Yima: A Second Generation of Continuous Media Servers *

Cyrus Shahabi, Roger Zimmermann, Kun Fu, Didi Yao
Integrated Media Systems Center
Department of Computer Science
University of Southern California
Los Angeles, California 90089–0781

Abstract

We report on the design, implementation and evaluation of a scalable real-time streaming architecture, termed Yima, that enable applications such as video-on-demand and distance learning on a large scale. While Yima incorporates lessons learned from first generation research prototypes, it also complies with industry standards in content format (MP4) and communication protocol (RTP/RTSP). Meanwhile, it distinguishes itself from both research prototypes and commercial products because it incorporates our recent research and it pushes the industry envelope by supporting HDTV and multi-channel panoramic clients. This article describes the principles of existing approaches (both research and commercial) and how Yima improves upon them through our research. We also included some detailed information on how we tweaked available hardware and software to achieve certain objectives (e.g., playback of HDTV streams on an HDTV monitor). We describe three major contributions of Yima. First, we show how we have reduced inter-nodal data exchange and hence improved the scalability of Yima using a *bipartite* design. We report our comparison results of two such designs, both implemented in a real-world experimental setup. Second, we explain two alternative approaches to handling variable-bit-rate (VBR) video. Finally, in order to recover from RTP's potential packet-loss, we report on our observations in integrating a selective retransmission protocol into Yima's RTP server. Since packet-loss can mainly be observed in a WAN environment, we experimented with streaming MPEG-4 data over the Internet. Yima is operational and supports a variety of display bandwidths from MPEG-4 at 800 Kb/s to five synchronized channels of MPEG-2 at a total of 20 Mb/s.

1 Introduction

In this article, we report on the design, implementation and evaluation of a scalable real-time streaming architecture that would enable several applications such as news-on-demand, distance learning, e-commerce, corporate training, and scientific visualization on a large scale. A growing number of applications store, maintain, and retrieve large volumes of real-time data, where the data are required to be available online. We denote these data types collectively as *"continuous media"*, or CM for short. Continuous media is distinguished from traditional textual and record-based media in two ways. First, the retrieval and display of continuous media are subject to real-time constraints. If the real-time constraints are not satisfied, the display may suffer from disruptions and delays termed $hiccups$. Second, continuous media objects are large in size. A two-hour MPEG-2 video with a 4 Megabit per second (Mb/s) bandwidth requirement is 3.6 Gigabytes

*This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC) and ITR-0082826, and unrestricted cash/equipment gifts from NCR, IBM, Intel and SUN.

(GB) in size. Popular examples of CM are video and audio objects, while less familiar examples are haptic, avatar and application coordination data [SBE+99].

The first research papers on the design of continuous media servers appeared about a decade ago (e.g. [AH91, Pol91]), followed by many papers on this topic during the past decade (here is an example for every year [GC92, RR93, FR94, FD95, CGM96, NMW97, SY98, GT99, LLG00, LWMS01]). Our research during this past decade started in 1993 [GS93] and continued through 2001 [SGC01]. Some of the projects described in these papers resulted in prototype servers, such as Streaming-RAID [TPBG93], Oracle Media Server [LOP94], UMN system [HLD+95], Tiger [BBD+96], Fellini [MNO+96], Mitra [GZS+97] and RIO [MSB97]. These first generation continuous media servers were primarily focused on the design of different proposals of data placement paradigms, buffer management mechanisms, and retrieval scheduling techniques to optimize for throughput and/or startup latency time.

There are two major shortcomings with these research prototypes. First, since they were implemented concurrently during the same time frame, each one of them could not take advantage of the findings and proposed techniques of the other projects. For example, UMN, Fellini and Mitra independently came out with a seemingly identical design for their data placement technique based on round-robin assignments of blocks to disk clusters. While this approach was already superior in throughput to RAID striping (used by Streaming-RAID), it resulted in higher worst case startup latency time. In contrast, RIO's data placement, which is based on random block assignment, is more flexible and results in the same throughput as round-robin with shorter expected startup latency time. On the other hand, UMN's simple scheduling policy resulted in a fairly good performance without complicating the code as opposed to the constrained scheduling policies of Mitra (round-robin), Fellini (cycle-based) and RIO (round-based). We extended UMN's scheduling (to deadline-driven) and adapted the disk cluster (in Mitra's and Fellini's terms) or *logical volume striping* (in UMN's vocabulary) storage design. We extended RIO's random data placement (to pseudo-random placement for easier bookkeeping and storage scale-up) and instead of an expensive shared-memory architecture of UMN (based on SGI's Onyx), we employed a shared-nothing approach.

The second shortcoming is that almost all of these research prototypes were completed before the industry's standardizations for streaming continuous media over the Internet. Hence, each prototype has its own proprietary media content format, client (and codec) implementation and communication/network protocol. As a matter of fact, some of these prototypes did not even focus on these aspects and never reported on their assumed network and client configurations. They mainly assumed a very fast network with constant-bit-rate media types in their corresponding research publications. Practically, these environment assumptions and the specific content type are not realistic.

Several commercial implementations of continuous media servers are now available in the marketplace. We group them into two broad categories: 1) single-node, consumer oriented systems (e.g., low-cost systems serving a limited number of users) and 2) multi-node, professional oriented systems (e.g., high-end broadcasting and dedicated video-on-demand systems). Table 1 lists some examples for both types. RealNetworks, Apple Computer, and Microsoft product offerings fit into the first category, while SeaChange and nCUBE offer solutions that are oriented towards the second category. Table 1 is of course a non-exhaustive list summarizing some of the more popular and recognizable industry products.

Commercial systems often use proprietary technology and algorithms, therefore, their design choices and development details are not publicly available and objective comparisons are difficult to achieve[1]. Because these details are not known, it is also unclear as to how much research resulting from academic work have been incorporated into these systems. For example, although a good body of work has been developed on how to configure a video server (e.g., block size, number of disks) given an application's requirements

[1]One notable exception is Apple Computer, Inc., which has published the source code of its Darwin Streaming Server.

Video Server	Yima	RealNetworks, Microsoft, Apple QuickTime,	SeaChange	nCUBE
	Prototype	Consumer Level	Professional Level	
Authoring Suite		Yes	Yes	Yes
MPEG-1 & 2	Yes	Yes[a]	Yes	Yes
MPEG-4	Yes	Yes[b]	Yes	?
HDTV MPEG-2	Yes		?	?
Multi-node Clusters	Yes		Yes	Yes
Multi-node Fault-tolerance[c]	Yes		Yes	Yes
Synchronized streams[d]	Yes			
Selective Retransmissions	Yes	Yes	?	?
Hardware	Commodity PC	Commodity PC	Commodity PC	Proprietary Hypercube
Software	Linux	Windows NT / Mac[e]	Windows NT	Proprietary

[a] These systems support many other codecs that are commonly used in PC/Mac environments.

[b] Supported in Windows Media Player.

[c] Multi-node Fault-Tolerance is defined as the capability of one node taking over from another node without replicating the data.

[d] By synchronized streams we mean the simultaneous playback of multiple independent audio and video streams for, say, panoramic systems.

[e] RealSystem Producer products are also available for several Unix variants.

Table 1: A comparison of continuous-media servers. Note that Yima is a prototype system and does not achieve the refinement of the commercial solutions. However, we use it to demonstrate several advanced concepts.

(e.g., tolerable latency, required throughput), as reviewed in [GS99], there is no indication that any of the commercial products utilize these results. However, those details that we are aware of are referred to throughout this paper when comparing them with the design choices of Yima.

To meet the objectives of an NSF ITR grant and an NSF engineering research center (Integrated Media Systems Center)[2], we needed to utilize and extend a working continuous media server to incorporate our recent results, to enable global distribution and to support other media types such as haptic and avatar data. We did a study on both commercial and research servers and due to the above mentioned limitations, we could use neither type the servers. Therefore, we decided to design and develop a second generation continuous media server, termed Yima[3]. Yima is unique because of the following:

- It incorporates and combines useful lessons learned from first generation research prototypes. Yima utilizes: 1) random data placement (vs. constrained placement), 2) non-deterministic scheduling (vs. restricted time-period approach), 3) simple scan or elevator algorithms at the disk level (vs. GSS [YCK93] or REBECA [GKS95]), and 4) both variable and constant bit rate media types. The main objective was *not* to complicate the code in order to squeeze the last 5% of disk bandwidth, but instead to emphasize on the flexibility and extensibility of the code to enable several key benefits as described later. This design choice is especially consistent with the current trend of increase in disk bandwidth [Gro97].

- It is designed to incorporate our own recent research results. In particular, Yima is designed as a completely distributed system (with no single point of failure or bottleneck) as well as separating physical disks (used to store the data) from the concept of logical disks (used for retrieval scheduling)

[2] See http://dimlab.usc.edu/Report/nsf-itr.html and http://imsc.usc.edu/ respectively.

[3] Yima in ancient Iranian religion, is the first man, the progenitor of the human race, and son of the sun.

to support fault tolerance [ZG00] and heterogeneous disk subsystems[4] [ZG97]. Yima includes a method to reorganize data blocks in real-time for online adding/removing disk drives [GSYZ00]. Also included are a flexible rate-control mechanism between clients and the server in order to realize the Super-streaming policy [SA00] and techniques to resolve stream contentions at the server for ensuring inter-stream synchronization as proposed in [CGS95].

- It is compatible with industry standards. Content-wise, Yima supports the MPEG-4 file format, MP4, and can stream MPEG-1, MPEG-2, MPEG-4 and Quicktime video formats. It supports the RTP and RTSP communication standards as described in Sec. 5. The clients can be off-the-shelf QuickTime players as well as our own proprietary clients that handle multi-channel audio and video playback and HDTV displays. Although our proprietary clients can support specialized playback of multiple media types, they still follow communication and format standards.

- It is operational. The system has been implemented and can stream a wide array of media types as we later explain. Yima has been used to stream MPEG-4 data from USC's campus to a residential location connected via ADSL, described in Sec. 6.3. There are plans to co-locate Yima at our industry partner's data center in June 2001 so that we can test multiple high-resolution streams to clients on our campus at up to 60 Mb/s through Internet2.

The preceding points make up the core of our contributions which we will elaborate in greater detail and show their effectiveness through a series of comparative experiments. Several interesting innovations of Yima are worth noting: 1) complete distribution where all nodes run identical software and single points of failure are eliminated, 2) efficient, online scalability of disks where disks can be added or removed without stream interruption, 3) synchronization of several $independent$ streams of audio and/or video within one-fiftieth of a millisecond and 4) smooth, client-dictated transmission rate control.

