Yima: A Second-Generation Continuous Media Server

Yima, a scalable real-time streaming architecture, incorporates lessons learned from earlier research prototypes to enable advanced continuous media services.

Cyrus Shahabi
Roger Zimmermann
Kun Fu
Shu-Yuen Didi Yao
University of Southern California

Applications such as news on demand, distance learning, e-commerce, and scientific visualization all store, maintain, and retrieve large volumes of real-time data over a network. These data are denoted collectively as *continuous media*, or CM. Video and audio objects are popular examples; haptic and avatar data are less familiar types. CM data require a streaming architecture that can, first, manage real-time delivery constraints. Failure to meet these constraints on CM data disrupts the display with "hiccups." Second, the architecture must address the large size of CM objects. A two-hour MPEG-2 video with a bandwidth requirement of 4 megabits per second is 3.6 gigabytes in size.

The currently available commercial implementations of CM servers fall into two broad categories:

- low-cost, single-node, consumer-oriented systems serving a limited number of users; and
- multinode, carrier-class systems such as high-end broadcasting and dedicated video-on-demand systems.

RealNetworks, Apple Computer, and Microsoft product offerings fit into the consumer-oriented category, while SeaChange and nCube offer solutions oriented toward carrier-class systems. While commercial systems ordinarily use proprietary technology and algorithms, making it difficult to compare their products with research prototypes, we have designed and developed a second-generation CM server that demonstrates several advanced concepts.

We call our system Yima, a name denoting the first man in ancient Iranian religion. While Yima has not achieved the refinement of commercial solutions, it is operational and incorporates lessons learned from first-generation research prototypes.[1,2] Yima distinguishes itself from other similar research efforts in the following:

- complete distribution with all nodes running identical software and no single points of failure;
- efficient online scalability allowing disks to be added or removed without interrupting CM streams;
- synchronization of several streams of audio, video, or both within one frame (1/30 second);
- independence from media types;
- compliance with industry standards;
- selective retransmission protocol; and
- multithreshold buffering flow-control mechanism to support variable bit-rate (VBR) media.

Yima is also a complete end-to-end system that uses an IP network with several supportable client types. This feature distinguishes it from previous research that focused heavily on server design.

SYSTEM ARCHITECTURE

Figure 1 shows the overall Yima system architecture. In our prototype implementation, the server consists of an eight-way cluster of rack-mountable Dell PowerEdge 1550 Pentium III 866-MHz PCs with 256 Mbytes of memory running Red Hat

