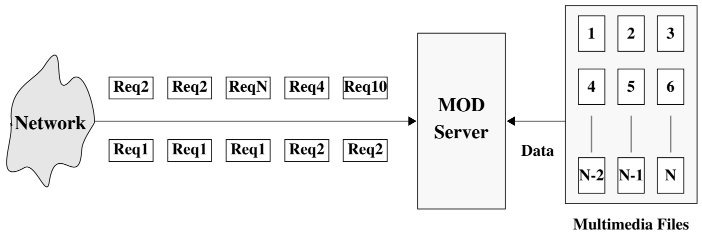

Figure 4. MOD server request-response model

leased movies and relatively less for old movies. Empirically, the arrival process at a movie rental shop can be modelled using Zipf's law [14]. Similarly, in the case of a teleshopping mall, some stores would be more popular than the others and would therefore receive more requests. In case of a digital lecture archive, lectures on difficult topics and those given by laconic faculty members are likely to be accessed more often. However, some of the applications may exhibit less spatial locality than the others, for example, all contemporary publications in a on-demand digital library or radiological images in a patient information database are likely to be uniformly accessed.

- **Temporal Locality:** Depending on the application, the request arrival process at a MOD server will be temporally clustered to various extent. For example, in case of on-demand movies, request rate for a set of popular movies would be higher during the evening time period of a weekend than on the weekdays and more requests would be received during the 6 to 9 pm than any other period of the day. On the other hand, in case of a digital library or a radiological database, the request arrivals would be Poisson or slightly bursty.

The spatial locality is exploited in placement of multimedia data in a storage hierarchy consisting of different storage types. Clearly, frequently requested data should be assigned to faster storage devices that are typically more expensive. The temporal locality property is important in deciding the service and network model for the server.

2.3 On-demand Service Types

An on-demand service type decides the method of data and control command exchange between the client and the storage server. It, thus, determines the extent of interactivity and personalization available to the client. The multimedia-on-demand services can be classified into following three types:

Pay-Per-View (PPV) **service:**
Analogous to the existing PPV channels on cable networks, the time of access for a multimedia document in this service is decided by the server rather than the client. The MOD server multicasts the multimedia program at fixed times and any clients interested in receiving the program tune in to the server at those fixed times. Clearly, a client of such a service cannot control the stream playout and also has minimal freedom as to when the multimedia stream will be available to him. Implementing such a server hardly presents any technological challenges.

Near Video-on-demand (NVD)**:**
This service is motivated by the observation that for certain applications such as on-demand movies, request arrivals are likely to exhibit significant temporal and spatial locality. The Near Video-on-Demand service, sometimes called **Shared Viewing with Constraints** (SVC) exploits these properties of request arrivals. In this service, the client accesses the multimedia documents at any time it desires by sending request to the MOD server. The server processes and accepts such requests in groups to exploit the clustering properties of the arrival process. A new client request may face a variable admission latency, after which the client becomes a part of a multicast group, all members of which are connected to the server by a single multicast connection.

Consider the example of on-demand movies with a typical movie of two hour duration. As shown in Figure 5, the server may decide to cluster requests received in a 10 minute duration and service all such clients as a single multicast group. Thus, in the worst case a client will face an admission latency of 10 minutes. Also, the maximum number of retrievals required at any given time will be 12, unlike only one in PPV service. One advantage of this scheme is that the movie can be divided into twelve independent units and stored on separate storage nodes, each of which can service a multicast group at a given instant as the movie playout progresses. For example, for clients belonging to cluster 1, the first 10 minutes of the movie are played out from one storage node and at the end of this duration, the cluster is switched to another storage node for next 10 minutes of the movie. Thus, under busy conditions there can be as many as 12 multicast groups, each being serviced by one storage node and switched to an appropriate storage node at the end of ten minute duration. However, any kind of interactivity is clumsy to support. First of all, excessive interac-

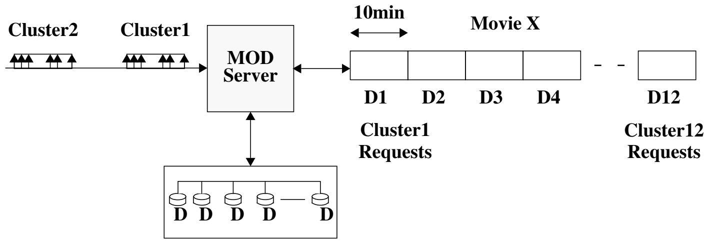

Figure 5. Shared Viewing with Constraints (SVC)

tivity such as rewind and fast forward requires frequent switching of clients to different multicast groups. Also, given that all the clients that are grouped and serviced together have the same 'view' of the movie, any fast forward/rewind by any member will alter this 'view' for all members of the group which is undesirable. If fast forward and rewind from each member are to be treated independently, multicast can't be used to realize bandwidth any bandwidth saving.

The NVD represents incremental improvement over the PPV service, as it allows user to access the multimedia documents at arbitrary instances, unlike the fixed instances in PPV service and for this reason it is sometimes called near-video. However, supporting independent interactive operations is difficult in this service. Thus, the NVD service is unsuitable for applications where request clustering is difficult to achieve and interactivity is important.

True-Multimedia-On-Demand (T-MOD):
Figure 6 illustrates this service model, in which the each accesses request from same or different clients is treated independently at the server. For example, in Figure 6, even if two requests received at time t_1 and t_2 are close in time and for the same video sequence `X`, if admitted, server will perform separate retrieval and transmissions for these requests. Due to this independence of requests, T-MOD service model is sometimes referred to as Dedicated Viewing (DV). Consider an example of a neighborhood on-demand movie server connected to a head-end switch which connects to maximum 1000 users (households), all 1000 users might watch the same movie independently using DV service and would therefore require 1000 independent retrievals and deliveries. The primary advantage of this service is that it is a natural paradigm for personalized, interactive multimedia delivery, as streams received by multiple clients are

independent and do not have any temporal relationship between them. Clearly, it is the appropriate service model for the future distributed multimedia on-demand applications. However, T-MOD service has serious implications on the throughput. Consider an example of a storage server serving HDTV quality movies to 200 customers. Each HDTV stream requires 20 Mbps on the average. So in the worst case the aggregate network and storage throughput requirement is in excess of 4 Gbps.

2.4 Network Model

If we assume the true-on-demand service model for all future applications, the aggregate throughput requirements for the global networking infrastructure can be overwhelming. Consider an example scenario in which each active user is provided with an independent 20 Mbps HDTV connection. In continental US alone, during the prime-time, approximately 150 million users in 77 million households will require an aggregate network bandwidth of 1.54 terabits/sec [2, 28]. Ideally, if bandwidth and storage were free, a centralized super server that stores all possible programs can serve all the clients. In an alternate scenario, if the storage costs were negligible, one can place as many programs close to the client premises as possible. Clearly, both these scenarios are unrealistic.

Though storage technology improvements have lead to remarkable increases in the storage capacities of devices and show every sign of continuing to do so [39], the storage costs will still be the major component of the total cost of a MOD system. In fact growing concern is being voiced that the enormous network and storage costs in design of large scale storage systems and network infrastructure may far outweigh the amount of revenue it may generate. A survey [25] shows that while MOD has a high appeal among the 44% of the respondents who were willing to pay for it, only a miniscule 14% were willing to pay *more* than the existing cable rates for it. Nearly two thirds of the respondents owned a personal computer, with nearly half of them equipped with a modem; hence, we can infer that they were technology savvy and very much knew the difference multimedia-on-demand would make in their lives. Given this, minimizing network and storage costs is very important to make future MOD services affordable.

The three main ways to meet network and storage costs in scalable fashion are: *hierarchical solutions, caching* and *sharing*. The consensus among the research community is that the network infrastructure that will support future interactive MOD services will be hierarchical in nature and will look somewhat like Fig-

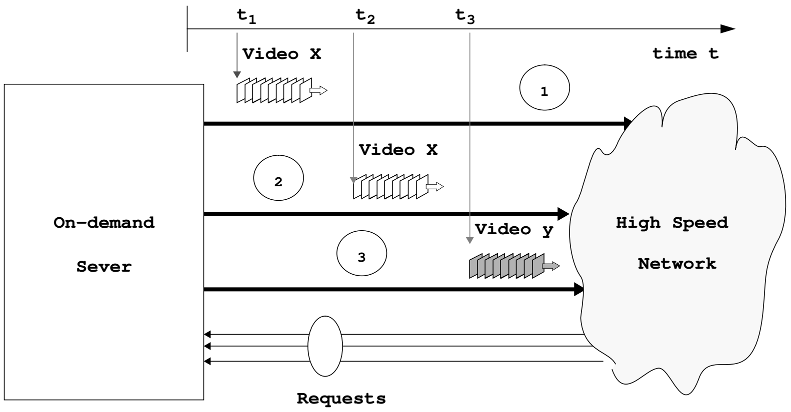

Figure 6. MOD server for a Dedicated Viewing service (DV)

ure 7. As shown there, it will consist of a international backbone network connecting several national backbone networks which in turn interconnect numerous regional networks. Each regional network may interconnect several access networks through which broadband network access is provided to the homes. The technology of choice for backbone network will be ATM over multiple high speed SONET links. In the short term, the access network, typically provided by a local phone exchange or a cable operator will be based on one of the several technology proposals such as *Subcarrier Modulated Fiber-Coax Bus* (SMFCB)[1] or *Baseband Modulated Fiber Bus* (BMFB) [2] [33]. This access network will connect the user equipment such as a set-top box to the rest of the network via the **headends**. In the long term, the headends may be replaced by an ATM switch of reasonable size such as $1K \times 1K$. A local MOD server will be connected to each headend. The higher levels in the hierarchy will have increasingly large scale storage servers.

In fact, as shown in Figure 8, the system will evolve from a synergy between various enterprises: *Storage Providers* that manage information storage at multimedia servers (a role akin to that of video rental stores and libraries in today's context), *Network Providers*, that are responsible for media transport over integrated networks (a role akin to that of cable and telephone and cable companies of today), and *Content Providers* such as entertainment houses, new producers etc., that offer a multitude of services to subscriber homes using multime-

[1] Also called Hybrid Fiber Coax (HFC)
[2] Also, called Switched Digital Video (SDV)

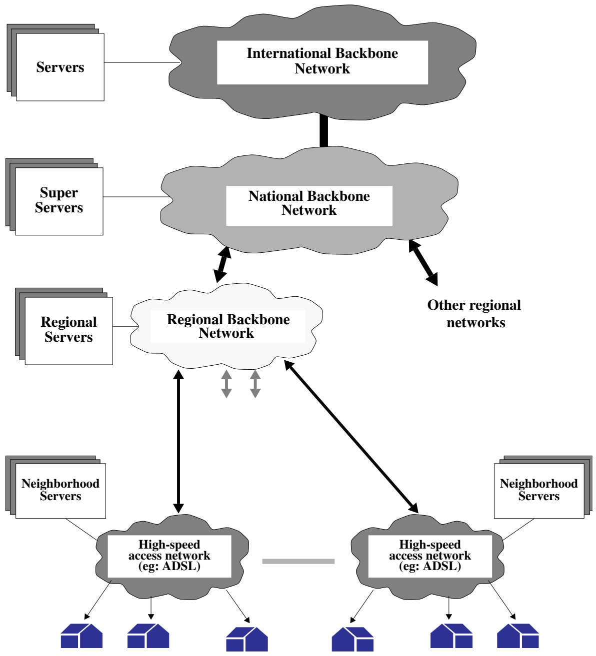

Figure 7. Hierarchical Network Model

dia servers and broadband networks.

The storage costs and network traffic in this hierarchy can be reduced by exploiting the spatial and temporal locality property of multimedia document accesses. Studies [1] have shown that the patterns of viewership are very much dependent on the time of the day. Akin to the telephone network, there are peak hours during which the traffic is maximum and the majority of the titles demanded is a small subset of the most recent set of hit movies. If the profile of user requests for various multimedia documents is available a-priori, only those documents that are likely to be requested with high probability should be stored on the neighborhood servers and less frequently requested documents should be stored on servers in the higher levels in the hierarchy. This technique, commonly called **Info/program caching** [30, 28], also minimizes accesses to the servers in the higher levels of the hierarchy and reduces upstream bandwidth requirements in peak hours.

2.5 Performance Metrics for a MOD server

In order to compare and classify storage server architecture, we define the following set of performance metrics.

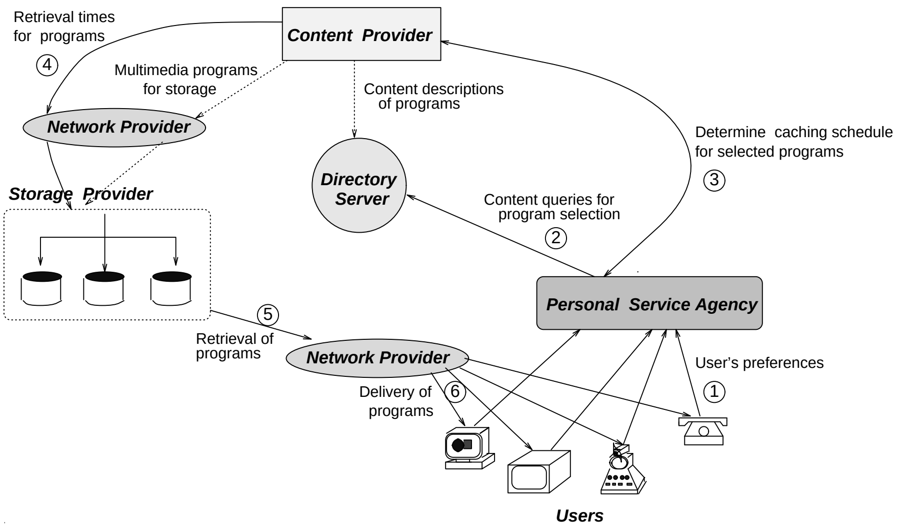

Figure 8. Service Architecture

- **Concurrency:** The server may have to support thousands of concurrent clients, each with a number of active media streams and each independently accessing the same or different data. The concurrency metric captures this requirement and is defined as the maximum number of clients that can independently access a multimedia document in any playout control state. Thus, in case of a on-demand movie server that can support maximum 1000 clients, potentially all 1000 clients should be able to access the same copy of the movie independently at any time. Higher concurrency minimizes need for document replication and storage costs.