We will describe our overall system architecture in Sec. 2 including an overview of system components and connectivity. In Sec. 3, we describe the internals of our server design including several comparisons between alternative approaches. We explain why we adopted the Yima-2 design over the Yima-1 design. Next, in Sec. 4, we describe the various types of clients we have developed to support MPEG-2, MPEG-4, HDTV and stream synchronized media delivery. The communications network and protocols are described in Sec. 5 and experiments justifying our design choices and contributions are reported in Sec. 6. Finally, in Sec. 7, we summarize this paper and layout our future plans for our work in this area.

2 System Architecture

Fig. 1 shows the overall system architecture of Yima. Our implementation emphasizes the use of low-cost, off-the-shelf, commodity hardware components for the complete end-to-end system. In our prototype implementation the server consists of a 4-way cluster of rack-mountable Dell™ PowerEdge™ 1550 Pentium III 866 MHz PCs with 256 MB of memory running Red Hat Linux 7.0. The media data are stored on four 18 GB Seagate Cheetah hard disk drives that are connected to the server nodes via Ultra160 SCSI channels. The software architecture of the server nodes is discussed in detail in Sec. 3.

The nodes in the cluster communicate with each other and send the media data via multiple 100 Mb/s Fast Ethernet connections. Each server is attached to a local Cabletron 6000 switch with either one or two Fast Ethernet lines. The local switch is connected to both a WAN backbone (to serve distant clients) and a LAN environment with local clients. Choosing an IP based network keeps the per-port equipment cost

[4]Note that our concept of *logical* disks is very different from the concept of logical volume striping used in the UMN system.

Client Media Type Support				
Media Type	DivX;-) MPEG-4	MPEG-2 + Dolby Digital	MPEG-2 HD	MPEG-1 & 2
Decoder	Software	Creative Dxr2 DVD	Software	Vela Research CineCast
Channels	1 video + 2 audio	1 video + 5.1 audio	1 video	4 video + 8 audio
OS	Linux (RH 6.2)	Linux (RH 6.2)	Linux (RH 6.2)	Windows NT 4.0
Min. CPU Speed	500 MHz	200 MHz	$>$ 1 GHz	400 MHz
Video Resolution	720×480	720×480	1920×1080	720×480 each
Audio Encoding	MP3	Dolby AC-3	N/A[a]	MPEG-1 & 2
Delivery Rate	1 Mb/s	6-8 Mb/s	19.4 Mb/s	4×5 Mb/s

[a] We currently do not decode the audio to save CPU cycles and to increase the speed of the video. This will change in the future.

Table 2: Yima client media type support. All clients are currently implemented on standard Pentium III PC platforms but could also be ported to digital set-top boxes.

low and is immediately compatible with the public Internet. In addition, our server nodes can be deployed within our LAN and do not need to be co-located.

The clients are again based on the commodity PC platform. The Yima client software (*Yima Presentation Player*) runs on either Red Hat Linux 6.2 or Windows NT 4.0. It is structured into several components and only one of them interfaces with the actual media decoder, see Fig. 6. This allows us to plug-in multiple software and hardware decoders and hence support various media types. Table 2 lists the different media types that Yima currently recognizes. Note that in addition to the standard MPEG-1, MPEG-2, and MPEG-4 media types, Yima also recognizes the high-definition TV (HDTV) formats as defined by ATSC[5]. Sec. 4.3 details some of the challenges that we encountered while implementing HD MPEG-2 support.

A second unusual media type is *panoramic video* with *10.2 channel audio*. Panoramic video consists of multiple independent, but tightly synchronized, NTSC-quality video channels that must be combined into a seamless, 360-degree video "cylinder" at the client location. Panoramic video may be acquired with a camera such as the FullView™ model from Panoram Technologies. Extending the visual field requires that the aural presentation be improved as well. The combination of 10.2 channels of audio and panoramic video is the next step in audio-visual fidelity.

Using two clients, each equipped with a 4-channel CineCast™ MPEG-2 decoding unit from Vela Research, we are able to deliver 10.2 channels of synchronized audio and 5 channels of synchronized video from our Yima server. These streams are rendered through twelve speakers and either a head-mounted or wide-angle display. This setup demonstrates the multi-stream synchronization capabilities of Yima. Note that the choice of five channels of video and twelve channels of audio was imposed by the available media content and is not an inherent limitation in the Yima design. Furthermore, the streams may be delivered from one, two, or more Yima servers residing in different locations. The final, fine-grained synchronization is achieved at the client-side. We further elaborate on this setup in Sec. 4.4.

3 Server Design Challenges

An important component of delivering isochronous multimedia over IP networks to end users and applications is the careful design of a multimedia storage server. The task of such a server is twofold: (1) it needs to efficiently store the data and (2) it must schedule the retrieval and delivery of the data precisely before it is transmitted over the network. We investigate two design approaches for our Yima CM server: *master-slave*

[5] Advanced Television Systems Committee, http://www.atsc.org.

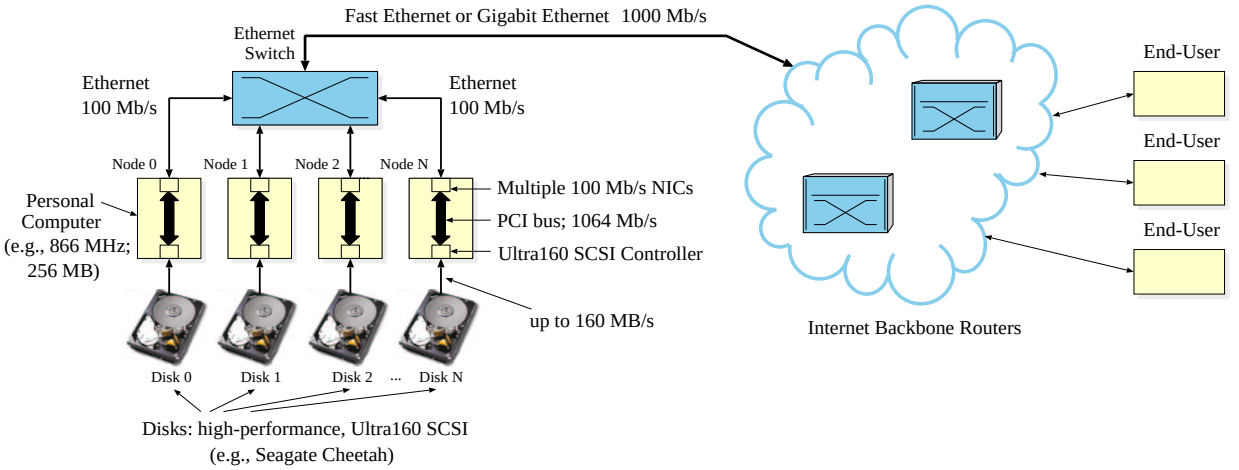

Figure 1: The Yima multi-node hardware architecture. Each node is based on a standard PC and connects to one or more disk drives and the network.

(Yima-1) and *bipartite* (Yima-2). One of the main differences between the two is the logical interconnection topology between the nodes in a cluster. We start by describing the common features for both server designs in Secs. 3.1 and 3.2. Then the specifics of each of the master-slave and bipartite servers are described in Sec. 3.3.

3.1 Data Placement and Scheduling

Magnetic disk drives have established themselves as the storage device of choice for CM servers because of their high performance and moderate cost. A single high-end disk, such as the Seagate Cheetah X15, can sustain an average transfer rate of more than 30 MB/s (e.g., sixty 4 Mb/s streams, under ideal conditions). If – for a large-scale server – a higher bandwidth or more storage space are required than a single disk can deliver then disk drives are commonly combined into disk arrays [CCM97]. For load-balancing purposes without requiring data replication a multimedia object X is commonly striped into blocks, e.g., $X_0, X_1, \ldots, X_{n-1}$ across an array [Pol91, TPBG93]. Data striping may be achieved *implicitly* by using a storage system with embedded RAID [PGK88] controller. In this case the CM server software is unaware of the data striping and treats the RAID system as one large disk drive. If the data striping is *explicitly* under the control of the CM server software, then the distribution of data can extend beyond a single machine. This allows a system to scale to very large server clusters. Examples of such architectures are Yima, SeaChange's *MediaCluster*, and nCUBE's *n4 Streaming Media System* (see Table 1).

Both, the display time of a block and its transfer time from the disk are a function of the display requirements of an object and the transfer rate of the disk, respectively. A multimedia object may either require a constant bit-rate (CBR) or a variable bit-rate (VBR) for a smooth display. VBR encoding generally results in a superior visual quality as compared with CBR for the same object size, because bits can be allocated to high-complexity scenes rather than being spread out evenly. However, the bursty nature of VBR media imposes additional challenges for the data scheduling and transmission mechanisms. A CM server should be designed to handle both types of rates. Many of today's popular compression algorithms, e.g., MPEG-4, produce VBR streams. Fig. 2 shows the display bandwidth requirement of a 10-minute segment for the movie "Saving Private Ryan."

The blocks of VBR multimedia objects can be stored with two fundamental approaches: (1) the size of

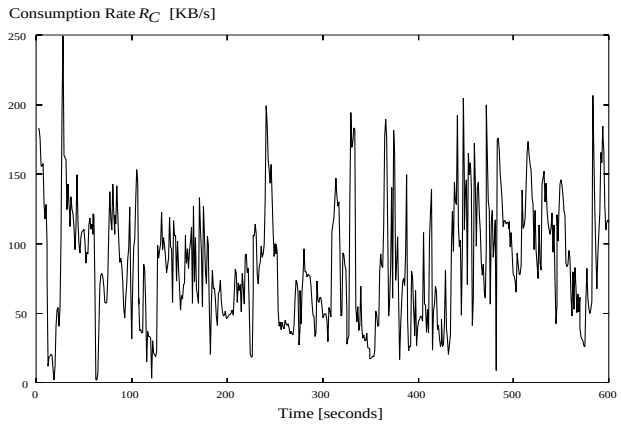

Figure 2: Variable consumption rate of a 10-minute segment of a typical MPEG-4 movie "Saving Private Ryan."

each block is varied to keep the display time per block constant, or (2) the size of each block is constant which results in a variable display time per block. The first approach simplifies the retrieval scheduling but the storage system becomes more complex. For the second approach the tradeoffs are reversed. The Yima framework is based on constant block sizes and our solution to the more complex VBR scheduling and delivery is presented in Sec. 4.1.