0018-9162/02/$17.00 © 2002 IEEE

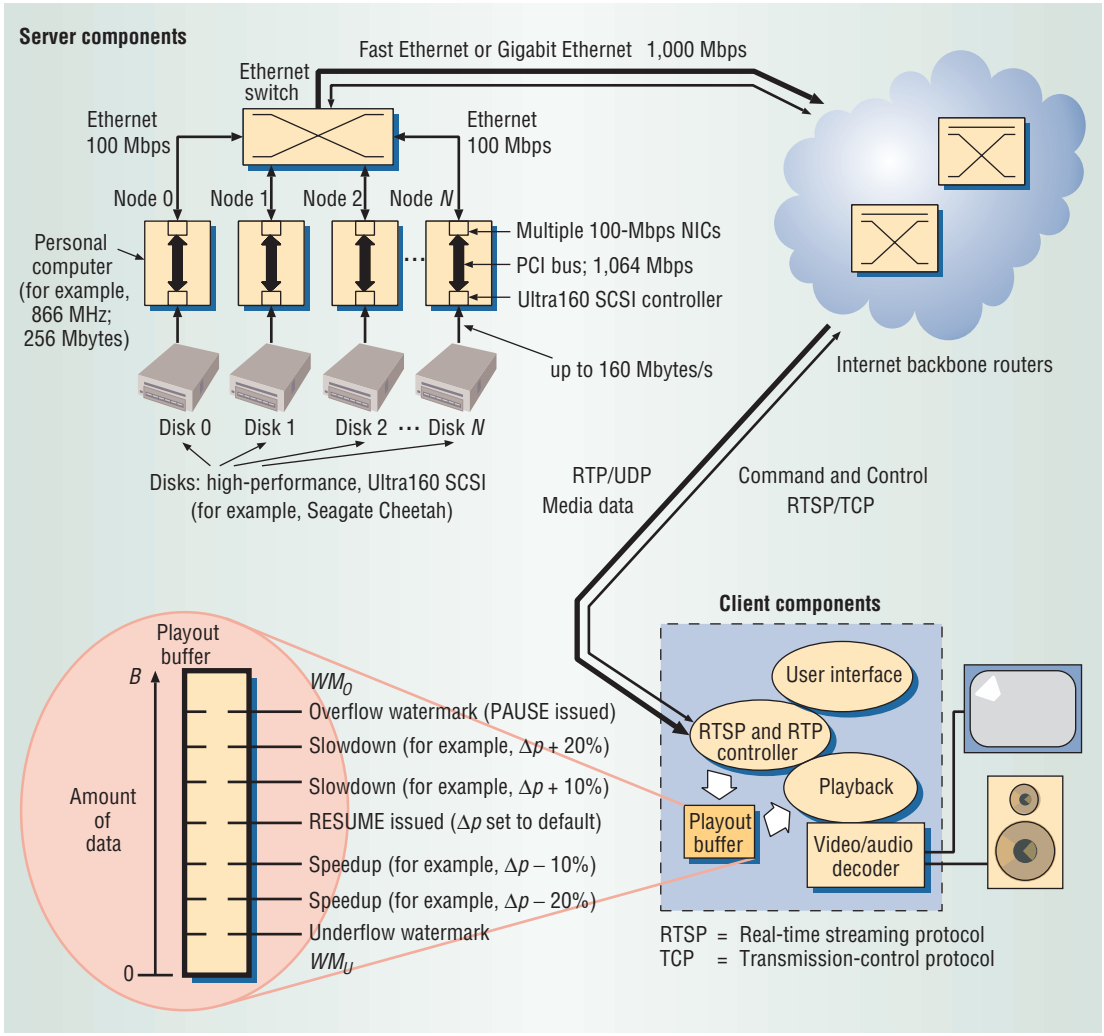

Figure 1. Yima system architecture. The prototype implementation uses off-the-shelf commodity hardware components and industry standards end to end.

Linux. Sixteen 36-Gbyte Seagate Cheetah hard-disk drives store the media data and connect to the server nodes via Ultra160 small computer system interface (SCSI) channels.

The nodes in the cluster communicate with each other and send the media data via multiple 100-Mbps Fast Ethernet connections. Each server is attached to a local Cabletron 6000 switch with either one or two Fast Ethernet lines. The local switch connects to both a WAN backbone for serving distant clients and a LAN environment for local clients. Our testbed also includes server clusters at other remote locations, for example, Metromedia Fiber Network in El Segundo, California, and Information Sciences Institute East in Arlington, Virginia.

Choosing an IP-based network keeps the per-port equipment cost low and makes the system immediately compatible with the public Internet.

The current prototype implements clients on standard Pentium III PC platforms, but we could also port them to digital television set-top boxes. The client software, Yima Presentation Player, runs on either Red Hat Linux or Windows NT. Structured into several components, the player lets various software and hardware decoders be plugged in. Table 1 shows the different media types that Yima currently recognizes. One unusual type is panoramic video with 10.2-channel audio.

SERVER DESIGN CHALLENGES

The servers for delivering isochronous multimedia over IP networks must store the data efficiently and schedule the data retrieval and delivery precisely before transmission. We studied both master-slave and bipartite design approaches in the Yima-1 and Yima-2 CM servers, respectively. These approaches share many features that address design challenges in this domain. They differ mainly in the logical interconnection topology between cluster nodes.