- **Access latency and Operation Latency:** The access latency metric is defined as the amount of time a client has to wait after sending a request to the MOD server to receive the requested multimedia document. It mostly depends on the service model and the round-trip network latency is a minimal part of it. The access latency should be typically less than a few seconds. The operation latency metric is defined as the time required to effect a interactive operation such as *ff, rw, pause, stop, random access* etc. Highly interactive MOD applications will typically require an operation latency of less than a second. It is desirable that the playout and operation latency be independent of the load on the server. Clearly, these two metrics characterize how effectively the server provides the QOS guarantees in the form of bounded delay.

- **Storage capacity and storage, network throughput per dollar:** Given the storage intensive nature of multimedia data, the storage requirements for a large scale server can be in excess of tens of terabytes. For example, a lecture archive in a small sized academic department offering 50 classes in a semester, each with 50 1hr lectures archived as MPEG video streams would require approximately 5.8 TBs. The independent retrievals and transmissions for each active client in the T-MOD service model demands large amounts of network and storage system throughput. For example, a movie server that supports one thousand users with HDTV quality movies, will require network and storage bandwidth in excess of 20 Gbps. This metric defines the cost-effectiveness of the storage server in meeting the requirement of large storage capacity and the large throughput. It must be small for the MOD services to be affordable.
- **Scalability:** It is desirable that a server architecture that supports say 100 clients and an aggregate throughput of 500 Mbps, should be easily extendable to 1000 clients and a ten fold increase in network/storage throughput without significant modifications. In other words the server architecture should scale with number of clients. In addition, the server must support heterogeneous clients, multimedia data and access networks.
- **Extendibility:** The server must be able to support multiple application scenario and must be extendable to support different service models.

2.6 Taxonomy and Hierarchy of Storage Servers

Using the performance metrics described above (Section 2.5), the storage servers can be classified as shown in Table 1. Since video is the most demanding of all the multimedia stream, in the classification the term client is synonymous with a video connection. Though the average rate of a video connection depends on various factors such as the size of the image, resolution and the type of compression used, in the following description, we assume a standard NTSC quality video connection to be a 5 Mbps MPEG compressed stream. For HDTV quality compressed video streams the storage and network bandwidth will be even higher. It must be noted that the storage capacity and storage bandwidth (throughput) are two attributes of a storage system that are independent of each other. Given that the advances in storage technology have almost always improved the storage capacity more than the storage throughput, extracting higher throughput from a storage system is a more challenging task than providing a large storage capacity. Also, as a general case, the storage in all classes of servers will be hierarchical consisting of magnetic Direct Access Storage Disks (DASDs), magneto-optic devices like optical disks, magnetic and optical tapes. Therefore, the classification below does not include any reference to the storage capacity.

Note that the throughput requirements increase dramatically for large storage servers. With the advances in fast packet switching, providing increased amount of network bandwidth is feasible. However, due to modest rate of improvements in the speed of storage devices such as magnetic or optical disks, same is not true of the storage throughput. Also, note that the access and operation latency requirements remain the same irrespective of the size of the server. Satisfying these throughput and latency requirements together can be very difficult.

3 Multimedia Storage Server Architectures

We will first motivate the need for new architectures for high performance large scale storage servers and then present a new architecture called *Massively-parallel and Real-time Storage* (MARS) architecture. We will then discuss alternate storage server architectures and their advantages and disadvantages.

3.1 Large Scale Servers: What is the problem?

The network based storage servers in their present form suffer from serious storage and network I/O bottlenecks. The storage I/O bottleneck results primarily due to performance gap between the storage and processor-memory systems. It manifests itself by restricting the rate (the requests per second and the throughput in Mbps) at which storage devices can handle read/write operations to two orders of magnitude smaller than the maximum rate at which an application can issue requests. This performance gap will continue to worsen as the processor, memory and network speeds keep improving faster than that of on-line secondary storage, which is limited largely by the transfer rate of direct access magnetic disks. In spite of the promise of all-optical technologies such as holographic storage, inexpensive availability of high throughput

Table 1. Taxonomy of Storage Servers

Scale	Number of Clients	Concurrency	Access Latency	Throughput
Small	25	25	$\approx< 1sec$	155 Mbps (OC-3)
Medium	100	100	$\approx< 1sec$	622 Mbps (OC-12)
Large	1000	1000	$\approx< 1sec$	0.5 Gbps
Super	$10,000$	$10,000$	$\approx< 1sec$	$50Gbps$

secondary storage in the near future is unlikely. On the other hand, the rate of improvement in the storage density of magnetic disks [23, 39] indicates that magnetic storage will be cost-effective in meeting the storage capacity requirements of future multimedia applications. However, as reported in [12], the rate of improvements in seek time (8% a year), rotational latency (practically no improvements) and transfer speeds (22% a year) will be very modest. This suggests that a large capacity disk storage used naively will not deliver the throughput required by future applications. To make matters worse, the traditional operating systems (such as UNIX) used in existing servers suffer from several drawbacks, such as random placement of data, mixing of the meta-data and data on the storage devices, excessive copying of data between the storage devices and the network interface, and the lack of real-time scheduling support. This mismatch between the requirements of multimedia and the traditional operating system support for secondary storage devices aggravates the storage I/O bottleneck even further. Clearly, the high performance storage I/O will be critical to successful deployment of multimedia applications.

The network I/O bottleneck can be defined as the phenomenon that restricts the application level throughput for network oriented tasks to a small fraction (20 to 30 megabits/sec (Mbps)) of the gigabits/sec (Gbps) network bandwidth. It results from a lack of integration of operating systems, network protocol processing, host architecture and network interface. However, several recent ideas, such as zero-copy or minimal copy host-network interfaces and efficient protocol processing have alleviated the network I/O bottleneck significantly. However, existing servers still do not support direct movement of data between the storage and network devices with minimal copying and guaranteed delay.

Also, existing network servers are exclusively bus based single processor or multiprocessor workstations. Since, the bus based interconnects do not scale well, large scale servers can not be built out of such machines. Also, the workload these servers support typically consists of online transaction processing, or bulk general purpose computing which do not need any soft-real-time guarantees necessary for future multimedia applications. In short, requirements of future MOD storage servers are radically different, and therefore designing such servers is a challenging task that requires significant architectural innovation.

The architectures of large scale servers that attempt to meet these challenges can be classified into three categories:

1. **Distributed Storage based Clustered Multimedia Servers**
2. **Shared Memory** MIMD **multiprocessor machines**
3. **Massively parallel** SIMD **machines machines**

We will now discuss each of these in greater detail.

3.2 Massively-parallel And Real-time Storage (MARS) Architecture

The *Massively-parallel And Real-time Storage* (MARS) is a representative example of the first of the three categories namely the *Distributed Storage* based architecture. It is an on-going project at the Washington University in St. Louis funded by the NSF National Challenges Award (NCA) grant, aimed at prototype development and deployment of a scalable multi-gigabit throughput media servers in a metropolitan area network.

The MARS architecture, shown in Figure 9, consists of a set of independent storage nodes that are connected together by a fast packet based interconnect and managed by a central manager. The high speed interconnect which can be in the form of a high-speed packet switched bus, a ring or even a general purpose multicast switch, directly interfaces the server to the external network. For example, an ATM interconnect would directly interface the server to an external ATM network. The interconnects such as ATM switches are already available off-the-shelf and are quite inexpensive. In foreseeable future, the interconnect speeds will increase and costs continue to decrease as they become commodity items. Thus, the high speed interconnect employed in MARS will be scalable and inexpensive.

Figure 10 shows a prototype MARS server architecture that uses a 2.4 Gbps ATM cell-switched interconnect based on a ASIC called the *ATM Port Interconnect Controller*. The three main components of this architecture: the **central manager**, the ATM interconnect, and the storage nodes are described in

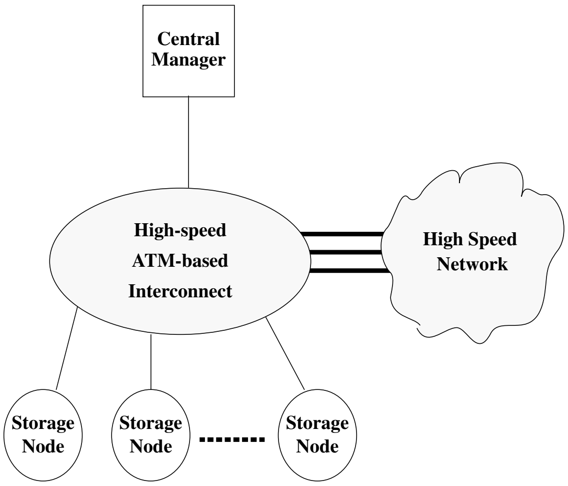

Figure 9. Massively-parallel and Real-time Storage (MARS) system

greater detail below.

3.2.1 Central Manager

The central manager is responsible for data distribution, data retrievals, admission and access control, and billing functions. The MARS is a stateful server and stores the state information for every active stream. It follows a connection oriented approach with resource reservations to provide QOS guarantees required by the media streams. Before a client can access a multimedia stream, the MARS server checks if it can reserve resources required to guarantee QOS for the new connection without affecting any of the existing active connections, and accordingly accepts or rejects the connection. The central manager performs this admission control in consultation with storage nodes. Typically, the central manager receives the connection requests from the remote clients. It uses an admission control procedure, that runs entirely locally or in consultation with the node level admission procedure, to admit or reject the new request. Once admitted, the MARS server reserves resources such as the network bandwidth, storage bandwidth, compute support, and I/O scheduling at each node to provide statistical or deterministic guarantees for the stream retrieval.

For every document, the central manager also decides how to distribute the data over the storage nodes and manages the associated metadata information. The metadata information asso-

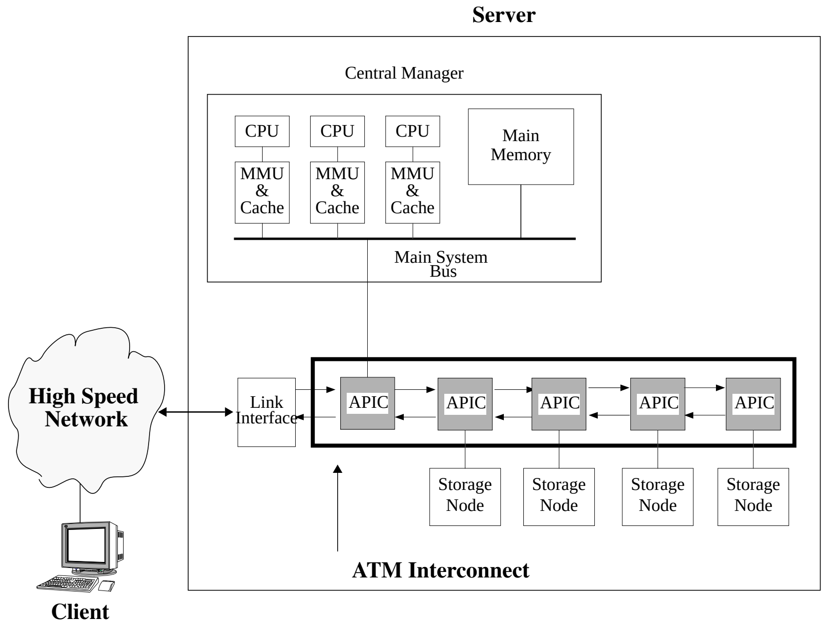

Figure 10. A prototype implementation of a MARS server

ciated with the data is also distributed among various storage nodes and the central manager. Such data and meta-data distribution (striping) depends on various factors such as, length of the document, demand in terms of the number of concurrent clients, degree of interactive behavior, and design of the data layout. Typically, the storage manager decides this distribution at the time the document is stored on the server.

For every active connection, the central manager 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 to and from the storage devices and/or the network and does not participate in actual data movement. This ensures a high bandwidth path between the storage and the network.

3.2.2 APIC Based Interconnect

The packet switched interconnect in our prototype MARS architecture is constructed out of an ASIC called APIC (ATM Port Interconnect Controller), that is currently being developed as a part of an ARPA sponsored gigabit local area ATM testbed. The APIC chip is the basic building block for a high bandwidth *networked*-I/O *subsys-*

tem, that provides a direct interface to the network for the host (workstations as well as servers) and a variety of I/O devices.

In its simplest form, the APIC behaves like a 3×3 switch, two of whose ports can be treated as ATM ports and the remaining one as a non-ATM port. Using one of the ATM ports and a line interface, APIC can be directly interfaced to the input port of an ATM switch. As shown in Figure 10, using the ATM ports, multiple APIC chips can be connected in a bidirectional daisy chain. Since each port is designed to operate at full 1.2 Gbps (SONET) rate, the aggregate data rate on the interconnect is ≈ 2.4 Gbps. The non-ATM port of an APIC can be looked upon as a read/write port to a data source, such as a memory or an I/O device. The APIC performs incremental AAL5 segmentation and reassembly of ATM cells through this port.

Each APIC manages a set of connections by maintaining certain state information for each active connection in a Virtual Circuit Translation Table (VCXT). The APIC also performs a simple single parameter (such as average bandwidth) based rate control or source pacing for each connection, which is important in scheduling multimedia streams from storage node to the network. The APIC also provides several DMA mechanisms, such as *Simple DMA, Pool DMA, Protected DMA* aimed at allowing efficient transfer of data between various devices and memory with minimal copying. The discussion of these APIC internals is beyond the scope of this text, but can be found in [17, 18].

3.2.3 Storage Node

The capability of MARS architecture to provide massive parallelism and support large number of concurrent accesses strongly depends on the functionality provided at the storage nodes.