There are two basic techniques to assign the data blocks to the magnetic disk drives that form the storage system: in a *round-robin* sequence [BGMJ94], or in a *random* manner [SM98]. Traditionally, the round-robin placement utilizes a cycle-based approach to scheduling of resources to guarantee a continuous display, while the random placement utilizes a deadline-driven approach. In general, the round-robin/cycle-based approach provides high throughput with little wasted bandwidth for video objects that are retrieved sequentially (e.g., a feature length movie). Block retrievals can be scheduled in advance by employing optimized disk scheduling algorithms (such as *elevator* [SCO90]) during each cycle. Furthermore, the load imposed by a display is distributed evenly across all disks. However, the initial startup latency for an object might be large under heavy load because the disk on which the starting block of the object resides might be busy for several cycles. The random/deadline-driven approach allows for fewer optimizations to be applied, potentially resulting in less throughput. However there are several benefits that outweigh this drawback such as 1) support for multiple delivery rates with a single server block size, 2) simpler design of the scheduler, 3) support for interactive applications due to short startup latencies and stream switching, and 4) automatic achievement of the average transfer rate with disks that are multi-zoned.

One potential disadvantage of random data placement is the need for a large amount of meta-data: the location of each block X_i must be stored and managed in a centralized repository (e.g., tuples of the form $\langle node_x, disk_y \rangle$). Yima avoids this overhead by utilizing a *pseudo-random* block placement. With random number generators, a seed value initiates a sequence of random numbers. Such a sequence is pseudo-random because it can be reproduced if the same seed value is used. By placing blocks in a pseudo-random fashion, the next block in a sequence of blocks can always be found using the pseudo-random number generator and the appropriate seed for that sequence. Hence, Yima needs to store only the seed for each file object instead of locations for every block.

3.2 Scalability, Heterogeneity, and Fault-Resilience

Scalability is a desired property of any CM server design so that it can easily expand to allow for future growth in user demand or increased application requirements. Scalability is addressed with several techniques for CM servers. First, instead of a single disk, multi-disk arrays are employed. However, if all the disks are connected to a single large computer then the I/O bandwidth constraints of this computer will limit the overall achievable throughput. The architecture of Yima server implementation is illustrated in Fig. 1. The distributed, multi-node design is based on commodity personal computer hardware components. The storage nodes are interconnected via a high-speed network fabric that can be expanded as demand increases. Each node provides high data throughput from and to the network via a Fast Ethernet connection (100 Mb/s, or a Gigabit Ethernet), a high performance disk interface (for example up to 160 MB/s for Ultra160 SCSI), and fast magnetic disk drive arrays. This modular architecture provides a cost-effective and scalable solution in which older PCs can be easily replaced by newer, more advanced PCs. More nodes can be added to support more simultaneous video requests.

Applications, such as video-on-demand, that rely on large-scale CM servers require continuous operation around the clock. To achieve high reliability and availability for all the CM data that are stored in the server, Yima implements a parity-based data redundancy scheme that, in addition to providing fault-tolerance, can also take advantage of a heterogeneous storage subsystem [ZG97, Zim98]. Current disk drives are fairly reliable with a mean time to failure of up to 1,000,000 hours. However, if a large number of disks is aggregated into a CM storage system, the chance that one disk fails increases. For example, assuming that all the disks are independent and that their lifetimes are exponentially distributed [PGK88], then a disk array consisting of 1,000 disks experiences a mean time to data loss (MTTDL) of 1/1,000th of a single disk, equivalent to 1,000 hours or approximately 42 days. The trend for disk devices is of annually increasing performance and storage capacity [Gro97]. Therefore, it is usually more cost-effective to replace a failed disk with a newer model. Moreover, in the fast-paced disk industry, newer model disks are added to systems requiring more storage capacity because the original disks are simply unavailable. For these reasons heterogeneous storage systems are a common occurrence and Yima manages these heterogeneous resources intelligently to maximize their utilization and to guarantee jitter-free real-time delivery. Yima manages heterogeneous storage systems with a technique termed *Disk Merging* [ZG97]. This technique presents a virtual view of logical disks on top of the actual physical storage system. The application layers of a system can now assume a uniform characteristic for all the logical disks. As a result of this abstraction, conventional scheduling and data placement algorithms as described previously can be applied.

Data Reorganization An important goal for any computer cluster is to achieve a balanced load distribution across all the nodes. Both round-robin and random data placement techniques distribute data retrievals evenly across all disk drives over time. A new challenge arises when additional nodes are added to a server. If the existing data are not redistributed then the system evolves to become a *partitioned* server where each individual retrieval request will impose a load on only part of the cluster. One might try to even out the load by storing new media files only on the added nodes (or at least skewed towards these nodes), but because future client retrieval patterns are usually not precisely known this solution becomes ineffective.

Redistributing the data will once again yield a balanced load on all nodes. Reorganizing blocks placed in a random manner requires much less overhead when compared to redistributing blocks placed using a constrained placement technique. For example, with round-robin striping, when adding or removing a disk, almost all the data blocks need to be relocated. Instead, with a randomized placement, only a fraction of all data blocks need to be moved. For example, increasing a storage system from four to five disks requires that 20% of the data from each disk be moved to the new disk. For disk removal, only the data blocks on the

disk that is scheduled for removal must be relocated. Redistribution with random placement must ensure that data blocks are still randomly placed after disk scaling (adding or removing disks) to preserve the load balance.

As described in Sec. 3.1, we use a pseudo-random number generator to place data blocks across all the disks of Yima. If the number of disks does not change, then we are able to reproduce the random location of each block. However, upon disk scaling, the number of disks changes and some blocks need to be moved to different disks in order to maintain a balanced load. Blocks that are now on different disks cannot be located using the previous random number sequence, instead a new random sequence must be derived from the previous one. In Yima we use a composition of random functions to determine this new sequence. Our approach, termed SCAling Disks for Data Arranged Randomly or SCADDAR, preserves the pseudo-random properties of the sequence and results in minimal block movements after disk scaling while it computes the new locations for data blocks with little overhead [GSYZ00]. This algorithm can support scaling of disks while Yima is online.

3.3 Yima Multi-Node Server Architecture

Yima servers are built from clusters of multiple server CPUs which we refer to as *nodes*. Each node connects to a common network fabric via Fast or Gigabit Ethernet. A distributed file system provides a complete view of all the data on each node without the need to replicate individual data blocks (except as required for fault-tolerance [ZG97]). A Yima cluster can run in one of two possible modes: master-slave or bipartite. We will now describe each one in turn.

Master-Slave Design (Yima-1) One technique to enable a server application to access the storage resources located on multiple nodes is to introduce a distributed file system. With this design, an application running on a specific node operates on all its files through the distributed file system layer, which transparently provides access to local and remote data. If remote data are read or written, then the distributed file system forwards the data via a network protocol to the corresponding node.

The Yima-1 software consists of two components: the Yima-1 high-performance distributed file system and the Yima-1 media streaming server as shown in Fig. 3. The distributed file system is actually comprised of multiple File I/O modules located on each node. The media streaming server itself is composed of a scheduler, a Real-Time Streaming Protocol (RTSP [SRL98]) module and a Real-Time Protocol (RTP [SCFJ96]) module, see Fig. 6. Each Yima-1 node may either run only the distributed file system, in which case the Yima-1 media streaming server is inactive, or both components. A node running only the file system is said to have *slave* capabilities while a node running both components is said to have *master* and *slave* capabilities.

A master server node is the point of contact for a client during a session. A session is defined as a complete RTSP transaction for a continuous media stream, starting with the `DESCRIBE` and `PLAY` commands and ending with a `TEARDOWN` command. When a client requests a data stream using RTSP, it is directed to a master server node. If multiple master nodes exist in the cluster, then this assignment is decided based on a round-robin domain name service (RR-DNS) or a load-balancing switch. Each master node retrieves and streams the corresponding data to its various clients. To satisfy the real-time constraints of each data stream, the distributed file system is optimized for fast block access and transfer between nodes. Furthermore, it keeps track of the locations (i.e. disk number) of all data blocks for each media file. This is facilitated by using a pseudo-random number generator that determines the particular disk number that a specific block falls onto based on a seed value as previously described in Sec. 3.1. The master node also must keep configuration files that provide the disk-to-node mapping for all disks. Requests for blocks can then be brokered to correct slave nodes which are connected to the particular disk containing the requested data

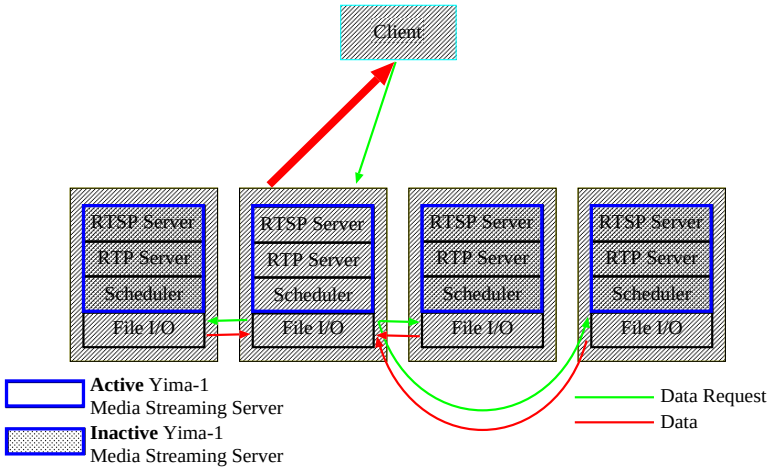

Figure 3: Yima-1 software design. A master node becomes active when clients establish sessions with it. Otherwise master nodes remain inactive.

block.

One major advantage of using a distributed file system, which transparently maps remote data into a local address space, is that applications need not be aware of the distributed nature of the storage system. Hence, even applications that were designed for a single node can to some degree take advantage of a cluster organization. In fact, the Yima-1 media streaming server component is based on the Darwin Streaming Server™ (DSS) project by Apple Computer, Inc. The DSS code assumes that all media data are located in a single, local directory. Enhanced with our distributed file system, multiple copies of the DSS code (each running on its own master node) were able to share the same media data.

With this architecture, the ease of utilizing clustered storage is offset by some major performance problems. The master node can easily become a bottleneck since all transmitted data funnels through it. Even more significantly, the master-slave architecture leads to heavy inter-node traffic which may saturate inter-node links. The outgoing network traffic, the inter-node traffic and the CPU load of the master nodes all increase as the number of sessions increases. Furthermore, a master node potentially becomes a single point of failure such that all sessions are lost if it fails. These drawbacks motivated the design of Yima-2 which addresses these issues to provide a higher performing and more scalable solution.

Finally, we require a bit rate smoothing technique to support the delivery of variable bit rate media. In our implementation of Yima-1, we use a binary variation: a stream is sent at a rate of either R_N or zero megabits per second, where R_N is the transfer rate across a network. R_N is fixed at a value between the average and maximum consumption rate of a stream (the correct value depends on the client buffer size). We denote this as the pause/resume transmission scheme since the client issues pause and resume commands to the server depending on how full the client buffer is. A disadvantage of this technique is that it may temporarily require a higher link transmission rate than other smoothing techniques. Additionally, its on/off nature can lead to bursty traffic. Advantages include its simple design and that it can be applied dynamically to streams that are in progress without any precomputation other than knowing the maximum consumption rate.

Bipartite Design (Yima-2) The design of Yima-2 addresses almost all of the limitations of Yima-1. It is based on a *bipartite* model. Recall that with the Yima-1 master-slave design, the slave nodes forward all data blocks to the master node where the blocks are placed into the correct order and scheduled for transmission

to the clients.

Hence, from the viewpoint of a single client, the scheduler, the RTSP and the RTP server modules are all centralized on a single master node. Yima-2 expands on the decentralization by keeping only the RTSP module centralized (again from the viewpoint of a single client) and parallelizing the scheduling and RTP functions as shown in Fig. 4. Hence, every node retrieves, schedules, and sends data blocks that are stored locally directly to the requesting client, thereby eliminating the bottleneck caused by routing all data through a single node. The elimination of the master node bottleneck and the distribution of the scheduler reduces the inter-node traffic to only control related messages, which is orders of magnitude less than the streaming data traffic. The term "bipartite" relates to the two groups, a server group and a client group (in the general case of multiple clients), such that data flows only between the groups and not between nodes of a group. Although the advantages of the bipartite design are clear, its realization imposes several new challenges. First, since clients are receiving data from multiple servers, a global order of all packets per session needs to be imposed and communication between the client and servers needs to be carefully designed, see Sec. 5. Second, an RTSP server node still needs to be maintained for client requests. The original DSS code component of our Yima-1 media streaming server clearly did not fit the Yima-2 decentralized model and it needed to be replaced with a distributed scheduler and RTP server. Lastly, a flow control mechanism is needed to prevent client buffer overflow or starvation.

With Yima-2, each client maintains contact with one RTSP module for the duration of a session for control related information (such as PAUSE and RESUME commands). For load-balancing purposes each server node may run an RTSP module. For each client, the decision of which RTSP server to contact can be based on either a round-robin DNS or a load-balancing switch. However, contrary to the Yima-1 design, if an RTSP server fails, sessions are not lost — instead they are reassigned to another RTSP server and the delivery of data is not interrupted.

With Yima-2, we replaced the DSS-based server component with our own light-weight code that accomplishes the distributed scheduling, handles multiple concurrent client sessions (via the RTSP protocol), and transmits media data as RTP streams. For the storage of media blocks we adapted the MPEG-4 *file format* as specified in MPEG-4 Version 2 [MP4]. The MP4 file format is a flexible container format that is based on the QuickTime™ file format from Apple Computer, Inc. Its structure is object-oriented and stores both a hierarchy of metadata (also called *atoms*) and the media data itself. Metadata known as *hint tracks* provide instructions on how to packetize and deliver the media data according to, for example, the RTP protocol. In Yima-2 we expanded on the MP4 format by allowing compressed media data other than MPEG-4 to be encapsulated (e.g., MPEG-2). This gives us the flexibility of easily delivering any data type by augmenting it with MPEG-4 conforming metadata information while still being compatible with the MP4 industry standard. Our server is multi-threaded with each thread having a specific task as shown in Fig. 4. We are using the thread library *Qpthread*, that handles message queues and thread pools. The result is a light-weight, easily extendible server framework.

In order to avoid bursty traffic caused by the pause/resume transmission scheme of Yima-1 (as described in further detail in Sec. 4.1), the client controls the data transmission rate in Yima-2. By periodically sending slowdown or speedup commands to the Yima-2 server, the client can receive a smooth flow of data by monitoring the amount of data in its buffer. If the amount buffer data decreases (increases), the client will issue speedup (slowdown) requests. Thus, the amount of buffer data can remain close to constant. This mechanism will complicate the server scheduler logic, but bursty traffic is greatly reduced as shown in Sec. 6.2.

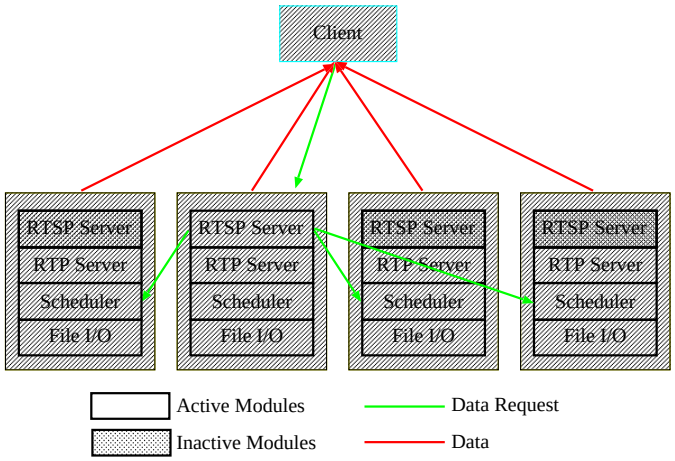

Figure 4: Yima-2 software design. An RTSP module becomes active when clients establish sessions with it. Otherwise the RTSP module remains inactive.

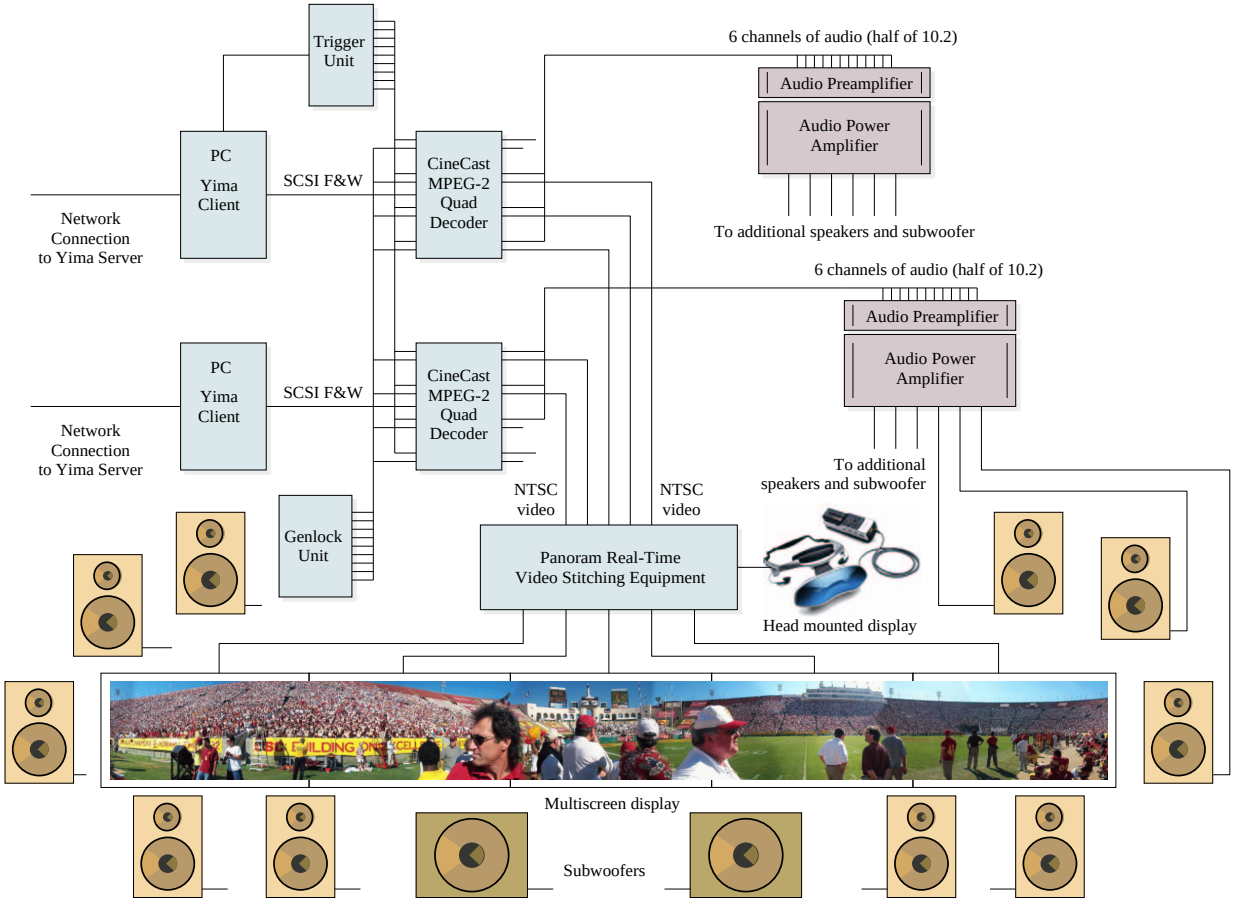

Figure 5: Panoramic video and 10.2 channel audio playback system block diagram. Two Yima clients are used to render 5 channels of synchronized video (i.e., a mosaic of 3000×480 pixels) and 10.2 channels of synchronized audio (the 0.2 refers to two low-frequency channels, e.g., subwoofers).

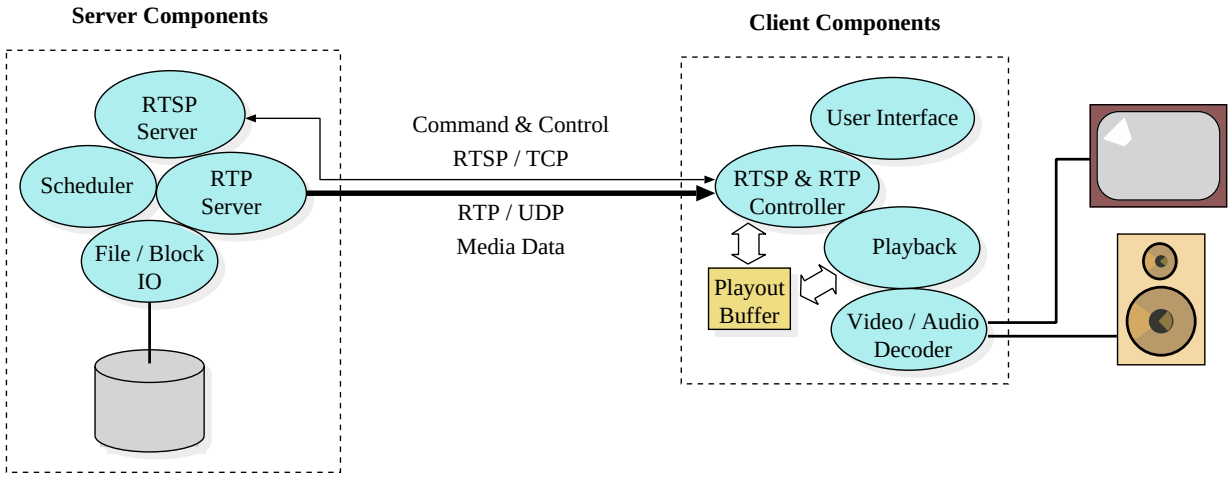

Figure 6: Common components in a CM server and client that adhere to the industry standard RTSP and RTP protocols. Additional components may be present for content, user and billing management that are not shown in this figure.