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 storage system collectively formed by all storage nodes may be homogeneous or hierarchical. If all storage nodes provide identical storage, such as disk arrays, the system will be homogeneous. On the other hand, a hierarchical storage can be constructed by assigning storage of different types to subsets of storage nodes. A client can be served the required data, either directly from the particular storage node (level) or by a staging mechanism, in which data is first moved to a node with faster storage (higher level) and then served from that device. For example, the nodes that use optical storage and robotically controlled tapes can be considered as off-line or near-line tertiary storage. When a client attempts to access the data stored on these devices, it is first cached on the magnetic disks at the other nodes and then served at full stream rate. Thus, the collective storage in the system can exceed a few tens of terabytes and still, allow a large number of concurrent clients to access documents at the standard stream rate.

In addition to providing a storage facility, each storage node supports one or more of the resource management functions outlined below.

- **File system support:** Each storage node performs typical file system functions such as data and metadata management, metadata and data cache management, and data buffer management. In addition, it may support advanced database functions for efficient browsing and content based searching of multimedia information.

- **Admission Control:** Each storage node keeps track of usage of local resources such as storage and interconnect bandwidth and performs local admission control.

- **Scheduling support:** Each node, completely manages real-time scheduling of local data read and write functions for each active stream.

- **Compute support:** A large amount of computing power can be provided by using sophisticated media processors such as the MVP [24] at each storage node. Such processors will be embedded on the path between the storage and the network, and can be used to perform media processing functions such as transcoding, speech recognition, image processing, and character recognition required by future multimedia applications.

The main objective in the design of the storage node is to provide a high bandwidth, efficient data path between the external network and the storage devices. The architecture of the storage node described here assumes the storage in the form of a RAID, however it can be extended easily to accommodate other forms of storage such as a set of independent high capacity optical and magnetic disks. A RAID storage node is illustrated in Figure 11. Each storage node is constructed out of a high performance processor subsystem such as a PENTIUM PC. The disk array at the node is constructed out of a set of *Small Computer System Interconnect* (SCSI) strings, with a fixed number of disks per string or a SCSI compatible disk array per string. In case of majority of commercial RAID disk arrays provide a on-board processor for performing disk array control functions.

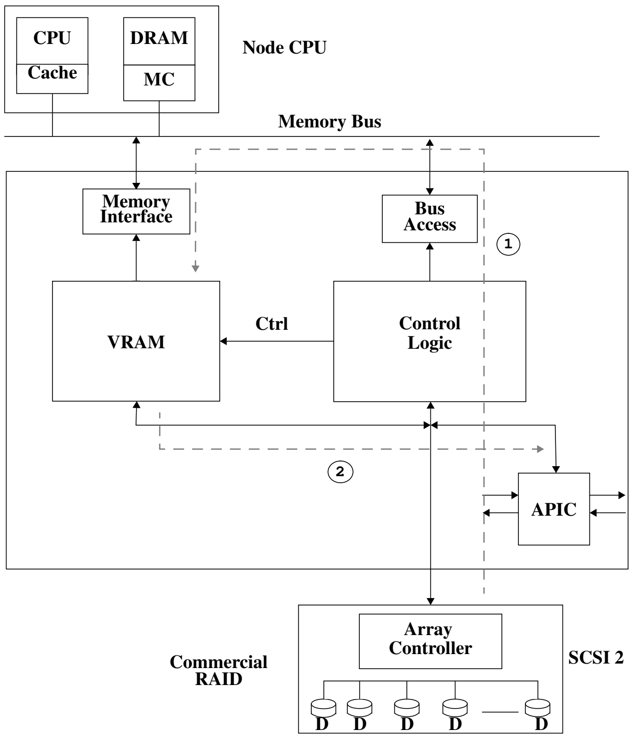

Figure 11. Example Storage node architecture

The same processor can serve as a local CPU at a storage node. The multiple SCSI strings interface to the APIC through a dual ported memory, such as a video RAM (VRAM).

In order to understand the requirements that must be satisfied in the design of such a storage node, we examine the data and control path within the storage node. Figure 11 illustrates the data path between the VRAM, the disk array and the APIC. As shown there, the disk array controller writes the data prefetched from the disk array into the preallocated buffers in the VRAM using the direct access port. The APIC reads this prefetched data using the serial port of the VRAM. Both these accesses are arbitrated by the control logic. The node CPU issues control commands required for controlling prefetch and transmission activities by writing the memory mapped registers of the APIC and disk controller. These accesses through the control logic constitute the control path. Clearly, the node CPU is not involved in the high speed data movement between the storage devices and external network.

The data is typically prefetched and stored in a buffer before it is transmitted on to the network. The commercial RAIDs commonly provide up to 40 MBps or 320 Mbps throughput. The APIC is being designed for 1.2 Gbps throughput per port. Thus, in the worst case, the total throughput in and out of the buffering system must be of the order of 2 Gbps. However, memory systems constructed out of commercial DRAMS can provide a peak bandwidth of 500 Mbps. Given this, avoiding data copying and use of high bandwidth memory alternatives such as VRAM provides significant performance advantage.

A rough estimate of the size of such a memory system is also in order. If the average BW requirement of an MPEG stream is assumed to be 5 Mbps, the interconnect BW of 1.2 Gbps can accommodate roughly 240 clients. If the storage node prefetched 1 GOP of typically 9 frames, each connection prefetch buffer is roughly 0.2 MBs, and over all 240 connections it is about 48 MBs. Clearly, increasing this prefetch granularity increases the required buffer size. Also, as much as twice the size of prefetch buffers may be required to ensure smooth playout and playout control operations. In short, the data path must provide high capacity memory of the order of a few hundred megabytes and must sustain throughput of the order 2 Gbps.

The control path is responsible for admission control as well scheduling of prefetches from disk arrays and transmissions by the APIC. These tasks involve allocating/deallocating the data buffers, and scheduling related data structures, periodically updating scheduling information, and retrieving and managing the metadata. Such operations will typically be implemented in software as a part of a node level operating system and will require periodic timely access to the memory, storage and network system to guarantee overall correctness of scheduling operations. Clearly, in addition to the aforementioned data path throughput, this added load must be sustained. As shown in Figure 12, in a read-only environment, the local node OS asynchronously writes the data in the VRAM and the APIC consumes it to transfer it to the network. The VRAM is shared by the following buffers:

- **Buffers for periodic streams:** A set of circular buffers serve as the prefetching buffers for the data from periodic streams retrieved off the disk array. These buffers can be statically reserved on a per connection basis at the connection setup time or dynamically allocated during every cycle of the scheduling scheme described earlier. The size of allocated buffers depends on the media type as well as the playback rate. For example, the buffer requirement of a video stream is much larger than that of an audio stream. Also, a video stream played at 15 frames/sec needs much smaller buffers than the one played at the standard 30 frames/sec.

The data compression techniques have implications for the buffer allocation policy as well as the buffer requirements. For example, the buffer required for a JPEG encoded stream is significantly larger than its MPEG encoded counterpart. Another characteristics of the compressed streams is that the size of successive frames in the media streams varies significantly, causing a frame-to-frame variability in the buffer requirements. For example, empirical evidence has shown a factor of four to five variability in the data content of single frame of MPEG compressed video streams. Thus, the static allocation of buffers, though simple, requires more memory because the size of the buffer allocated at the connection set up time has to be the maximum expected size of a frame.

- **Buffers for non-real time tasks:** A set of buffers are used for read/writes of non real-time streams, such as still images, text, and progressive image transmissions.
- **Request buffers:** The storage node operating system sends control commands for read and write schedule management in a request queue in the VRAM (Figure 11). The disk array controller reads these commands to initiate appropriate SCSI transactions to retrieve data from the storage devices. The request queue is partitioned based on the connection identifier or the VCIs.
- **Positional metadata buffers:** The positional metadata is the information about the location of the data on the storage devices. This data is provided by the central manager and used by the array controller to read (write) disks in the array. Like the request queue, the metadata queue is also partitioned based on VCIs.

These buffers are allocated and managed by the the local operating system at the storage node. In the prototype architecture, each node runs a NETBSD UNIX operating system enhanced to handle multimedia data. Specifically, the filesystem and buffer management func-

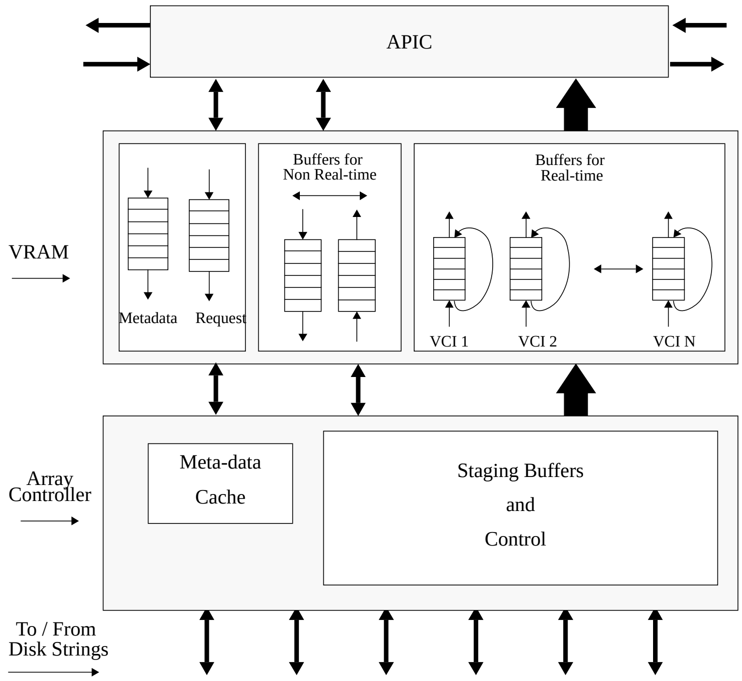

Figure 12. Details of node-APIC interface

tion in this OS have been modified to allow periodic retrieval of data and minimize buffer copying [15, 16]. Also, a new process scheduling technique called *Real Time Upcalls* (RTU) is employed to support periodic event processing required to provide QOS guarantees [20, 19, 21].

3.2.4 A Scalable Extension

The Cluster Based Storage (CBS) architecture, which is a scalable extension of the prototype implementation discussed earlier, is illustrated in Figure 13. It consists of a set of independent clusters interconnected through a fast multicast ATM switch, such as [37, 38]. The ATM switch interfaces the server to the external network. Each of the clusters resembles our prototype implementation architecture. The APIC at the end of the daisy chain in a cluster transparently interfaces to one port of an ATM switch. Each cluster has a local cluster manager. The central manager in this architecture, manages the switch, performs the network signaling operations, and cooperates with the cluster level storage managers to perform admission control.

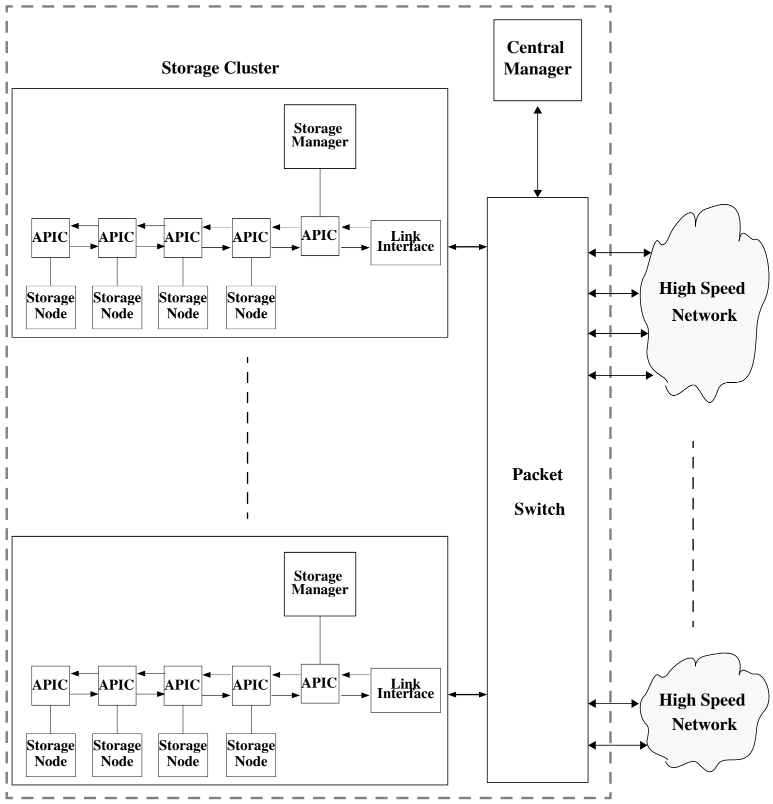

Figure 13. Cluster Based Storage (CBS) architecture for MARS

Two types of information flows between the CBS server and the external network: one type is the control information such as the client requests for multimedia streams, stream manipulation commands such as fast forward, reverse, pause etc., server responses, and the network signaling information. The other type of information is the actual multimedia data for all active connections. Depending upon the implementation, the CBS architecture may reserve certain switch ports for control information and the rest of the ports for data traffic.

The scalability of this architecture accrues from increasing the number of ports and the bandwidth per port of the ATM switch, a trend supported by advances in the field of ATM switching. Note that the availability of multiple clusters provides a greater number of options for data distribution. For example, popular documents can be replicated in multiple clusters and incoming requests for them can be assigned to one of these clusters. If the number of documents is very large, a subset of them can be assigned and served from each cluster. Also, the documents of very long duration can be conveniently split into smaller parts, each of which can be stored independently on a separate cluster. Note that the packet switch in this architecture allows a connection from any input port to be switched and re-mapped to a different network level connection at any output port. This allows multimedia data to be striped not only over the storage nodes in a cluster, but over multiple clusters as well. The increased parallelism that results from greater degree of data distribution in turn, can support larger number of concurrent accesses to the same document. In short, the CBS architecture is truly scalable in terms of number of clients, storage capacity and storage and network throughput.

3.3 Shared Memory MIMD Multiprocessor Based Storage Servers

Use of shared memory MIMD multiprocessor machines as MOD represents another architectural approach to design of large scale MOD servers. The proposed MOD servers from vendors like Silicon Graphics, and Whittaker Communications, Inc. are examples of this approach. Here we will briefly describe the Silicon Graphics server architecture shown in Figure 14 [5].