4 Client Systems

We have built a client application called the *Yima Presentation Player* to demonstrate and experiment with our Yima server. The Yima player can display a variety of media types and is available for both Linux and Windows platforms. Clients receive streams via standard RTSP and RTP communications, which are detailed in Sec. 5. In the following sections we describe the client design and how it handles various media types. We start with an overview of our playout buffer management technique that is common across all media types. In Sec. 4.3, we report on the current state of our HDTV client. Finally, we describe how we can achieve stream synchronization across multiple clients required by certain applications in Sec. 4.4.

4.1 Client Buffer Management

A circular buffer in the Yima Presentation Player reassembles media streams from RTP packets that are received from the server nodes. Recall that Yima supports variable bit rate media. Numerous techniques have been proposed in the literature to transmit VBR media (for an overview see [AG98]). Most of the techniques *smooth* the *consumption rate* R_C by approximating it with a number of constant rate segments. Such smoothing algorithms implemented at the server side need complete knowledge of R_C as a function of time to compute each segment length and its appropriate constant bit rate. Segment boundaries are determined based on the following two assumptions: (1) the data transfer takes a fixed amount of time and (2) no data are lost, hence the client buffer fullness can be computed. Neither of these two assumptions is true in real world networks, where packet delays and drops are common.

To work in a dynamic environment we based our buffer management techniques on a flow control mechanism. A circular buffer of size B accumulates the media data and keeps track of two watermarks: buffer overflow WM_O and buffer underflow WM_U. Data are consumed from the same buffer by the decoding thread. We introduce two schemes termed the pause/resume scheme and the Δp scheme and describe them as follows.

Pause/Resume If the data in the buffer reaches WM_O, the data flow from the server is paused. The playback will continue to consume media data from buffer. When the data in the buffer reaches the underflow watermark WM_U, the delivery of the stream is resumed from the server. If the delivery rate (R_N) of the data is set correctly then the buffer will not underflow while the stream is resumed.

In an ideal case the server could be paused and resumed instantaneously and the watermarks could be set as follows: $WM_O = B$ and $WM_U = 0$, where B is the buffer size. However, in a real world implementation the processing of pause and resume results in some delays (Yima uses the RTSP commands PAUSE and PLAY for these purposes). The accurate values of WM_O and WM_U are calculated as follows. After a pause request is sent out, the server is still sending data during a short time T_d, where T_d is the one-way delivery time from server to client. The data accumulated during T_d is $(R_N - R_C) \times T_d$. Hence, the buffer overflow watermark must be set to

$$WM_O \leq B - (R_N - R_C) \times T_d \tag{1}$$

to avoid a buffer overrun. Similarly, the buffer underflow watermark must be set to

$$WM_U \geq R_C \times T_d \tag{2}$$

The delay time T_d depends on the network delay between the client and the server as well as the server request processing time and must be empirically obtained. Finally, the following condition must hold: $WM_O \geq WM_U$. If this condition is not met then the buffer size B is not large enough for this operating environment. Although the pause/resume technique is a simple, clean design, if pausing and resuming actions coincide across multiple sessions, then bursty traffic will become a noticeable effect.

Client-controlled Δp Δp is the inter-packet delivery time used by the schedulers to schedule the transmission of packets to the client. A global Δp is maintained across all the schedulers of all the server nodes. Schedulers use the Network Time Protocol (NTP) as the common time across nodes in order to know when to send packets. Using NTP and the timestamp of each packet, server nodes send packets in order at Δp time intervals. The Δp delivery rate at which packets are sent should be adjusted to match the client-side variable consumption rate as closely as possible. This adjustment is fine-tuned by the client by updating the server with new Δp values based on the amount of its buffer data. The client-side buffer is used to smooth out any fluctuations in network traffic which might lead to late packets. Thus, the client adjusts the amount of data within this buffer by issuing speedup and slowdown commands to the server to keep the amount of data between predetermined overflow and underflow watermarks. Ideally, by fine-tuning Δp, the client should be able to keep a constant amount of buffer data throughout the session. Speedup commands decrease the Δp value while slowdown commands increase the Δp value. The initial default Δp value needs to be small enough to quickly fill the buffer to a desired amount. Changing the Δp is the control mechanism which affects the amount of data within the buffer. This control mechanism can be used to enable different types of policies such as a policy dictating the fastest possible delivery rate given a specific server load and network load.

To compute the amount of Δp fluctuation, the client issues commands to speedup or slowdown the data stream as shown in Fig. 7. Several watermark thresholds are used to fine-tune the streaming rate. As the amount of client buffer data approaches the overflow watermark, the client will increase Δp first by 10% and eventually by 20% if the amount continues to increase. Similarly, as the amount of data approaches zero due to a high consumption rate, the client will speedup the delivery rate by decreasing Δp first by 10% and then by 20%. As in Yima-1, if the amount of data reaches either the overflow or underflow watermarks, then PAUSE or RESUME commands are issued, respectively. A RESUME command will set Δp to the initial default value. The percentages of increase and decrease were chosen in accordance to the characteristics of our media. Differing media consumption rates may necessitate a different set of percentages for optimized

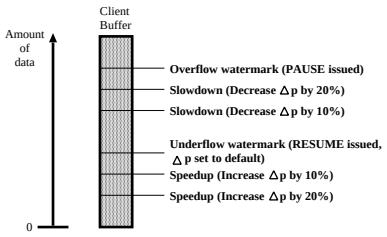

Figure 7: The amount of data in the client buffer triggers the client to issue speedup or slowdown commands if a threshold is crossed.

delivery smoothness. Thus, the client is able to control the delivery rate of received data to achieve smoother delivery and prevent bursty traffic.

4.2 Yima Presentation Player Media Types

We have experimented with a variety of media types for our Yima player. Fig. 6 shows the three-threaded structure of the player. The Playback thread is responsible for interfacing with the actual media decoder. The decoder can either be software or hardware based. Table 2 lists some of decoders that we incorporated.

First, we started with CineCast™ hardware MPEG decoders from Vela Research. They support both MPEG-1 and MPEG-2 video and 2 channel audio. However, content that includes 5.1 channels of audio (i.e., Dolby Digital) is not recognized. To remedy for this situation, we use the open source *ac3dec* audio library to downmix 2 channels from the 5.1 channels of received audio. Hence, the CineCast decoders process the video and the PC sound cards render the 2 channels of audio. However, this solution presented a problem in that the audio and video were not synchronously displayed. Extensive fine tuning was performed to shift the audio to match the video.

Next, we moved to a Dxr2™ PCI card from Creative which can decompress both MPEG-1 and 2 video in hardware. Additionally it decodes MPEG audio and provides a 5.1 channel Dolby Digital audio output terminal. With this solution the multi-channel audio synchronization was accomplished in hardware and the rendering presented no problems.

With the emergence of MPEG-4 we started to experiment with a software decoder called *DivX;-)* [Hib01]. Using this decoder, we can decompress MPEG-4 data entirely in software without any specialized hardware. MPEG-4, a newer standard than MPEG-2, provides a higher compression ratio. A typical 6 Mb/s MPEG-2 media file may only require a 800 Kb/s delivery rate when encoded with MPEG-4. Sec. 6.3 contains details on one of our end-to-end experiments, where we delivered an MPEG-4 video stream at near NTSC quality to a residential client site via an ADSL connection.

4.3 HDTV Client

The streaming of HD content presented several challenges for our Yima implementation. First, HD media require a high transmission bandwidth. For example, a video resolution of 1920×1080 pixels encoded via MPEG-2 results in a data rate of 19.4 Mb/s. This turned out to be less of a problem at the server side because Yima was designed to handle high data rates.

We faced the more intriguing problems at the client-side. Consumer hardware decoders that can be easily

integrated with our software are not yet available[6]. Professional solutions can cost up to tens of thousands of dollars. Hence we decided to integrate a software decoder. We started with a library from the *mpeg2dec* open source project. However, even though we were able to decode our content, achieving the real-time frame rates with HD video was non-trivial. HD media contains up to seven times more pixels per frame than standard NTSC video and hence requires much more CPU cycles to decode than, for example, a DVD movie. On a 933 MHz Pentium III we were able to achieve 13 to 14 frames per second[7]. The software decoder does take advantage of MMX and SSE multimedia instructions, but does not utilize the graphics chip iDCT capabilities. After further optimization to the code we are confident that we will achieve real-time decoding. A software decoder has the advantage of being cheap and easy to disseminate.

An additional problem was introduced with the software decoder: the interfacing with an actual HD television set. Both, resolutions and refresh rates of conventional computer monitors are different from what HD sets can handle. The 4×3 aspect ratio of most computer monitors is different than the 16×9 aspect ratio used with HD content. Also, unlike multi-sync monitors, HD sets usually only synchronize to the specific formats defined in the ATSC standard. We were able to overcome these limitations by configuring the computer graphics card to a non-standard, HD-compatible mode, such as 1280×720. Sometimes even the actual physical video connection can present a problem. Conveniently, our AF3.0HD test monitor from Princeton Graphics did have a 15-pin VGA-type input connector. For other models, which only feature YPbPr input terminals, a converter is needed (e.g., Audio Authority's model 9A60). This solution allowed us to configure the output also for 960×540 progressive display, which the HD set interpreted similarly to 1920×1080 interlaced video. In this lower resolution mode (which is still considerably better than NTSC) we were able to demonstrate full screen MPEG-4 compressed video at just over 2 Mb/s bandwidth, see Fig. 8. We estimate that full-resolution HDTV is hence possible at approximately 8 Mb/s with MPEG-4. An interesting application would be to upgrade the picture quality of the panoramic video described in the next section. The five synchronous video channels would require a total bandwidth of approximately 40 Mb/s and produce an astounding 9000×1000 pixel display.

4.4 Multi-Stream Synchronization

The flow control techniques implemented in the Yima client-server communications protocol allow us to synchronize multiple, independently stored media streams. Fig. 5 illustrates the client configuration for the playback of panoramic, 5 channel video and 10.2 channel audio. The five video channels originate from a 360-degree video camera system such as the FullView™ model from Panoram Technologies. The audio is recorded from multiple microphones. Each video channel is combined with two audio channels and encoded into a standard MPEG-2 program stream[8]. During playback, all of the streams must be rendered in tight synchronization such that the five video frames that correspond to one time instance are accurately combined into a panoramic 3000×480 mosaic every $\frac{1}{30}$ of a second. The real-time seamless frame stitching is achieved with processing equipment from Panoram Technologies. A segment of the resulting panoramic video can either be shown on a wide-screen display or an observer may wear a head-mounted display (HMD). If the HMD is combined with a head tracking device, then the observer can freely "look around" (360 degrees) and experience a much more immersive visual sensation than is possible with a conventional display.

The experience is enhanced with surround audio that is presented phase-accurately and in synchronization with the video. 10.2 channels of immersive sound create a seamless, fully three-dimensional aural environment that preserves the correct spatial sound localization for a participant, even if he or she moves

[6]The Win-HD card from Hauppauge may be a candidate in the future.

[7]With Linux 6.2 and XFree86 4.0.1 on an nVidia Quadro 2 graphics accelerator.

[8]One stream is audio only because we have 5 video and 12 audio channels.

Figure 8: HDTV display connected to an Yima client. The Yima client can decode HD MPEG-2 formats in software up to the full the resolution of 1920×1080i (with some degradation in the frame rate on a 933 MHz PC). The picture shows a quarter-resolution video with 960×540p pixels compressed via MPEG-4.

around in the environment. In our system we output the audio streams into mixing equipment which allows for interactive changes and adjustments. For example, undesirable room characteristics can be compensated before the audio signals are amplified and rendered with high-quality speakers.

Yima achieves such precise playback with two levels of synchronization: (1) coarse-grained via the flow control protocol and (2) fine-grained through hardware support. The flow control protocol allows us to maintain approximately the same amount of data for each MPEG-2 stream in the client buffers, even if there are variations in the bitrate across multiple streams. With this pre-requisite in place, we can lock-step the hardware MPEG decoders to produce frame-accurate output. For this we use multiple CineCast™ MPEG hardware decoders from Vela Research genlocked to one timing signal. The genlock signal assures that the current frame playout starts and stops together in all decoders. However, if one decoder originally started its playback several frames ahead of others, then this difference will actually be "locked in" instead of corrected. For example, if decoder A is three frames ahead of decoder B, then genlocking them together will always keep A three frames ahead of B because the genlock signal only provides synchronization within the current frame (however, there will be no drift). Moreover, lost packets which result in missing frames will affect the synchronization and may lead to drift. In our LAN we do not experience dropped packets, but a solution will eventually be required for WAN and Internet delivery. All streams need to be started precisely at the same time. The CineCast decoders provide an external trigger input for this purpose. In our system we can software-control this trigger to initiate the playback when all channels have sufficient initial data. Using two clients, each of them equipped with a 4-channel CineCast decoder, we are able to render up to eight synchronous streams of MPEG-2 data (i.e., 8 video and 16 audio channels).

5 RTP/UDP and Selective Retransmission

Yima supports the industry standard real-time protocol (RTP) for the delivery of time-sensitive data. Because RTP transmissions are based on the connection-less User Datagram Protocol (UDP), a data packet could arrive out of order at the client or be altogether dropped at some point along the network. The characteristics of real-time data places rigid constraints on packet delivery and may be sensitive to packet loss. If a missing packet does not arrive at the client before it is scheduled to be consumed, then it is considered lost. To achieve the best visual and aural quality possible, we start decoding a movie only after the arrival of an initial amount of data at the client-side buffer.

To reduce the number of RTP data packets lost between the server and client we implemented a selective retransmission protocol [PP96]. This protocol was configured to attempt at most one retransmission of each lost RTP packet only if it seemed likely (based on the round-trip packet delay) that the retransmitted packet would arrive in time for consumption by the decoder. With this technique the *raw* transmission data loss $P_0 = p$ can be reduced to an *effective* data loss of

$$P_{eff} = p \times (q + ((1 - q) \times p)) \tag{3}$$

where p is server-to-client and q is the client-to-server data loss probability. Eq. 3 assumes a different loss rate for both link directions since most broadband technologies (such as ADSL and cable modems) provide asymmetric throughput for upstream and downstream data flow. If the loss probability is identical in both link directions, then Eq. 3 simplifies to

$$P_{eff\_symmetric} = p^2 \times (2 - p) \tag{4}$$

As a result of this technique, the transmission of video/audio streams is more error-resilient for both local or wide area networks (LANs and WANs, e.g., Internet2). We report on the reduction in data loss in an end-to-end experiment in Sec. 6.3.

5.1 RTP/UDP with Multiple Server Nodes and Selective Retransmission

When multiple servers deliver packets which are part of a single stream, as with Yima-2, the following question arises: If a data packet did not arrive, how does the client know which server node attempted to send it? In other words, it is not obvious to which node the client needs to send its retransmission request. There are two solutions to this problem: 1) the client computes which server node to issue the retransmission request to, or 2) the client broadcasts the retransmission request to all the server nodes.

Unicast approach Recall from Sec. 3.1, that the locations of the data blocks can be determined by regenerating a pseudo-random sequence. With the unicast approach, retransmission requests are addressed to a specific server node possessing the requested packet. The client can determine the source of a missing packet if it has all of the following information: the block ID of this packet, the seed of the movie, a mapping of which logical disks lay on which server nodes and the original pseudo-random number generator used to randomly distribute the data blocks. While this approach sends the retransmission request to the correct server node, the amount of meta-data needed by the client leads to greater complexity and book-keeping. Each time disks are added or removed from Yima during disk scaling, the disk-to-node mapping information needs to be passed onto the clients. Logically, the clients should be oblivious to the configurations and components of the file system. Moreover, if the client is low on resources, then the added computation of the packet locations would be unfavorable.

Broadcast approach We have incorporated the broadcast approach into our Yima-2 design. Broadcasting the packet retransmission request to all of the server nodes places little load on the client since no computation to identify a specific server node is necessary. Instead, server nodes receive the request, check whether they hold the packet and either ignore the request or perform a retransmission. Thus the client remains unaware of the server sub-layers. Moreover, most local area networks have LAN IP addresses which can be used to multicast packets to all IP addresses on that LAN. In essence, the client could send a unicast of the retransmission request to the LAN IP address upon which the LAN gateway would broadcast the request to all the server nodes. So with the broadcast model, we separate the client from knowledge of the underlying server file system such as the server disk configurations and the random sequence of the data blocks.

Even though this technique works well in our system, we realize that it may impact the scalability of Yima-2. We plan to investigate other techniques, such as forward error correction (FEC) to find an optimal solution.

6 Experiments

We conducted a series of experiments to verify and evaluate our design choices and to provide evidence of improvements in Yima-2 vs. Yima-1. While the majority of our server and client tests were on our local area network, we also report on the results of a test across the public Internet. All of our experiments were conducted on the same hardware components in order to provide fair comparisons.

6.1 Server Experiments

We made several design choices to improve the overall performance of Yima-2 over Yima-1. Most significantly, we eliminated the inter-node data traffic which limits the number of streams per node, $\mathcal{N}$, as shown in Eq. 5. This equation assumes full-duplex network links with a throughput of R_N each and m nodes. Intuitively, by increasing the number of nodes, the fraction of the data that is local decreases. The second factor $\frac{1}{2-\frac{1}{m}}$ denotes the reduction that is due to the inter-node traffic. For large configurations the throughput is cut in half.

$$\mathcal{N} = \frac{R_N \times m}{R_C + R_C \times (1 - \frac{1}{m})} = \frac{R_N \times m}{R_C} \times \frac{1}{2 - \frac{1}{m}} \tag{5}$$

We tested the Yima-1 and Yima-2 software on a 4-way server configuration as described in Sec. 2. The MPEG-2 encoded test media required 8 Mb/s of bandwidth per stream. In the case of Yima-1, all four nodes were slave nodes and we increased the number of masters from 1 to 4. For Yima-2, we increased the number of participating nodes from 1 to 4, because every node sends out data.

Method of measurement We needed a way to measure the number of sustainable, concurrent clients. For Yima-1, the Darwin Streaming Server component of the Yima-1 media streaming server begins to exhibit a significant drop in throughput when the number of sessions increases to a point where the server bandwidth or the server CPU are maxed out. Because our system bottleneck is in the network, the available bandwidth maxed out before the CPU utilization. Hence, increasing the number of sessions by one should increase the total server throughput by 8 Mb/s. The maximum number of sustainable, concurrent clients is determined when an 8 Mb/s increase in throughput is not observed after increasing the number of sessions by one.

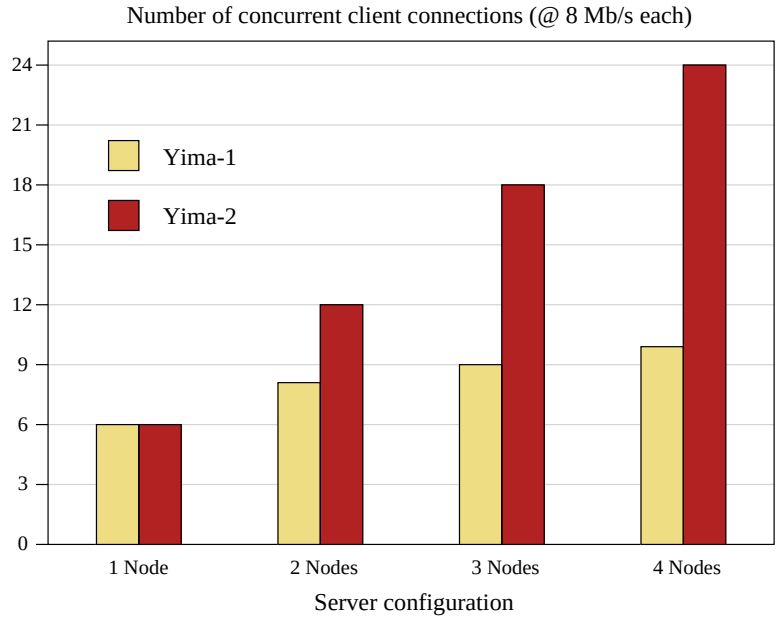

Figure 9: The number of client streams (at 8 Mb/s) that can be supported with the Yima-1 and the Yima-2 architecture as the number of nodes increases. The inter-node data traffic clearly limits the Yima-1 scalability.

For Yima-2, a noticeable increase in hiccups will occur when the number of sessions increases beyond a certain number of sustainable, concurrent sessions. Again, this is the point at which the available network bandwidth begins to max out. To find this threshold, we considered that the maximum number of sessions had been reached when the last session admitted experienced a 2% delay of packets. A delayed packet is defined as a packet which was sent 1 or more seconds too late. We observe that visual and audible quality degrades significantly when the number of delayed packets is $> 2\%$.

A limiting factor to the results which we obtained for our experiments is that the network cards supplied by our server manufacturers seemed to only support a maximum uplink delivery rate of 50 Mb/s instead of the expected 100 Mb/s. However, the results for our experimental sets were all performed on the same hardware components so that the relative performances are compared fairly.