The architecture consists of several R4400 CPU boards, I/O subsystems in the form of HIO boards, and interleaved memory boards interconnected by a 1.2 GBps *PowerPath-2* bus with separate 256 bit data and 64-bit address paths. Each HIO board uses a internal proprietary HIO I/O bus rated at 320 MBps throughput. At present, maximum four such boards are allowed in the high-end configuration, theoretically providing a 1.280 GBps of I/O throughput.

Each HIO board, shown in Figure 15 contains a intelligent ethernet controller, serial RS-232/422 and parallel port controllers, VME64 controller and a graphics connection. The storage I/O capability is provided in the form of two SCSI-2 controllers, each capable of 20 MBps rated throughput. Each I/O board has slots for two HIO modules, each of which can be populated with upto three SCSI-2 modules. Thus, each HIO board can have upto eight SCSI-2 controllers allowing maximum of 160 MBps storage throughput.

The ID, IA chips shown in Figure 15 act as bus adapters and provide interface to POWERPATH bus. They perform vir-

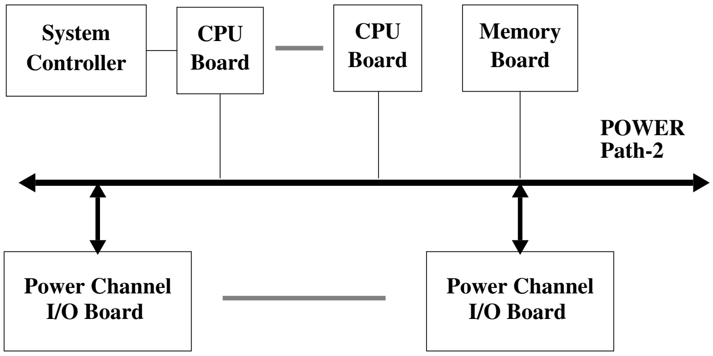

Figure 14. Silicon Graphics symmetric shared memory MIMD multiprocessor machine

tual address mapping for DMA operations - maintain cache coherency between system bus and I/O systems.

The SGI server runs a UNIX like operating system called IRIX with file system extensions. Specifically, it supports a data layout method called **logical volume striping** that treats multiple independent storage devices such as disk arrays to be treated as a single logical device or volume. Another level of striping called **Application Level Striping** allows an application to decide it's own striping scheme for distributing data over multiple logical volume. Both these extensions are aimed at increasing number of concurrent accesses to the same document and parallelism in data accesses. Detailed performance measurements on a fully configured 20 processor system with 512 MBs memory and 32 RAID3 disk arrays have been reported in [22].

3.4 Massively Parallel SIMD Multiprocessor based servers

Another approach to developing large scale storage servers is to employ massively parallel SIMD multiprocessor machines which traditionally have been used for performing high performance signal processing and numerical computations. The Sarnoff Real-time Systems **MAGIC** media server, the NCube/Oracle-M10 MediaNet server are two examples of this approach. Here we will take a brief look at the the MAGIC media server [3].

The MAGIC architecture consists of five main components: input boards, output boards, a processor array of SIMD processors, a special purpose disk array called *ImageVault*, and the host com-

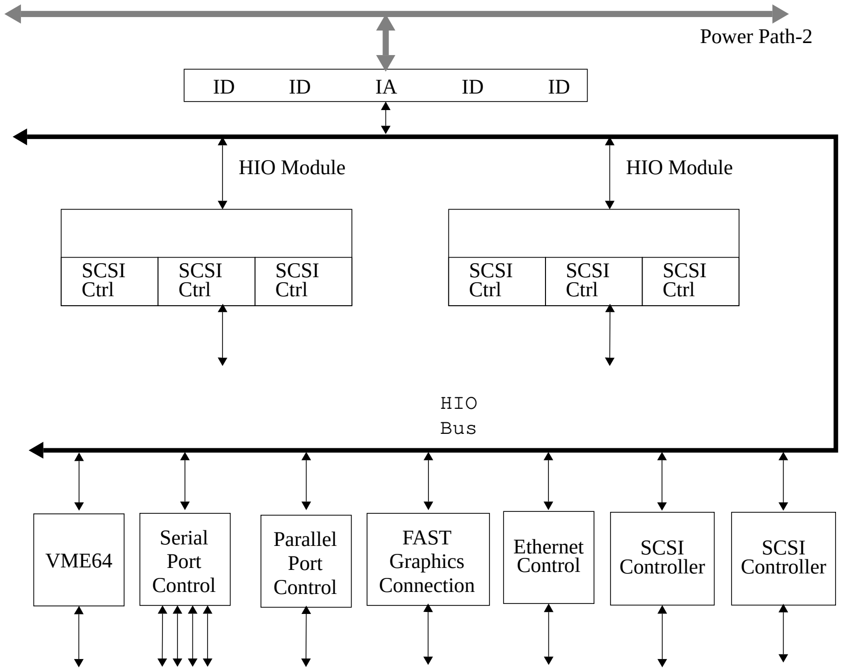

Figure 15. Details of the Power Channel Board

puter. The processor array uses a special purpose 144 bit VLIW video signal processor VSP developed at David Sarnoff Research Center. Each processor is capable of specialized video processing function, thus equipping the MAGIC server with a formidable real-time data processing power. The array is constructed out of boards of 128 processors and uses a proprietary Inter Processor Connection (IPC) network operating at 1.3 Gbps for communication between the processors. Each processor in the array has a a 160 Mbps data path to a SCSI-2 disk controller in the *ImageVault* disk array. Each disk controller controls one disk in the disk array. In the full scale configuration, the server has maximum 8 such boards or total 8196 processors and total 8196 disks in the disk array. The multimedia data is striped over all disks in the image vault allowing extensive concurrency and high throughput. The aggregate output bandwidth of the processor array is approximately 10.8 Gbps, which is connected to the variety of external interfaces for data rates of OC3, STS-3, DS-3, OC12. The input boards provide a seamless interfaces between the processor array's in-

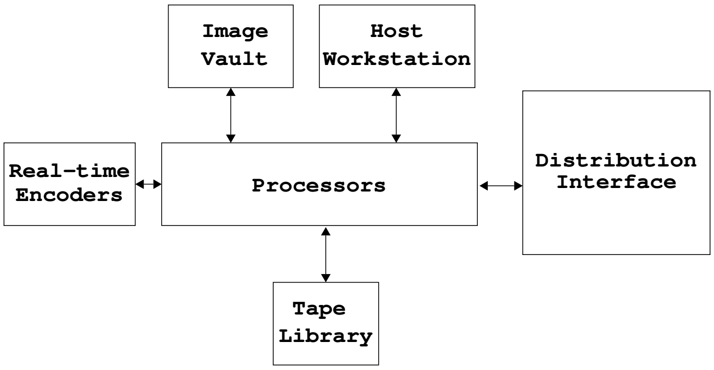

Figure 16. Silicon Graphics symmetric shared memory MIMD multiprocessor machine

put bandwidth of 5.4 Gbps and external data sources such as networks and real-time video encoders. Both these boards can handle in hardware incoming data in ATM formats. The host computer block in this architecture is implemented by a Silicon Graphics SGI Indy machine running UNIX like operating system. It communicates and controls the processor array through a GIO32 interface operating at 672 Mbps. It also communicates control messages from the clients to the processor array and also, implements billing and meta-database searches.

An example configuration of the MAGIC server with 1024 processors rated at aggregate 44,000 MIPS and an I/O BW of 2016 MB/sec ($\approx$ 1000 SCSI-2s!), sports close to a terabytes of IMAGEVAULT storage (with 1024 1 GB SCSI disks!). It supports a true-MOD DV service for as many as 6000 MPEG-1 connections and supports a stream operation latency of approximately 1.2 seconds [3].

3.5 Comparison of the Three Architectures

The MARS architecture has the advantage that it uses off-the-shelf commodity components such as Pentium PCs, ATM chips or switch based interconnects, and RAID based high performance reliable storage systems. The storage nodes constructed out of Pentium PCs provide excellent performance and have advantage of low cost due to high volume market. Also, MARS architecture provides a zero-copy direct path between the storage devices and the network and is truly scalable. However, distributed storage architecture such as MARS architecture may pose difficulties in a environment that requires media edits with very stringent QOS guarantees.

The primary drawback of the MIMD architecture based storage servers is that providing zero-copy data path from the storage devices to the external network is difficult. The data moves twice over the high-speed bus: once from the storage devices to the the main memory and from memory to the external network, reducing the available throughput by a factor of two. Also, the bus-based architecture of the server does not scale well and is prohibitively expensive.

Similarly, the SIMD architecture based storage servers represent a very expensive solution that employs a lot of proprietary components such as the VSP processors, the interconnection network for the processor array, and the IMAGEVAULT.

4 Data Layout and Access

The purpose of this section is to discuss layout of data in storage servers. We will follow a hierarchical approach: we will first describe data layouts in servers that store multiple video streams on a single disk system. We will highlight advantages, disadvantages of this approach. In the end, we will describe distributed data layouts in large scale storage servers. These data layouts can potentially use disk level and disk array level layouts and support high concurrency and parallelism.

4.1 Data Layouts on Disks

In this subsection, we describe mechanisms for managing the storage of real-time digital media such as motion video and audio. We call a sequence of continuously recorded video frames or audio samples a *Strand*. Each strand is organized on the disk in terms of media blocks separated by gaps of free space. The storage pattern for a strand consists of a *media block size,* M and a *gap size,* G, both in terms of sectors on the disk. We develop a model that generates a *storage pattern* for a given strand. The model relates disk and device characteristics (such as, disk read/write latency, and video capture/display times) to the media recording rate, and derives the block and gap sizes that result in continuous retrieval.

Given multiple strands each with its own storage pattern, we develop mechanisms for merging their storage so as to utilize disk space efficiently. In merging of strands, gaps between media blocks of one strand are filled with media blocks of other strands. Hence, the storage pattern and hence the continuity of the second strand may not be maintained strictly for each media block. However, it must be possible to introduce read-ahead and buffering of a finite number of blocks so as to preserve the storage pattern and continuity properties on an average over those finite number of blocks.

Otherwise, merging of real-time media strands (without violation of their continuity requirements) is impossible.

4.1.1 Preliminary Definitions

Frame is the basic unit of video.

Sample is the basic unit of audio

Strand is a sequence of continuously recorded audio samples or video frames.

Sector is the basic unit of disk storage.

Media block is a sequence of consecutive sectors storing media units (frames or samples). The size of a media block of a strand s_i is denoted by M_i.

Media Gap is the separation between successive blocks of a strand. The size of a media gap of a strand s_i is denoted by G_i.

Media blocks are assumed to be homogeneous, i.e., each block contains exactly one medium. This permits the file system to exploit the properties of each medium to independently optimize its storage. Heterogeneous blocks on the other hand may entail additional processing for combining media during storage and separating them during retrieval.

Each media block M_i and gap G_i is assumed to span an integral number of sectors. The tuple (M_i, G_i) is referred to as the storage pattern of strand s_i. A strand consists of repetitions of this pattern.

4.1.2 Storage Pattern of a Media Strand

The guiding factor in determining the storage pattern of a strand is the requirement of continuous retrieval. Continuity in retrieval can be guaranteed if each media block is available at the display device at or before the scheduled time of its playback. If blocks of a strand are located in a random fashion on a disk, separations between successive blocks of the strand may not be constrained enough to support continuous retrieval of media strands. At the other end of the spectrum, employing contiguous allocation of blocks can guarantee continuous retrieval, but entails enormous copying overheads during insertions and deletions. Both of these drawbacks can be overcome if blocks are allocated such that (1) the block size is bounded from below, and (2) the gap size is bounded from above, with both of these bounds determined from the requirement of continuity.

Equation 1 below formulates the continuity requirement. The continuity requirement is met if the time to skip over a gap and read the next media block (left hand side of the equation), does not ex-

Symbol	Explanation	Unit
s_{mu}	Size of media unit	bytes
s_{ds}	Size of a disk sector	bytes
M_i	Media block size in storage pattern of strand s_i	sectors
G_i	Gap size in storage pattern of strand s_i	sectors
B_i	Buffer size for retrieval of strand s_i	bytes
$\mathcal{R}_i$	Size of read-ahead for retrieval of strand s_i	bytes
r_{dt}	Rate of data transfer to or from disk	sectors/sec
T_{tt}	Track-to-track seek time of the disk	sec
T_{rot}	Rotation time of the disk	sec
T_{dis}	Display time of a media block	sec
L	LCM($M_1 + G_1$, $M_2 + G_2$)	sectors

Table 2. Symbols used in this paper

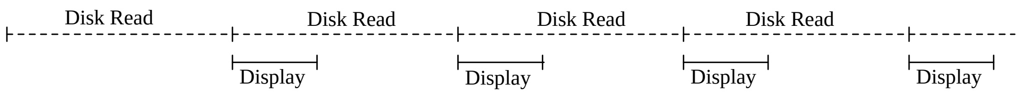

Figure 17. Continuous retrieval of media blocks of a strand

ceed the duration of its playback (right hand side of the equation) (see Figure 17).

$$\frac{M_i + G_i}{r_{dt}} \leq T_{dis} \qquad (1)$$

The storage pattern (M_i, G_i) for a strand must satisfy the above equation. However, since there are two variables and one equation, the storage pattern is not unique for a strand. The exact storage pattern may be determined by fixing one of the variables (for instance, the block size may be made equal to the memory page size in the system), and substituting its value in Equation 1 to compute the other.

If the disk is always sufficiently empty, then each strand may be stored on the disk exactly in accordance with its storage pattern. However, this may not be possible in reality. Typically, a disk would contain a large number of strands. Hence, blocks of a new strand have to be stored in the gaps of already existing strands on the disk. As a result, it is not guaranteed that the blocks of the new strand can be laid out on disk in accordance with its storage pattern. In order to utilize disk space efficiently, it is desirable to fill the gaps between media blocks of existing strands with media blocks of the new strand. In doing so, the continuity of the new strand may not be maintained strictly for each media block.

However, it must be possible to introduce read-ahead and buffering of a finite number of blocks so as to satisfy its continuity requirements on an average over those finite number of blocks. We will now derive the conditions for merging, and obtain the read-ahead and buffering requirements that guarantee continuous retrieval of media strands.

4.1.3 Merging the Storage of two Strands

Consider two media strands s_1 and s_2 with storage patterns (M_1, G_1) and (M_2, G_2), respectively (see Figure 18). Let the blocks of strand s_1 be laid out on the disk in accordance with its storage pattern (thus, guaranteeing continuous retrieval of s_1). To efficiently utilize the storage space, the file system must store the blocks of s_2 in the gaps of the pattern for s_1. The constraint in this process of merging is that the continuity requirement of s_2 must not be violated.

Real disks are organized in tracks, and a non-negligible track-to-track seek time, T_{tt} is incurred in changing between tracks. However, if the storage pattern of strand s_1 is chosen such that $T_{rot} * r_{dt}$ is a multiple of $M_1 + G_1$, and $M_1 \geq T_{tt} * r_{dt}$ (see Figure 19), the track-to-track seek time incurred during the retrieval of a merged strand s_2 is "hidden" in the time to jump over M_1 of s_1. Hence, in the analysis that follows, we will not separately consider track-to-track seek times.

Merging of s_2 into s_1 is straightforward if $G_1 = M_2$ and $M_1 = G_2$. In such a case, each media block of s_2 will exactly fit into a gap of s_1. However, in general, this can be very restrictive. If we introduce a buffer between the file system and the display (for temporary storage of media blocks during the retrieval of a strand), and read-ahead a sufficient number of media blocks before initiating the display of a strand (prefetch and store media blocks in the buffer), the continuity requirement may need to be satisfied only on an average. This is because, buffering can nullify the effects of jitter in the pattern, and result in the relaxation of the condition for merging multiple strands. We will now derive the exact condition for merging of two strands to be possible. We assume that the lengths of media strands may not be bounded at the time of the start of its recording, but the read-ahead and the buffer sizes are required to be bounded.

Intuitively, a strand s_2 can be merged into s_1 if the fraction of space occupied by media in s_2's pattern does not exceed the fraction of empty space available in strand s_1. That is,

$$\frac{M_2}{M_2 + G_2} \leq \frac{G_1}{M_1 + G_1} \qquad (2)$$

Simplification of the above equation yields the following proposition that formally derives the condition for merging:

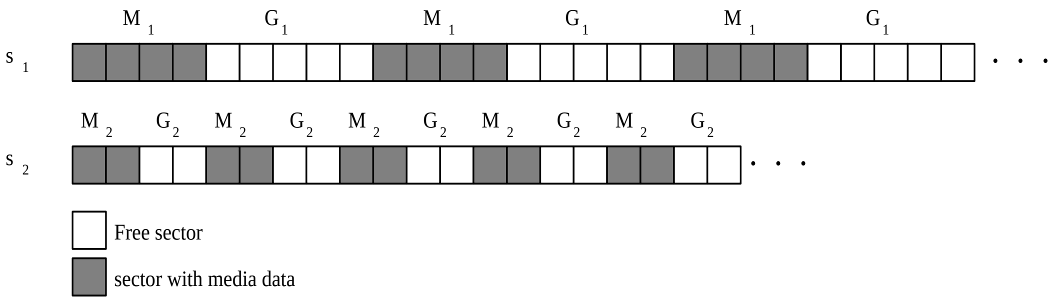

Figure 18. Media strands as repetitions of storage patterns

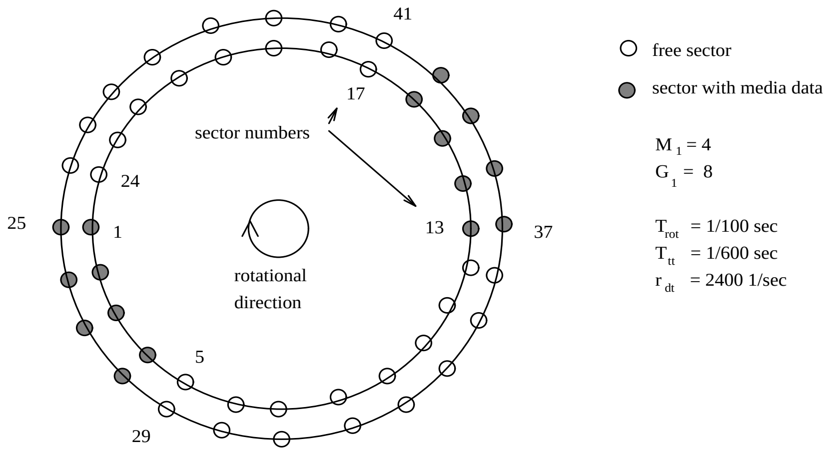

Figure 19. Availability of gaps of s_1 to store s_2

Proposition 1 *(***Merge Condition***) Let s_1 and s_2 be two strands with associated storage patterns (M_1, G_1) and (M_2, G_2) respectively. We can merge strand s_2 into strand s_1 if and only if*

$$\frac{G_1}{M_2} \geq \frac{M_1}{G_2} \tag{3}$$

The proof of this proposition can be found in the papers [34, 35].

4.1.4 Layout of Media Blocks

If the merge condition is satisfied, the simplest way to lay out s_2's media blocks is to fill them into s_1's gaps continuously starting from the very first gap. After storing the blocks belonging to one cycle, the remaining gaps would be left free. However, if s_2's pattern is sparse compared to the empty space available in s_1, a large number of media blocks of s_2 are read earlier than their time of display leading to peaks in buffering requirements. The buffering requirements

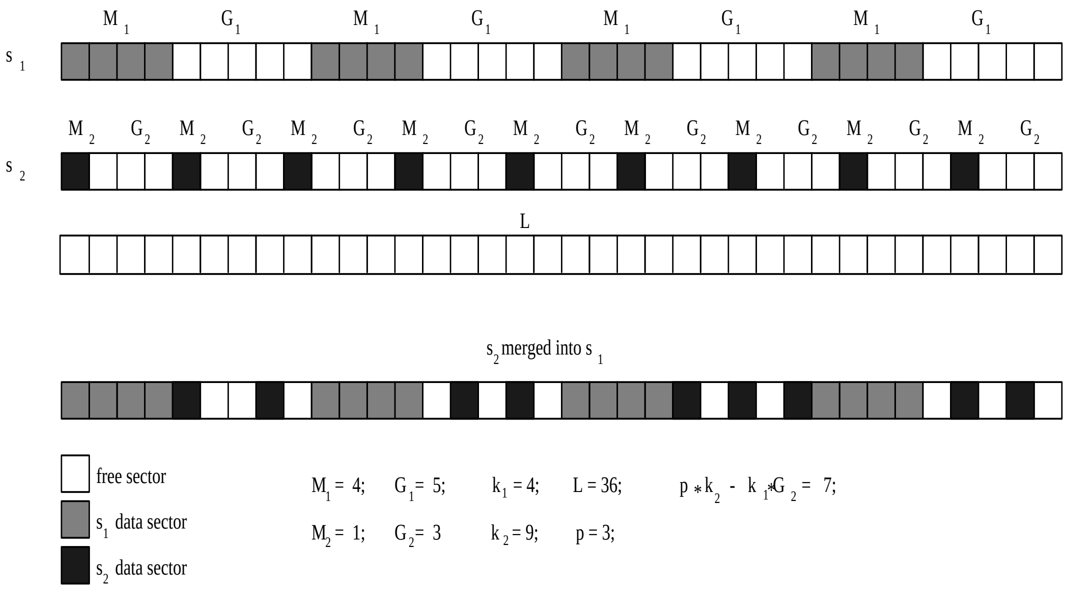

Figure 20. Layout of strand s_2 merged into strand s_1

can be reduced by distributing the media blocks uniformly over all the gaps of s_1. Formally, let k_1 patterns of strand s_1 and k_2 patterns of strand s_2 span over a merge cycle L. We define the pseudo block size of s_2 with respect to s_1 as the smallest integer M_2' such that

$$k_2 * M_2' \geq k_1 * G_1 \tag{4}$$

The merge condition guarantees that $M_2' \geq M_2$. If $M_2' = M_2$, then the merge condition is exactly satisfied, and the media blocks of s_2 occupy all the gaps of s_1. If $M_2' > M_2$, in each cycle L of the merge pattern, we can distribute the k_2 blocks of s_2 such that there are either $M_2' - 1 - M_2$ or $M_2' - M_2$ free sectors between consecutive media blocks of s_2. (To be exact, in each cycle of k_2 blocks, $k_2 * M_2' - k_1 * G_1$ blocks will have a separation of $M_2' - 1 - M_2$ sectors, and the remaining will have a separation of $M_2' - M_2$ sectors.) The (k_2+1)th block of s_2 will be stored exactly L sectors away from the first media block of s_2, resulting in a pattern of filled and free sectors that repeats after every L sectors, which is the length of one merge cycle (see Figure 20). Such a distribution is almost uniform in the sense that the number of free sectors between successive blocks of s_2 can differ only by one.

An important consequence of merging s_2's pattern into that of s_1 is that the continuity requirements, which hold for each block in s_2's pattern, are now guaranteed to hold only over an average of k_2 blocks in the merged pattern for s_2. Thus, a read ahead of at most k_2 blocks of s_2 may be necessary. We will now compute the exact read-ahead requirements of the merged strand.

4.1.5 Determining the Read-Ahead

Let us consider the playback of strand s_2 after merger. If $\mathcal{R}$ is the number of blocks that are read ahead, the process of displaying s_2 at the media output device starts immediately after $\mathcal{R}$ blocks have been transferred. In order to satisfy the continuity requirements, the total time to access and transfer any n blocks of s_2 *after the read ahead*, should not exceed the playback duration of $\mathcal{R}$ blocks that have been read ahead plus the $n-1$ subsequent blocks that have been transferred thereafter. Formulation of this condition yields:

$$\forall n : \sum_{i=1}^{n} T^i_{access} \leq (n-1) * T_{dis} + \mathcal{R} * T_{dis} \tag{5}$$

where T^i_{access} is the time needed to access the ith media block, and T_{dis} is the playback duration of a media block.

$$\mathcal{R} \geq \max_n \left(\frac{\sum_{i=1}^{n} T^i_{access}}{T_{dis}} - (n-1) \right) \tag{6}$$

However, since (1) s_2's merged pattern repeats after k_2 blocks and (2) the pattern of s_2 before merger meets the continuity requirement (Equation 1), satisfying Equation 6 for values of $n \leq k_2$ automatically implies that it is satisfied for all the media blocks of s_2.

The total time to access n media blocks of merged strand s_2 is the sum of the times to (1) read n blocks, (2) skip over media blocks of strand s_1, denoted by ϕ_1 (colored grey in Figure 20), that are present in the separations between the n blocks of s_2, and (3) skip over free sectors, denoted by ϕ_2 (colored white in Figure 20), that are present in the separations between the n blocks of s_2. That is,

$$\sum_{i=1}^{n} T^i_{access} = \frac{n * M_2}{r_{dt}} + \frac{\phi_1(n)}{r_{dt}} + \frac{\phi_2(n)}{r_{dt}} \tag{7}$$

where r_{dt} is the disk rotation rate in sectors per second (which is assumed to determine the speed at which blocks can be read or skipped).

The value of read-ahead R can be computed using following equation derived in [34].

$$\begin{aligned} \mathcal{R} \geq & \max_{1 \leq n \leq k_2} \\ & \left(\frac{n * (M_2' * (\frac{M_1}{G_1} + 1))}{M_2 + G_2} \right) \\ & - \frac{n}{M_2 + G_2} \\ & + n * \left(\frac{M_1}{M_2 + G_2} + 1 \right) \end{aligned} \tag{8}$$

The read-ahead reaches a maximum

when the term containing n in the above equation takes the maximum value. This term reaches a maximum when n is 1 or k_2 depending on whether its coefficient is positive or negative. It can be shown that the coefficient is positive [34]. Therefore, the read-ahead required to guarantee continuity of strand s_2 after merger can be derived as follows [34].

$$\mathcal{R} \geq \frac{k_2 * (M_2' * (\frac{M_1}{G_1} + 1) - M_2 - G_2)}{M_2 + G_2} \frac{M_1}{M_2 + G_2} + 1 \quad (9)$$

This condition can be used to determine the read-ahead when the pattern of s_1 and s_2 are given. However, if the continuity requirements are oversatisfied and T_{dis} is known, the read-ahead can be calculated more accurately using Equation 10.

$$\mathcal{R} \geq \max_{1 \leq n \leq k_2} \left(n * \left(\frac{M_2' * (\frac{M_1}{G_1} + 1)}{T_{dis} * r_{dt}} \right) \right) -n + \left(\frac{M_1}{T_{dis} * r_{dt}} + 1 \right) \quad (10)$$