Measurement results The maximum number of sustainable 8 Mb/s data streams for Yima-1 and Yima-2 are shown in Fig. 9. Yima-2 exhibits an almost perfectly linear increase in the number of streams while Yima-1 levels off early. Of course, Yima-2 will also become sub-linear with larger configurations and/or low-bit-rate streams, but we are pleased to see that it scales much better than Yima-1. This expectation of non-linearity is attributed to the increase of inter-nodal control commands. We observe a decrease in the average number of supportable streams per server node as the number of server nodes is increased for Yima-1.

6.2 Client Experiments

We conducted two sets of experiments on our client system. First, we compare the delivery rate of media streams for Yima-1 and Yima-2 over a period of time. Recall from Sec. 4.1 that Yima-1 utilizes the pause/resume method, which leads to bursty traffic, while Yima-2 uses the Δp method, which leads to a smoother delivery rate. The actual consumption rate is compared to the delivery rate using Δp and the similarity between the two verifies that the client attempts to keep a constant amount of data in its buffer.

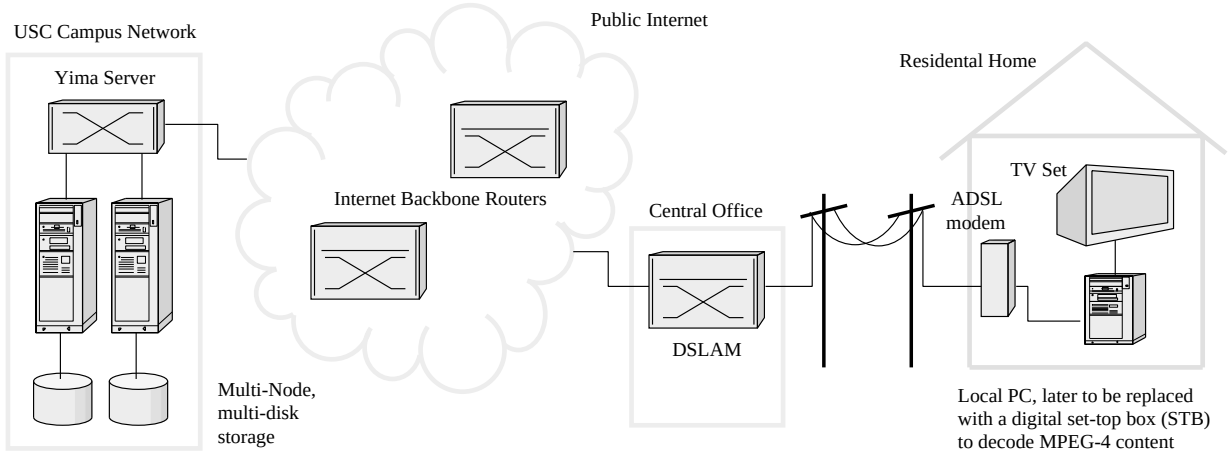

Figure 10: Yima end-to-end configuration.

The closer the amount of buffer data is to becoming constant, the closer the delivery rate will mimic the consumption rate. Keeping the amount of data in the buffer constant also improves client utilization of the buffer. The client can better allocate unused portions of memory for other uses if the amount of used buffer space is known beforehand.

In the second set of experiments, we evaluated the accuracy of our synchronization mechanism for multi-channel clients. With our experimental setup we have been able to achieve frame accurate video playback over an extended period of time. Genlocked frames are played out very precisely with differences of at most a few milliseconds. Audio multi-channel synchronization is actually more critical because even slight time delays between two channels result in phase differences that can shift the position of an instrument from its correct rendering location to the left or right. Our audio content is sampled at either 44,100 or 48,000 Hz. Even a few samples worth of time difference can be noticed by a trained listener. To test our experimental setup we played the same content on combinations of two channels (out of eight possible channels on one CineCast™ MPEG decoder) and recorded the rendered signals as a stereo file. This stereo file was used as input into a program, written in Matlab, to find the sampling difference between the maximum cross-correlations of a 90-second segment of each channel. Correlating all combinations of two channels resulted in either a 0 or 1 sample time difference between channels.

6.3 End-to-End Experiments

There have been a number of reports of trials and deployments of video-on-demand services. However, little to no information is available about the architectural details and performance tradeoffs for these systems. We have used our Yima testbed to investigate the end-to-end performance of near NTSC quality video delivery over IP. Fig. 10 illustrates the experimental setup.

The data sent out from the Yima servers in our lab were transported via the USC campus network to the public Internet. Table 3 shows the data route between one of our test client locations and the Yima servers. The geographical distance between the two end points is approximately 40 kilometers. The client was set up in a residential apartment and linked to the Internet via an ADSL connection. The raw bandwidth achieved end-to-end between the Yima client and servers was approximately 1 Mb/s[9].

Our test media was encoded with an MPEG-4 software codec called “DivX;-)” [Hib01]. The high

[9]The ADSL provider did not guarantee any minimum bandwidth but stated that 1.5 Mb/s will not be exceeded.

Hop #	Router
1	user-2iniv81.dialup.mindspring.com (165.121.125.1)
2	207.69.228.1 (207.69.228.1)
3	s4-1-1.lsanca1-cr1.bbnplanet.net (4.24.24.13)
4	p2-1.lsanca1-ba1.bbnplanet.net (4.24.4.5)
5	p0-0-0.lsanca1-cr3.bbnplanet.net (4.24.4.18)
6	s0.uscisi.bbnplanet.net (4.24.40.14)
7	usc-isi-atm.ln.net (130.152.128.2)
8	rtr43-18-gw.usc.edu (128.125.251.210)
9	rtr-gw-1.usc.edu (128.125.254.1)
10	zanjaan.usc.edu (128.125.163.158)

Table 3: End-to-end route from one Yima client (located in West Covina, California, connected via an ADSL line) to the Yima server (USC campus, Los Angeles). The distance between the two locations is approx. 40 km.

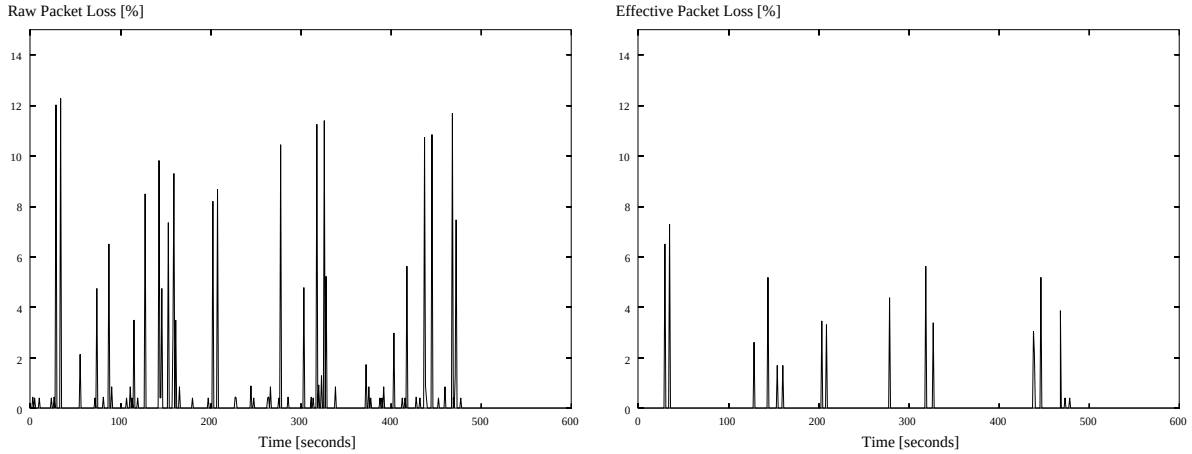

Fig. 11a: Raw data loss.

Fig. 11b: Effective data loss.

Figure 11: The raw and effective packet loss for a MPEG-4 encoded movie segment of 10 minutes. The average declines from 0.365% to 0.098% with packet retransmissions.

compression ratio of MPEG-4 allows for near NTSC quality video transmissions at less than 1 Mb/s bandwidth requirement, suitable for a residential ADSL connection. For example, our test stream was encoded with a frame size of 720×576 pixels and 25 frames per second (fps). The stream required an average of 105 KB/s (840 Kb/s) bandwidth for both the video and audio layers (audio was encoded in MPEG-1 Layer 3 format); the bit rate of a 10 minute video segment is shown in Fig. 2. This compares favorably with MPEG-1 which would require 1.5 Mb/s for a lower video resolution of 320×240 pixels at 30 fps.

We have conducted several end-to-end video streaming playback tests with the physical setup described in the previous paragraphs and the selective retransmission scheme presented in Sec. 5. Figs. 11a and 11b illustrate the reduction in lost data from an average of 0.365% to 0.098% with this protocol in our test system. Note that the loss characteristic of a real network link is very bursty (as illustrated in Fig. 11a) as compared with the uniform approximations of Eqs. 3 and 4. Hence, the real reduction of the loss rate is less dramatic than would be expected from the analytical equations. However, in practice the protocol worked very well and some of the loss peaks are completely eliminated while others are reduced significantly.

The visual and aural quality of an MPEG-4 encoded movie at less than 1 Mb/s is surprisingly good. Our

test movie, encoded at almost full NTSC resolution, displayed some slight blocking artifacts during complex scenes that can be attributed to the high compression ratio. If a highly compressed movie is streamed over a network and packets are lost, then artifacts are visible. In our setup, because of the low packet loss rate, almost no degradation of the movie was noticeable compared with its local playback.

7 Conclusions and Future Work

Yima is a second generation scalable real-time streaming architecture, which incorporates results from first generation research prototypes, and is consistent with industry standards. By separating the physical disk subsystem used for data storage from logical storage units used for retrieval scheduling, Yima can support heterogeneous magnetic disks and can achieve fault-tolerance. By utilizing a pseudo-random block assignment technique, we are able to reorganize data blocks efficiently when new disks are added or removed. Furthermore, Yima has a rate-control $mechanism$ that can be utilized by a control $policy$ such as Super-streaming, to speedup and slowdown streams. Finally, it supports inter-stream synchronization.

In this article, we explained and compared two alternative designs for Yima: master-slave and bipartite. The former was implemented by porting an operational single-node server (i.e., Darwin Streaming Server) as a component on top of our distributed file server architecture. The advantage of this design as opposed to several independent single-node servers is its load balancing capabilities by avoiding the partitioning of the resources. The main disadvantage is that only a single-node (the master node) exists, from the client perspective, resulting in a bottleneck node and single point of failure. The latter was implemented from scratch utilizing our own lightweight streaming server components. The bottleneck node was eliminated in the bipartite design resulting in close to linear scale-up as we increase the number of nodes.

We also demonstrated that Yima is a working prototype. We discussed the details of its high-end client implementation, the communication protocol and its retransmission component, and Yima's handling of variable-bit-rate video. Finally, extensive sets of experiments were conducted to verify different aspects of our design. The results demonstrated the superiority of Yima-2 in scale-up and rate-control. It also demonstrated the effectiveness of the incorporated retransmission protocol.

We intend to extend Yima in three ways. First, we intend to co-locate Yima at a data center to stream several high-resolution audio and video content to our campus. This environment provides us with a good opportunity to not only perform several intensive tests on Yima but also collect detailed measurements on hiccup and packet-loss rates as well synchronization differences. Second, we plan to extend our experiments to support distributed clients from more than one Yima server. We already have three different hardware setups hosting Yima. By co-locating one Yima server cluster at an off-campus location and the other two servers in different buildings within our campus, we have an initial setup to start our distributed experiments. We have some preliminary approaches to manage distributed continuous media servers [SAK01] that we would like to incorporate, experiment with and extend. Finally, we performed some studies on supporting other media types, in particular the haptic [SKB+01] data type. Our next step would be to store and stream haptic data.

8 Acknowledgment

The authors would like to acknowledge Hong Zhu for his extensive contributions in the development of different Yima clients. Ashish Goel helped with the theoretical aspects of the SCADDAR algorithm and Christos Papodoupolos provided expertise for the network retransmission protocol. We would also like to

thank the following students for helping us with the implementation of certain Yima server components: Farnoush Banaei-Kashani, Nitin Nahata, Atousa Golpayegani and Gary Zeng. Chris Kyriakakis and Ulrich Neumann were our contacts for the 10.2 channel audio and panoramic video content.

References

[AG98] J. A.M. and S. Ghandeharizadeh. An Evaluation of Alternative Disk Scheduling Techniques in Support of Variable Bit Rate Continuous Media. In *EDBT98,Valencia,Spain*, March 1998.

[AH91] D. Anderson and G. Homsy. A continuous media I/O server and its synchronization. *IEEE Computer*, October 1991.

[BBD+96] W. J. Bolosky, J. S. Barrera, R. P. Draves, R. P. Fitzgerald, G. A. Gibson, M. B. Jones, S. P. Levi, N. P. Myhrvold, and R. F. Rashid. The Tiger Video Fileserver. In *6^{th} Workshop on Network and Operating System Support for Digital Audio and Video*, Zushi, Japan, April 1996.

[BGMJ94] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered Striping in Multimedia Information Systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1994.

[CCM97] E. Chang and H. Carcia-Molina. Efficient Memory Use in a Media Server. In *Proceedings of the International Conference on Very Large Databases*, 1997.

[CGM96] E. Chang and H. Garcia-Molina. Reducing Initial Latency in a Multimedia Storage System. In *Proceedings of IEEE International Workshop on Multimedia Database Management Systems*, 1996.

[CGS95] S. Chaudhuri, S. Ghandeharizadeh, and C. Shahabi. Avoiding Retrieval Contention for Composite Multimedia Objects. In *Proceedings of the VLDB Conference*, 1995.

[FD95] C. S. Freedman and D. J. DeWitt. The SPIFFI Scalable Video-on-Demand System. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 352–363, 1995.

[FR94] C. Fedrighi and L. A. Rowe. A Distributed Hierarchical Storage Manager for a Video-on-Demand System. In *Storage and Retrieval for Image and Video Databases II, IS&T/SPIE Symp. on Elec. Imaging Science and Tech.*, pages 185–197, 1994.

[GC92] D. J. Gemmell and S. Christodoulakis. Principles of Delay Sensitive Multimedia Data Storage and Rtrieval. *ACM Trans. Information Systems*, 10(1):51–90, Jan. 1992.

[GKS95] S. Ghandeharizadeh, S. H. Kim, and C. Shahabi. On Configuring a Single Disk Continuous Media Server. In *Proceedings of the 1995 ACM SIGMETRICS/PERFORMANCE*, May 1995.

[Gro97] E. Grochowski. Average Price of Storage and Internal (Media) Data Rate Trend, 1997. IBM Almaden Research Center, San Jose, California.

[GS93] S. Ghandeharizadeh and C. Shahabi. Management of Physical Replicas in Parallel Multimedia Information Systems. In *Proceedings of the Foundations of Data Organization and Algorithms (FODO) Conference*, October 1993.

[GS99] S. Ghandeharizadeh and C. Shahabi. Distributed Multimedia Systems. In John G. Webster, editor, *Wiley Encyclopedia of Electrical and Electronics Engineering*. John Wiley and Sons Ltd., 1999.

[GSYZ00] A. Goel, C. Shahabi, S.Y. D. Yao, and R. Zimmermann. SCADDAR: An Efficient Randomized Technique to Reorganize Continuous Media Blocks. Technical report, University of Southern California, 2000.

[GT99] L. Gao and D. F. Towsley. Supplying Instantaneous Video-on-Demand Services Using Controlled Multicast. In *ICMCS*, volume 2, pages 117–121, 1999.

[GZS+97] S. Ghandeharizadeh, R. Zimmermann, W. Shi, R. Rejaie, D. Ierardi, and T.W. Li. Mitra: A Scalable Continuous Media Server. *Kluwer Multimedia Tools and Applications*, 5(1):79–108, July 1997.

[Hib01] J. Hibbard. What the $%@# is DivX;-)? *Red Herring Magazine*, January 2001.

[HLD+95] J. Hsieh, J. Liu, D. Du, T Ruwart, and M. Lin. Experimental Performance of a Mass Storage System for Video-On-Demand. *Special Issue of Multimedia Systems and Technology of Journal of Parallel and Distributed Computing (JPDC)*, 30(2):147–167, November 1995.

[LLG00] P. W.K. Lie, J. C.S. Lui, and L. Golubchik. Threshold-Based Dynamic Replication in Large-Scale Video-on-Demand Systems. *Multimedia Tools and Applications*, 11(1):35–62, 2000.

[LOP94] A. Laursen, J. Olkin, and M. Porter. Oracle Media Server: Providing Consumer Based Interactive Access to Multimedia Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 470–477, 1994.

[LWMS01] S.H. Lee, K.Y. Whang, Y.S. Moon, and I.Y. Song. Dynamic Buffer Allocation in Video-on-Demand Systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2001.

[MNO+96] C. Martin, P. S. Narayan, B. Özden, R. Rastogi, and A. Silberschatz. The Fellini Multimedia Storage Server. In Soon M. Chung, editor, *Multimedia Information Storage and Management*, chapter 5. Kluwer Academic Publishers, Boston, August 1996. ISBN: 0-7923-9764-9.

[MP4] Moving Pictures Expert Group (MPEG). *MPEG-4 (ISO/IEC 14496)*. URL: http://www.cselt.it/mpeg/standards/mpeg-4/mpeg-4.htm.

[MSB97] R. Muntz, J. Santos, and S. Berson. RIO: A Real-time Multimedia Object Server. *ACM Sigmetrics Performance Evaluation Review*, 25(2):29–35, September 1997.

[NMW97] G. Nerjes, P. Muth, and G. Weikum. Stochastic Service Guarantees for Continuous Data on Multi-Zone Disks. In *Proceedings of the Principles of Database Systems Conference*, pages 154–160, 1997.

[PGK88] D. Patterson, G. Gibson, and R. Katz. A case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, May 1988.

[Pol91] V.G. Polimenis. The Design of a File System that Supports Multimedia. Technical Report TR-91-020, ICSI, 1991.

[PP96] C. Papadopoulos and G. M. Parulkar. Retransmission-Based Error Control for Continuous Media Applications. In *Network and Operating Systems Support for Digital Audio and Video*, Zushi, Japan, April 23-26 1996.

[RR93] S. Ramanathan and P. V. Rangan. Feedback Techniques for Intra-Media Continuity and Inter-Media Synchronization in Distributed Multimedia Systems. *The Computer Journal*, 36(1):19–31, 1993.

[SA00] C. Shahabi and M. Alshayeji. Super-streaming: A New Object Delivery Paradigm for Continuous Media Servers. *Journal of Multimedia Tools and Applications*, 11(1), May 2000.

[SAK01] C. Shahabi, M. Alshayeji, and F. B. Kashani. Resource Management in Distributed Continuous Media. *Under 2nd round of reviews for IEEE Transaction on Parallel and Distributed Systems*, 2001.

[SBE+99] C. Shahabi, G. Barish, B. Ellenberger, N. Jiang, M. Kolahdouzan, A. Nam, and R. Zimmermann. Immersidata Management: Challenges in Management of Data Generated within an Immersive Environment. In *Proceedings of the International Workshop on Multimedia Information Systems*, October 1999.

[SCFJ96] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real Time Applications, 1996. URL: http://ww.itef.org/rfc/rfc1889.txt.

[SCO90] Margo Seltzer, Peter Chen, and John Ousterhout. Disk Scheduling Revisited. In *Proceedings of the 1990 Winter USENIX Conference*, pages 313–324, Washington DC, Usenix Association, 1990.

[SGC01] C. Shahabi, S. Ghandeharizadeh, and S. Chaudhuri. On Scheduling Atomic and Composite Multimedia Objects. In *Transactions on Knowledge and Data Engineering*, 2001.

[SKB+01] C. Shahabi, M. R. Kolahdouzan, G. Barish, R. Zimmermann, D. Yao, K. Fu, and L. Zhang. Alternative Techniques for the Efficient Acquisition of Haptic Data. In *ACM SIGMETRICS/Performance*, 2001.

[SM98] J. R. Santos and R. R. Muntz. Performance Analysis of the RIO Multimedia Storage System with Heterogeneous Disk Configurations. In *ACM Multimedia Conference*, Bristol, UK, 1998.

[SRL98] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP), 1998. URL: http://ww.itef.org/rfc/-rfc2326.txt.

[SY98] H. Shachnai and P. S. Yu. Exploring Wait Tolerance in Effective Batching for Video-on-Demand Scheduling. *Multimedia Systems*, 6(6):382–394, 1998.

[TPBG93] F.A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID-A Disk Array Management System for Video Files. In *First ACM Conference on Multimedia*, August 1993.

[YCK93] P.S. Yu, M-S. Chen, and D.D. Kandlur. Grouped Sweeping Scheduling for DASD-based Multimedia Storage Management. *Multimedia Systems*, 1(1):99–109, January 1993.

[ZG97] R. Zimmermann and S. Ghandeharizadeh. Continuous Display Using Heterogeneous Disk-Subsystems. In *Proceedings of the Fifth ACM Multimedia Conference*, pages 227–236, Seattle, Washington, November 9-13, 1997.

[ZG00] R. Zimmermann and S. Ghandeharizadeh. HERA: Heterogeneous Extension of RAID. In *Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, Las Vegas, Nevada, June 26-29 2000.

[Zim98] R. Zimmermann. *Continuous Media Placement and Scheduling in Heterogeneous Disk Storage Systems*. Ph.D. Dissertation, University of Southern California, Los Angeles, California, December 1998.