4.1.6 Determining Buffer Sizes

The blocks that are retrieved during a read ahead need to be stored in an intermediate buffer between the file system and the media output device. When the display of the retrieved media blocks starts, media blocks are both (1) continuously removed from the buffer by the output device, and (2) continuously retrieved from the disk and added to the buffer by the file system. The size of the buffer grows and diminishes during the retrieval. Continuity of playback requires that the buffer should never be empty when the output device is ready to remove a block from the buffer. If s_2 were to be stored as an unmerged pattern (M_2, G_2) and continuity requirements were satisfied exactly, each block would be transferred to the buffer and then played back immediately. However, after the merge, continuity requirements are satisfied only over a merge cycle (i.e., k_2 media blocks), and the buffer size starts with being equal to the read ahead, and grows and diminishes over a period of the merge cycle at the end of which it is equal to the read ahead again. The maximum value of buffers assuming that the continuity requirements are satisfied exactly can be computed by following equation [34],

$$\mathcal{B} = \mathcal{R} + \left\lceil \frac{q * G_1 + \min(r * (M_1 + G_1), G_1)}{\lfloor \frac{k_1 * G_1}{k_2} \rfloor} \right\rceil -(n-1) \quad (11)$$

where, $\mathcal{R}$ is the read-ahead given by Equation 9 or 10. Generally additional buffering requirements will result from oversatisfied continuity requirements. These can be caused by variations in the rate of playback (e.g., in slow motion), where the retrieval of media blocks proceeds faster than their display. Moreover the storage pattern of a strand can be determined in a way, that continuity requirements are over-satisfied even during the playback with the fastest rate. File systems designed to satisfy several requests simultaneously have to compute those kind of storage patterns. In order to prevent unbounded accumulation of media blocks in buffers, the disk can switch to some other task after all the buffers allocated to the retrieval of a media strand are filled, and switch back when sufficient buffers become empty. However, after the disk switches to some other task, the disk head may have moved to a random location. Hence, besides the time used for other tasks T_{ot} the disk head may have to incur maximum seek (and latency) time, l_{seek}^{max} before being able to resume the transfer of blocks of the earlier media strand. Thus, in order to guarantee that the display does not run out of media blocks during a switch to another task, the disk must read ahead additional blocks into buffers before the switch, given by:

$$(T_{ot} + l_{seek}^{max}) * \frac{r_{dt}}{M_2 + G_2}$$

4.1.7 Merging More than Two Strands

The above binary merging techniques can be easily extended to interspersed storage of three or more strands. Let us suppose that we have to merge m strands, s_1, s_2, s_3, ..., s_m. When the first two strands s_1 and s_2 are merged, the resulting storage pattern can be viewed as that of a *composite strand* s_{12}, whose media block size M_{12} and gap size G_{12} are given by:

$$\begin{aligned} M_{12} &= k_1 * M_1 + k_2 * M_2 \\ G_{12} &= k_1 * G_1 - k_2 * M_2 \\ &= k_2 * G_2 - k_1 * M_1 \end{aligned}$$

where $L_{12} = \text{LCM}(M_1 + G_1, M_2 + G_2)$ is the length of the pattern, $k_1 = \frac{L}{M_1+G_1}$, and $k_2 = \frac{L}{M_2+G_2}$. The merger of s_3 with s_{12} can now be carried out using the binary merging techniques to yield strand s_{123}. Continuing the merge operation in this fashion, the m strands can be merged using a sequence of $m-1$ binary merge operations to result in a final composite strand, $s_{1,2,3,\ldots m}$ The derivations for read-ahead and buffer sizes of binary merging can also be extended to *M-ary merging* in a similar fashion.

Deletion of a strand s_s from a composite strand s_c can be slightly tricky: the sectors occupied by the media blocks of strand s_s are released back to the pool of gaps of the composite strand s_c. Assume (M_s, G_s) is the pattern of s_s, and

(M_c, G_c) of s_c. After deletion, the resulting strand s'_c will have a pattern given by (M'_c, G'_c), where $M'_c = M_c - M_s$, and $G'_c = G_c + M_s$. Generally the pattern cycle length $(M_c + G_c)$ of the composite strand remains unchanged.

However, when the strand to be deleted was the last one to be merged, deletion restores the pattern to that before the last merge. For example, if strand s_m is deleted from a composite strand $s_{1,2,\ldots,m}$, the parameters $L_{1,2,\ldots,m}$, $G_{1,2,\ldots,m}$, and $M_{1,2,\ldots,m}$ are restored back to $L_{1,2,\ldots,m-1}$, $G_{1,2,\ldots,m-1}$, and $M_{1,2,\ldots,m-1}$, respectively, which are the values prior to the merging of strand s_m into $s_{1,2,\ldots m-1}$.

4.2 Data Layouts in Scalable MOD Servers

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 21. 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 [10, 11, 26]. 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 21 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.

Note that each storage node stores the chunks assigned to it using a local storage policy. For example, if the node uses ordinary disk storage, the *storage pattern* based methods described above can be employed. On the other hand, if the node uses a disk array, the way each chunk is stored or striped on the array may depend 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 [13] 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.

4.3 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

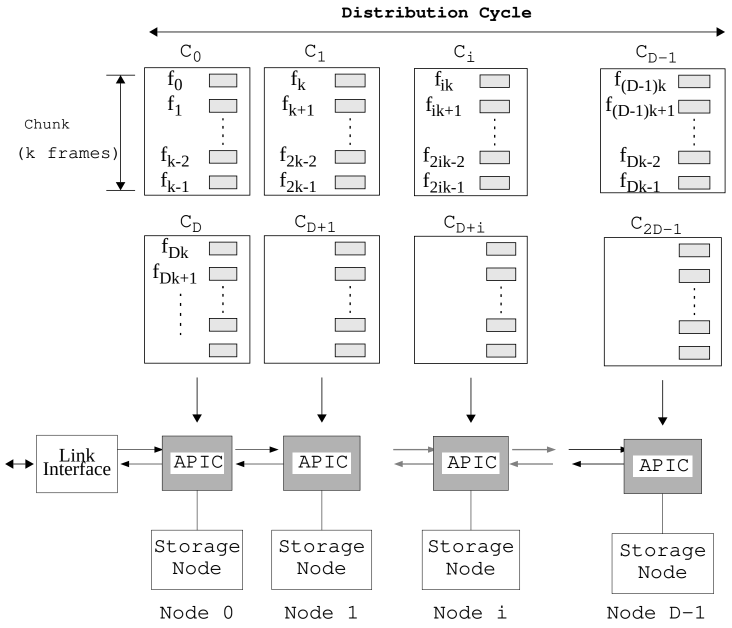

Figure 21. Distributed Chunked Layout

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

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.

5 Admission Control

A multimedia server may be required to service multiple subscriber requests simultaneously. In the best scenario, all the subscribers request the retrieval of the same media stream and the multimedia server only needs to retrieve the media stream once from the disk and then multicast it to all the subscribers. However, more often than not, different subscribers may request retrieval of different media streams; even when the same media stream is being requested by multiple subscribers, there may be phase differences among their requests (such as each subscriber retrieving a different portion of the stream at the same time). A simple scheme for guaranteeing real-time retrieval of each of the requests is to dedicate a disk head per request. However, this limits the total number of requests that can be serviced simultaneously to the number of disk heads, which may be orders of magnitude smaller than the number of subscribers needed to make the multimedia on-demand server economically viable.

5.1 Admission Control for Disk system based storage servers

In this section, we develop algorithms for maximizing the number of subscriber requests that can be serviced simultaneously, under the constraint that the retrieval of all of the requested media streams must proceed at their respective playback rates. In order to precisely formulate this requirement, let us suppose that a multimedia server has admitted r subscribers, each of whom is retrieving a media stream. The multimedia server multiplexes among all the r subscriber requests, retrieving a finite number of blocks k_i of each request $i \in [1, r]$, before switching to the next request. Each sequence of $k_1, k_2, ..., k_r$ retrievals constitutes a *service round*, and the multimedia server repeatedly executes service rounds until the completion of all the requests. Whereas the granularity and scattering parameters govern the rate of retrieval of successive media blocks of a media stream, switching from one media stream to another may entail an overhead of up to the maximum seek time to move the disk head from a block in one stream to a block of another stream (since the layout does not constrain the relative positions of different streams). Thus, the total time spent retrieving k_i blocks of i^{th} request in a service round consists of: (1) α: the overhead of switching from the $(i-1)$th request to the ith request, and then transferring the first block of the ith request, and (2) β: the time to transfer remaining $(k_i - 1)$ blocks of the ith request in this service round. If $\eta^1_{ms}, \eta^2_{ms}, ..., \eta^r_{ms}$ denote the granularities, $l^1_{ds}, l^2_{ds}, ..., l^r_{ds}$ denote the scattering parameters, and $\mathcal{R}^1_{pl}, \mathcal{R}^2_{pl}, ..., \mathcal{R}^r_{pl}$ denote the playback rates of the r streams being retrieved simultaneously, then the continuous retrieval of each of the requests can be guaranteed if and only if the total service time per round does not exceed the minimum of the playback durations of all the requests That is,

$$\sum_{i=1}^{r}(\underbrace{(l_{seek}^{max}\frac{\eta_{ms}^{i} * s_{mu}^{i}}{\mathcal{R}_{dr}}) +}_{\alpha} \underbrace{(k_i - 1) * (l_{ds}^{i} + \frac{\eta_{ms}^{i} * s_{mu}^{i}}{\mathcal{R}_{dr}}))}_{\beta} \leq \min_{i\in[1,r]}(k_i * \frac{\eta_{ms}^{i}}{\mathcal{R}_{pl}^{i}}) \quad (12)$$

The multimedia server can service all the r requests simultaneously if and only if $k_1, k_2, ..., k_r$ can be determined such that above equation is satisfied. Since this formulation contains r variables and only one equation, determination of the values of $k_1, k_2, ..., k_r$ requires additional policies.

The simplest policy is to assign equal values to $k_1, k_2, ...,$ and k_r, yielding what is generally referred to as a round robin servicing algorithm. Formally, if $k_1 = k_2 = \cdots = k_r = k$, Equation 12 yields the maximum number of subscriber requests that can be serviced in a round robin algorithm to be:

$$r_{max}^{c} = \frac{\eta_{ms}^{avg}}{\mathcal{R}_{pl}^{max} * (l_{ds}^{avg} + \frac{\eta_{ms}^{avg} * s_{mu}^{avg}}{\mathcal{R}_{dr}})} \quad (13)$$

Clearly, the number of subscriber requests that can be serviced by the round robin algorithm is limited by the request with maximum playback rate. Furthermore, the request with the maximum playback rate will have retrieved exactly the number of data blocks it needs for the duration of a service round, whereas other requests whose playback rates are smaller will have retrieved more data blocks than they need in each service round. Consequently, by reducing the number of data blocks retrieved per service round for such subscriber requests, it may be possible to accommodate larger number of subscribers.

At the other extreme is an *exhaustive* algorithm, which initially assigns the minimum value to each k_i, i.e., $k_1 = k_2 = \cdots = k_r = 1$, and then selectively increments the values of each k_i until the continuous retrieval equation (Equation 12) is satisfied. The values of $k_1, k_2, ..., k_r$ thus obtained can be shown to be minimal, thereby guaranteeing that minimal time is spent on each subscriber and maximum number of subscribers are serviced during a service round. However, the computational overhead of such an algorithm can be prohibitive.

We propose a Quality Proportional Multi-subscriber Servicing (QPMS) algorithm, in which the number of blocks retrieved during each service round for each subscriber request is proportional to its playback rate. Thus, if k is the proportionality constant, we get, $k_1 = k * \mathcal{R}_{pl}^1$, $k_2 = k * \mathcal{R}_{pl}^2$, ..., $k_r = k * \mathcal{R}_{pl}^r$. Substituting the values of k_i in Equation (12) yields:

$$\sum_{i=1}^{r}((l_{seek}^{max} + \frac{\eta_{ms}^i * s_{mu}^i}{\mathcal{R}_{dr}}) + (k * \mathcal{R}_{pl}^i - 1) * (l_{ds}^i + \frac{\eta_{ms}^i * s_{mu}^i}{\mathcal{R}_{dr}})) \leq k * \min_{i \in [1,r]}(\eta_{ms}^i) \quad (14)$$

Furthermore, we can derive the maximum number of requests r_{max}^p that can be serviced simultaneously from Equation 14, and is given by:

$$r_{max}^p \leq \lfloor \frac{\eta_{ms}^{avg}}{\mathcal{R}_{pl}^{avg} * (l_{ds}^{avg} + \frac{\eta_{ms}^{avg} * s_{mu}^{avg}}{\mathcal{R}_{dr}})} \rfloor \quad (15)$$

Given the number of subscriber requests $r \leq r_{max}^p$, Equation (4) can also be used to determine k, from which we can obtain the values of k_1, k_2, ..., k_r. However, the values of k_i so obtained may not be integral. Since playback of media streams proceed in terms of quanta such as video frames, if each block is assumed to contain a quantum, then retrieval of a fraction of a block cannot be used for playback, causing the playback to starve until the remaining fraction arrives, possibly in the next service round. To avert such situations, the number of blocks retrieved in each service round must be integral for all the requests, to accomplish which we have proposed a staggered toggling technique in [18]. In this technique, the number of blocks retrieved for each subscriber request i during each round toggles between $\lfloor k_i \rfloor$ and $\lceil k_i \rceil$, so that on an average, the retrieval rate is k_i blocks per service round. Furthermore, the toggling for various subscriber requests is staggered: the toggling up for one subscriber is matched by the toggling down for another subscriber, so that, over all, there is no net increase in the service time of any round and hence, no violation of continuity requirements. The subscribers for whom the number of blocks toggles up are exactly those who may face an impending starvation owing to an earlier toggling down in a preceding service round. This technique is shown to be provably correct, i.e., it satisfies the continuity requirements of each media stream in each service round [18].

It can be seen from Equation 15 that the QPMS algorithm can admit a much larger number of subscriber requests compared to the round robin algorithm, if there is a large variation in the playback rates of media streams requested by subscribers. In fact, it can be shown that the QPMS algorithm is provably optimal, that is, given any set of r subscriber requests a priori, if there is a set of $k'_1, k'_2, ..., k'_r$ satisfying the continuous retrieval equation (Equation 12), then the QPMS algorithm is guaranteed to yield non-negative integers $k_1, k_2, ..., k_r$ that also satisfy Equation 12.

If the total number of subscriber requests that need to be admitted are not known a priori, but may change dynamically, then continuity should not only be maintained during steady state, but also during transitions accompanying dynamic admission of new subscribers. To see why, suppose that a multimedia server receives a new request while servicing a set of r requests. If $r+1 \leq r^p_{max}$, then the multimedia server can compute the new value of k, namely k^{new}, necessary for satisfying $(r+1)$ requests. If $k^{new} = k^{old}$ (where, k^{old} is the value of k using which the multimedia server has been servicing the existing r requests), then the multimedia server can immediately admit the $(r+1)$th subscriber request. However, if $k^{new} \neq k^{old}$, then $k^{new} > k^{old}$, and the multimedia server computes the new values of k_i, namely $k_i^{new} = k^{new} * \mathcal{R}^i_{pl}$, for all $i \in [1, r+1]$, and begins transferring k_i^{new} blocks for each $i \in [1, r+1]$. However, during the first service round after admitting the $(r+1)^{th}$ subscriber, the number of blocks available to the ith subscriber for playback are those of the previous service round, namely k_i^{old}, the time for which may fall short of the time spent to transfer k_i^{new} blocks for each $i \in [1, r+1]$, leading to a violation of continuity requirement during this transitional service round. In order to avoid such a transitional discontinuity, the QPMS algorithm can be modified such that the time to transfer (k_i+1) blocks (i.e., $k*\mathcal{R}^i_{pl}$ instead of $k*\mathcal{R}^i_{pl}-1$) is used in the left hand side of Equation (4), but the time to playback k_i blocks is used on its right hand side (i.e., no change in the right hand side). In such a case, it can be shown that $\forall r < r^p_{max}$, the difference between right hand side and the left hand side of Equation (4) increases with increase in k, as a consequence of which a smooth transition from k^{old} to k^{new} can be guaranteed in steps of 1, yielding a QPMS admission control algorithm that supports both transient and steady state continuity of media playback for subscribers.

5.2 Admission Control in Large Scale Storage Servers

The large scale storage servers that consist of multiple co-operating storage nodes providing data for a multimedia document need to to employ hierarchical admission control. Each storage node in such an architecture has finite resources in the form of buffer of capacity B and the storage bandwidth R_N. A fraction of these resources must be reserved for each active periodic real-time connection to provide guaranteed accesses. Such resource reservation requires an admission control procedure that must be consulted before any new reservations are made. Since, compressed data is typically VBR in nature, providing deterministic guarantees requires the wasteful approach of peak BW reservation. Also, multimedia data is tolerant of occasional errors and missed deadlines. Given this, statistical or probabilistic guarantees, an approach in which guarantees can be occasionally violated, if the "bursts" for different connections collide, is more useful. For this approach to work successfully, multiple VBR streams must be statistically multiplexed together. In a cycle based scheduling scheme such as one described above, the lack of sufficient BW due to burst collisions translates to total retrieval time exceeding the cycle length required to meet the periodic requirements of all clients. We characterize such multiplexing overloads in two steps: the buffer overflow and the cycle overflow. Some of this characterization is similar to the one in [10, 11].

Due to the VBR nature of streams, the amount of data fetched at each node for each connection varies from cycle to cycle, or stated formally, it is a random variable (RV) $D_i, i \in [1 \ldots C_a]$, whose distribution depends on the statistical properties of the MPEG compressed document being accessed. If such distribution is known for each RV D_i, the probability that the sum of D_i's exceeds the total buffer capacity B can be computed. If the tolerable buffer overflow probability is $P_{buffover}$, and if the Equation 16 is true for $n = C_a$, then upon admitting $(C_a + 1)^{th}$ client, it should still be valid.

$$Pr\left[\sum_{i=1}^{n} D_i \geq B\right] \leq P_{buffover} \quad (16)$$

Similarly, the probability of cycle overflow can be estimated using the knowledge of node level layout and node throughput characteristics, such as disk transfer rate, minimum/maximum disk seek and rotation times. The various node level prefetch schemes and transmit options will have different cycle overflow performance. Specifically, if for a given setup, $T_r(D_i)$ represents time to fetch data of size D_i, and if Equation 17 is true for $n = C_a$, then upon admitting $(C_a+1)^{th}$ client, Equation 17 should still hold true.

$$Pr\left[\sum_{i=1}^{n} T_r(D_i) \geq T_{cycle}\right] \leq P_{cycleover} \quad (17)$$

Note that Equation 16 and Equation 17 need to be satisfied at each node in our architecture. Thus, the admission control policy uses a computation procedure at each node which, given a pair of values $[P_{buffover}, P_{cycleover}]$, verifies if the aforementioned two equations are satisfied in the event of admission of a new request. This formulation is versatile: for example, an aggressive admission control can be realized by using higher buffer and cycle overload probabilities. Alternatively, a conservative admission control policy can be realized using smaller overload probabilities. We can also construct a service with multiple levels of QOS. Specifically, we can consider different values for 2-tuples $[P_{buffover}, P_{cycleover}]$ to represent different classes of clients. We order such classes in increasing order of overflow probabilities. Thus, a client belonging to class C_1 can tolerate smaller overflow probabilities than the client belonging to the class C_5. In other words, $P_{buffover,1} < P_{buffover,5}$ and $P_{cycleover,1} < P_{cycleover,5}$. In the event of overloads, the service is discontinued for clients from higher classes. Thus, the server can overbook its resources and on an average admit larger number of customers.

In the above formulation of performance of statistical multiplexing, Equation 16 depends on the chunk size used by the layout over the nodes, whereas Equation 17 depends on how the chunks are stored at each node, that is on the node level data layout. The node level data layout depends on the local storage devices at the node. If the storage node has a disk array, which can be of two types: one, an ordinary disk array in which all disks are used to store the data and the second, called Redundant Array of Inexpensive Disks (RAID) [32], in which some of the disks are used to store redundant information to improve reliability and data availability in the event of disk failures. When using commercial RAID devices, as we intend to do, the node level striping scheme is fixed to the one enforced by the RAID device. Given this, the chunk size happens to be the only design parameter that affects the efficiency of statistical multiplexing. The larger the chunk size, the smaller is the variability of the chunk size and offered load. Also, large chunk sizes amortize seek and rotational latency overhead at the disks in a disk array over large data transfers and thus, improve disk utilization. On the other hand, for good subjective quality of *ff* by chunk skipping, a small chunk size is desirable. In the presence of a large number of concurrent clients, small chunk size minimizes per connection buffer requirements. Also, a smaller chunk size allows quick response time for interactive playout control operations.

6 Multimedia Information Caching - Technique for realizing TVOD

On-demand movie viewership patterns are highly dependent on the time of the day. Drawing a parallel to the telephone network, there are peak hours during which the traffic is maximum and the majority of the titles demanded is a small subset of the most recent set of hit movies. As a result, there is significant confusion as to whether True Video on Demand is more lucrative than Near Video on Demand. Caution has been expressed about the success of True Video on Demand because it is felt that Cable and Telephone companies pushing Video on Demand are investing heavily to create massive storage providers to deliver thousands of programs when all that consumers want are the eight or ten hit movies every month.

Information Caching has the potential not only to smoothen out the peak, but also to amortize storage costs and maximize network utilization. In the system configuration proposed at the UCSD Multimedia Laboratory, a synergy between several enterprises occurs: *Storage Providers* that manage information storage at multimedia servers (a role akin to that of video rental stores and libraries in today's context), *Network Providers*, that are responsible for media transport over integrated networks (a role akin to that of cable and telephone and cable companies of today), and *Content Providers* such as entertainment houses, new producers etc., that offer a multitude of services to subscriber homes using multimedia servers and broadband networks. In the proposed architecture, multimedia servers of suitable capacities are installed at the different hierarchical levels and function as temporary caches for information delivered from metropolitan repositories. The locations of switching equipment in the network are obvious choices for locating the multimedia servers. For managing distribution of the selected multimedia programs to each subscriber's home, we propose *Distribution Agents* that negotiate the storage and retrieval times of programs with storage and network providers, and program authorizations with the content providers. The distribution agents judiciously determine locations of programs and their distribution times depending upon the anticipated demand for the future, the costs of storage and the utilization of bandwidth.

To facilitate the process of program selection, we propose *Personal Service Agents* (PSAs) that search and locate programs that match each subscriber's needs, and customize the content providers' offerings to deliver personalized services. The main functions of a PSA are: (i) It monitors each subscriber and constructs a behavioral profile for the subscriber. (ii) Based on this profile, the PSA queries the network directory service to identify content providers offering relevant programs. (iii) Then, the PSA contacts each of the content providers, navigates

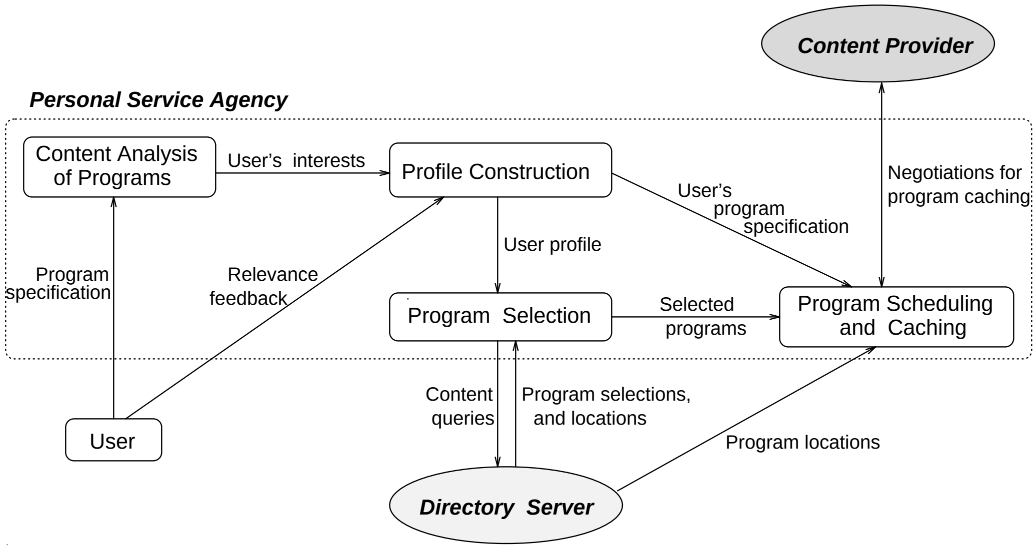

Figure 22. Functions of a Personal Service Agent (PSA)

through their content bases, and constructs a list of programs relevant to the subscriber. (iv) Once the subscriber makes a selection of programs to view, the PSA continuously interacts with the content bases and chooses portions of programs and even their levels of detail to display to the subscriber. (v) The PSA also solicits explicit *relevance feedback* from the subscriber for its selections, so as to tailor its selections in the future to better suit the subscriber's needs [36].

6.1 Program Retrieval and Caching

The different programs requested by a user on a personal channel could be offered by different content providers. The PSA can judiciously schedule retrieval (from storage providers) and subsequent transmission (via network providers) of the programs so as to minimize costs borne by the user. For instance, a PSA can retrieve programs *a priori* and cache them either at the user's site or at a neighborhood server, during a period when the network and server are relatively underutilized. In practice, however, the storage capacities necessary for caching all the programs that a user views daily on a personal channel are prohibitively large: For instance, assuming that a user views upto five, one hour long video programs daily, and that the videos are MPEG-2 encoded (requiring storage of 4 Megabits/second for each program), storage of 18 Gigabytes is necessary; Even assuming that in future, storage prices drop down to a half of the current list price of 25 cents per Megabyte, the investment necessary for providing 18 Gigabytes of storage at a user's site is \$2,250, which is about seven-folds the price of a high quality VCR. Therefore, based on the above computations, we infer that caching all of the programs on a personal channel well in advance of their viewing times, at a user's site itself is not likely to be economically viable.

Resource optimizations are most likely when several users indicate similar preferences for programs (e.g., popular movies). In the best case, all the users may choose to view the same programs, at the same times. For such users, PSAs can arrange with the content providers to simulcast programs of the users' common choice, thereby amortizing network transmission costs amongst all those users. More often than not, however, users' preferred viewing times do not match. In such cases, multiple independent transmissions of the requested program may have to be incurred, all the way from a storage server located at one end of a network to users' sites located at another end. As an example, consider the metropolitan-area network shown in Figure 23(a). Suppose that a program $\mathcal{P}$, created by a content provider at 1pm and stored at server S_1, is requested by users U_1, U_2, and U_3 for viewing at 1pm, 2pm, and 11pm, respectively. Further, suppose that (i) the storage cost for program $\mathcal{P}$ at the server increases with duration of storage (the rate of increase depending upon program $\mathcal{P}$'s storage space requirement), (ii) the cost of each transmission of $\mathcal{P}$ over the network links is fixed (depending upon the bandwidth required for transmission of program $\mathcal{P}$), and (iii) the delay between the transmission of program $\mathcal{P}$ from the servers and its reception at the users' sites is negligible compared to the total duration of program $\mathcal{P}$. For servicing users U_1, U_2, and U_3, program $\mathcal{P}$ must be transmitted thrice over the network, at 1pm, 2pm, and 11pm, respectively, incurring a cost of \$18 each time.

With a view to reducing service costs, we propose that metropolitan-area networks be configured with multimedia servers of suitable capacities installed at various strategic locations, such as basements of residential complexes, recreational areas in townships, and libraries of universities. The neighborhood servers can function as *tem-*

porary caches for frequently accessed video programs, thereby avoiding the cost of repeated transmissions of the same programs all the way from a remote metropolitan server. In order to facilitate information caching, we propose that the storage servers should be co-located with the network switches to constitute "multimedia hubs" that support both storage and transmission of programs. In such a configuration, it is possible for neighborhood servers to cache the video programs *as and when they are transmitted* via their respective switches (possibly for servicing users in other neighborhoods), thereby avoiding even the cost of program transmissions to the neighborhood servers explicitly for the purpose of caching.

Figure 23(b) illustrates a possible configuration for the metropolitan-area network of Figure 23(a). Intermediate servers S_2 and S_3 function as neighborhood stores for programs requested by the users U_1, U_2 and U_3. By caching programs at servers S_2 and S_3, PSAs can avoid the costs of retransmissions from the distant metropolitan-area server S_1. For instance, caching program $\mathcal{P}$ at server S_3 when the program is transmitted to user U_1 reduces the overall transmission cost by \$36.

In practice, there are interesting tradeoffs between the cost of renting storage space at neighborhood servers and the cost of repeated transmissions from a metropolitan server, which must be evaluated by PSAs and content providers before deciding to cache programs. In the following section, we elaborate on these tradeoffs and devise simple, yet effective strategies for information caching in metropolitan area networks.

6.2 Information Caching

At the time of caching, PSAs or the content providers must evaluate the various storage and network alternatives. The storage costs may differ from server to server, depending upon the capacities of the servers, their data transfer rates, and current demands. Likewise, transmission costs of programs may also differ from one network link to another, depending upon the available link bandwidth, and the desired real-time performance: end-to-end delay, delay jitter, and reliability of transmission.

In order to illustrate the storage-network tradeoffs, as before, consider the simple, tandem configuration of storage servers shown in Figure 24(a). Suppose that users U_1, U_2, and U_3, all belonging to the same neighborhood (and hence, connected to the same neighborhood server S_3) request for a program $\mathcal{P}$ that is created and stored at the metropolitan server, by a content provider at 1pm (see Figure 24(a)). Clearly, no optimizations are possible for servicing the first

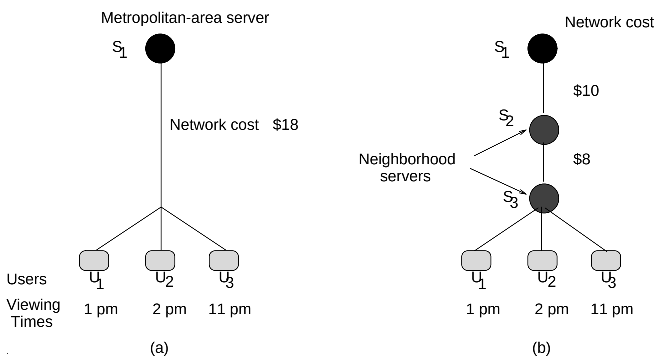

Figure 23. (a) A metropolitan-area network with a central server S_1, which transmits program $\mathcal{P}$ to users U_1, U_2, and U_3; in this case, since users' viewing times do not match, three independent transmission of program $\mathcal{P}$ are necessary. (b) A more efficient network configuration, with intermediate servers S_2 and S_3 functioning as neighborhood caches. In this case, program $\mathcal{P}$ can be cached at server S_3 while it is transmitted down the network, to service user U_1, thereby reducing the overall transmission cost by 66%.

user U_1 at 1pm: the program must be transmitted all the way from server S_1 to user U_1. However, during this transmission, any of the servers S_2 and S_3 can cache the program for future usage. Whereas caching at S_3 only entails a storage cost, caching at S_2 entails an additional network cost, but a lower storage cost. Considering such tradeoffs, for the period $[1pm, 11pm]$, it is cheaper to cache program $\mathcal{P}$ at server S_2, entailing a total cost of $54, as compared to a cost of $68 if the program $\mathcal{P}$ were cached at server S_3. The *Caching Schedule* thus derived (see Figure 24(b)), although it amortizes storage costs between U_2 and U_3, does not constitute the the overall optimum. For instance, a schedule that besides caching program $\mathcal{P}$ at server S_2 for the period $[1pm, 11pm]$, also caches program $\mathcal{P}$ at server S_3 for the period $[1pm, 2pm]$ (see Figure 24(c)) entails a lower overall service cost, amounting to $51.

In general, the storage-network optimization is carried out at the time of scheduling programs on users' personal channels. PSAs, together with the content providers, must determine *when, where*, and for *how long* programs must be cached, so as to minimize the costs borne by users. [31, 30] outline an efficient dynamic programming technique that can be used by a PSA to derive an

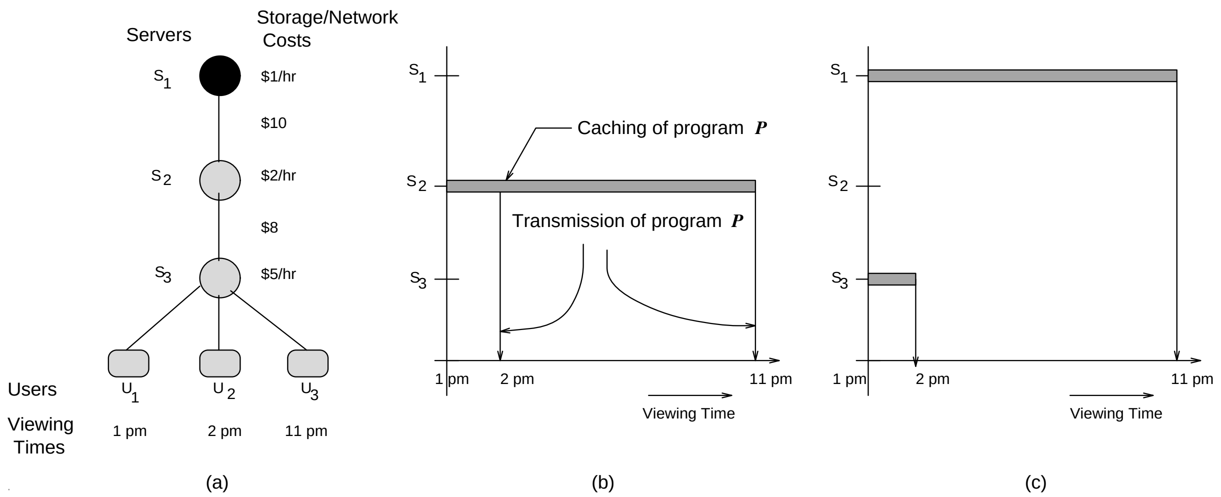

Figure 24. Illustration of storage-network tradeoffs: Figure (a) depicts the network configuration: S_1 is the metropolitan-area server, S_2 and S_3 are the neighborhood caches. Program $\mathcal{P}$, created and stored at S_1 at 1 pm is requested by users U_1, U_2, and U_3 for viewing at 1 pm, 2 pm, and 11 pm, respectively. The storage and transmission costs for $\mathcal{P}$ are indicated in the figure. Figure (b) illustrates a caching schedule that amortizes program storage between users U_2 and U_3. The total cost of this schedule is \$54; Figure (c) represents the overall optimal schedule, the total cost of which is \$51.

optimal caching schedule for a program $\mathcal{P}$ that is requested for viewing by several users located in the same neighborhood of a metropolis.

Surprisingly, even in the worst-case, this heuristic performs *no worse than twice the optimal*; this is analytically provable [30]. Preliminary performance simulations indicate that in practice, this caching strategy performs even better. Several thousand configurations of storage servers, together with several combinations of users' viewing schedules were generated and the performance of the heuristic caching strategy was compared with that of the optimal. These simulations reveal that the heuristic caching strategy performs within 1% of the optimum on an average, with the worst-case deviation being at most 10%.

6.3 Discussion

There are some realistic generalizations of the above caching problem. Until now, we have assumed that entire programs are cached between storage servers. In many a case, users' interests pertain only to selected portions of programs offered by content providers. Moreover, the storage-network tradeoffs may differ for the different media components of a program. For example, due to the larger bandwidth and storage requirements of the video component, it may be profitable to cache the video component at a neighborhood server. The audio component, on the other hand, requires much smaller bandwidth and storage space, and hence, repeated transmissions of the audio component of programs from a metropolitan server may be preferred.

At the time of caching, PSAs must not only consider heterogeneity in storage costs and transmission costs, but they must also adapt to any changes that may occur in these costs. For instance, as in today's telephone networks, there may be "peak hours" (e.g., 8am to 8pm) when the networks are in greater demand (this holds for storage servers too), and hence, more expensive to use. Cost changes may occur even on-the-fly; for example, a storage provider, upon anticipating an overcommitment of server resources, may increase the storage cost so as to discourage further demand. The caching strategies we propose adapt easily to dynamic changes in storage and network costs. Consequently, these cost change times must be incorporated into the dynamic programming technique described in [31, 30]. The complexity of determining caching schedules is now a function of both the number of distinct viewing times of users and the number of cost-change times (and of course, the number of servers).

7 Concluding Remarks

In this chapter, we have explored and expanded on each of the aspects of video servers and have given performance evaluations of various architectures ranging from those that are in the market to those that are in prototype form in research labs. We envision that personalized multimedia on demand services will trigger a radical change in the role of many enterprises of today: For instance, video rentals will be reduced to the role of storage providers; cable companies and telephone carriers will be reduced to the role of network providers. Libraries too are expected to take on a different role: Rather than being a central storehouse for information, libraries will function more as rapid access mechanisms to geographically distributed multimedia databases.

The practical realization of personalized multimedia services poses a number of interesting and challenging problems in the areas of multimedia content-extraction, database organization, information retrieval, and caching. Research in these areas is, relatively speaking, in the preliminary stage.

Work is in progress to develop information analysis and feature-extraction methods for content-based multimedia retrieval, that will form the basis of personalized service agents. Efforts are also underway to integrate these content-based retrieval techniques with adaptive, connectionist approaches that have been developed for retrieval of text information from a corpus in which various documents have little structure in common. In conjunction with researchers in library science, visual arts, music, medicine, and psychophysics, researchers are investigating expanded applications of personalized multimedia services in a broad range of areas.

In short, rapid advances in the practical realization of multimedia on demand heralds the arrival of an unprecedented information epoch.

References

[1] —– "Request for Information (RFI) on Digital Media Servers" Cable Labs Inc., June 1994.

[2] *Statistical Abstract of the United States.* U.S. Bureau of Census, 113th ed. 1993.

[3] *MAGIC™ Media Server: A Scalable and Cost Effective Video Server*, Sarnoff Real Time Corporation, Princeton, NJ 08543.

[4] —-, "Storage Concepts, Concept 810-SW real-time RAID," Product Description, Storage Concepts, Inc. Irvine, CA.

[5] —-, "Symmetric Multiprocessing Systems," Technical Report, Silicon Graphics Computer Systems, Mountain View, California.

[6] 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.

[7] 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.*

[8] 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.

[9] 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.

[10] Chang, Ed, and Zakhor, A., "Admissions Control and Data Placement for VBR Video Servers," Proceedings of the First International Conference on Image Processing, November, 1994.

[11] 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.

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

[13] 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.

[14] Chervenak, A, "Tertiary Storage: An Evaluation of New Applications," PhD dissertation,port, Department of Computer Science, University of California, Berkeley, 1995.

[15] Cranor, C., and Parulkar, G., "Universal Continuous Media I/O: De-

sign and Implementation," Washington University Department of Computer Science, Technical Report 94-34, December 1994.

[16] Cranor, C., and Parulkar, G., "Design of Universal Continuous Media I/O," *Proceedings of the 5th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '95),* pp 83-86, April 1995.

[17] Dittia, Z., Cox., J., and Parulkar, G., "Catching up with the networks: Host I/O at gigabit rates," Technical Report WUCS-94-11, Department of Computer Science, Washington University in St. Louis, Apr. 1994.

[18] Dittia, Z., Parulkar, G., and Cox, J., "The APIC Approach to High Performance Network Interface Design: Protected DMA and Other Techniques," Technical Report WUCS-96-12, Department of Computer Science, Washington University in St. Louis, February 1996.

[19] Gopalakrishnan, R., and Parulkar, G.M., "A Framework for QoS Guarantees for Multimedia Applications within an End-system," *Swiss German Computer Science Society Conf.*, 1995.

[20] Gopalakrishnan, R., and Parulkar, G.M., "A Real-time Upcall Facility for Protocol Processing with QOS Guarantees," (Poster) *ACM Symposium on Operating Systems Principles (SOSP)*, Copper Mountain, Colorado, Dec. 1995.

[21] Gopalakrishnan, R., and Parulkar, G.M., "Real time Upcalls," Tech. Rep. WUCS-95-06, Washington Univ. Mar 95.

[22] Hsieh, J., et al., "Performance of a Mass Storage System for Video-On-Demand," *Proceedings of IEEE INFOCOM'95*, pp. 771-778, April, 1995.

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

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

[25] Jessel, A., H., "Cable Ready: The High Appeal for Interactive Services," *Broadcasting & Cable*, May 23, 1994.

[26] Keeton, K., and Katz, R., "The Evaluation of Video Layout Strate-

gies 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.

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

[28] Nussbaumer, J., Patel, B., Schaffa, F., and Sterbenz, J., P., G., "Networking Requirements for Interactive Video on Demand," *IEEE Transactions of Selected Areas in Communication*, Jan. 95.

[29] Papadimitriou, C., and Steiglitz, K., "Combinatorial Optimization: Algorithms and Complexity," *Prentice Hall, Inc.*, 1982.

[30] Papadimitriou, C., H., Ramanathan., S., and Rangan, P., V., "Information Caching for Delivery of Personalized Video Programs on Home Entertainments Channels," *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, Boston, May 1994.

[31] Papadimitriou, C., and Rangan, V., "————", *U.S. Patent Pending.*

[32] Patteson, D., et al., "A Case for Redundant Arrays of Inexpensive Disks (RAID)," Proceedings of the 1988 ACM Conference on Management of Data (SIGMOD), Chicago IL, pp. 109-116, June 1988.

[33] Pugh, W., and Boyer, G., "Broadband Access: Comparing Alternatives," *IEEE Communications Magazine*, pp. 34-46, Aug. 1995.

[34] 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.

[35] Rangan, V., and Vin, H., "Efficient Storage Techniques for Digital Continuous Media," *IEEE Transactions on Knowledge and Data Engineering*, August, 1993.

[36] Ramanathan, S., and Rangan, V., "System Architectures for Personalized Multimedia Services," *IEEE Multimedia*, 1(1), pp. 37-46, Feb. 1994.

[37] Turner, J., "An Optimal Nonblocking Multicast Virtual Circuit Switch," Proceedings of IEEE INFOCOM94, Vol. 1, pp. 298-305, June, 1994.

[38] Turner, J., "A Gigabit Multicast Switch: System Architecture Document," Applied Research Labora-

tory, Washington University in St. Louis, Feb., 1994.

[39] Venkatramani, C., and Chiueh, T., "Survey of Near-Line Storage Technologies: Devices and Systems," Experimental Computer Systems Laboratory, Technical Report #2, Department of Computer Science, SUNY Stony Brook, NY, Oct. 1993.