Data placement and scheduling

There are two ways to assign data blocks to the magnetic disk drives that form the storage system: in a round-robin placement[3] or randomly.[4] Traditionally, round-robin placement uses a cycle-based approach for resource scheduling to guarantee a continuous display, while random placement uses a deadline-driven approach.

In general, the round-robin cycle-based approach provides high throughput with little wasted band-

Table 1. Yima client media support.

Media type	Decoder	Channels	Operating system	Minimum CPU speed	Video resolution (in pixels)	Audio encoding	Delivery rate
DivX MPEG-4	Software	1 video, 2 audio	Linux	500 MHz	720 × 480	MP3	<1 Mbps
MPEG-2 and Dolby Digital	Creative Dxr2 DVD	1 video, 5.1 audio	Linux	200 MHz	720 × 480	Dolby AC-3	6-8 Mbps
MPEG-2 HD	Software	1 video	Linux	>2 × 1.5 GHz	1,920 × 1,080		19.4 Mbps
MPEG-2 HD	Vela Research CineCast HD	1 video, 10.2 audio	Linux	500 MHz	1,920 × 1,080	Dolby AC-3 or uncompressed PCM	19.4-45 Mbps and 11 Mbps
Panoramic MPEG-2	Vela Research CineCast	5 video, 10.2 audio	Windows NT	2 × 400 MHz	(5 × 720) × 480 each	Uncompressed PCM	4 × 5 Mbps and 11 Mbps

width for video objects that are retrieved sequentially, such as a feature-length movie. The startup latency for an object might be large under heavy loads, but object replication can reduce it.[5]

The random deadline-driven approach supports fewer optimizations, so it could lower throughput, but several benefits outweigh this potential drawback.[6] First, random data placement supports multiple delivery rates with a single server block size; it also simplifies the scheduler design, supports interactive applications, and automatically achieves the average transfer rate with multizoned disks. Finally, random placement reorganizes data more efficiently when the system scales up or down.

Random placement can require a large amount of metadata to store and manage each block's location in a centralized repository, for example, in tuples of the form <$node_x$, $disk_y$>. Yima avoids this overhead by using a pseudorandom block placement. A seed value initiates a sequence of numbers that can be reproduced by using the same seed value. By placing blocks in a pseudorandom fashion across the disks, the system can recompute the block locations. Since Yima numbers disks globally across the server nodes, it will assign blocks to random disks across different nodes.

Hence, Yima stores only the seed for each file object instead of locations for every block.

Scalability, heterogeneity, and fault resilience

Any CM server design must scale to support growth in user demand or application requirements. Several techniques address this requirement, including the use of multidisk arrays. However, if the design connected all the disks to a single large computer, the I/O bandwidth constraints would limit the overall achievable throughput—hence, Yima's architecture uses multiple computers, or multinodes.

As Figure 1 shows, the Yima server architecture interconnects storage nodes via a high-speed network fabric that can expand as demand increases. This modular architecture makes it easy to upgrade older PCs and add new nodes.

Applications that rely on large-scale CM servers, such as video-on-demand, require continuous operation. To achieve high reliability and availability for all data stored in the server, Yima uses disk merging[7] to implement a parity-based data-redundancy scheme that, in addition to providing fault tolerance, can also take advantage of a heterogeneous storage subsystem. Disk merging presents a virtual view of logical disks on top of the actual physical storage system, which might consist of disks that provide different bandwidths and storage space. This abstraction allows a system's application layers to assume a uniform characteristic for all the logical disks, which in turn allows using conventional scheduling and data placement algorithms across the physical storage system.

Data reorganization

Computer clusters try to balance load distribution across all nodes. Over time, both round-robin and random data-placement techniques distribute data retrievals evenly across all disk drives. When a system operator adds a node or disk, however, the system must redistribute the data to avoid partitioning the server. Reorganizing the blocks involves much less overhead when the system uses random rather than round-robin placement. For example, with round-robin striping, adding or removing a disk requires the relocation of almost all data blocks. Randomized placement requires moving only a fraction of the blocks from each disk to the added disk—just enough to ensure that the blocks are still randomly placed to preserve the load balance.

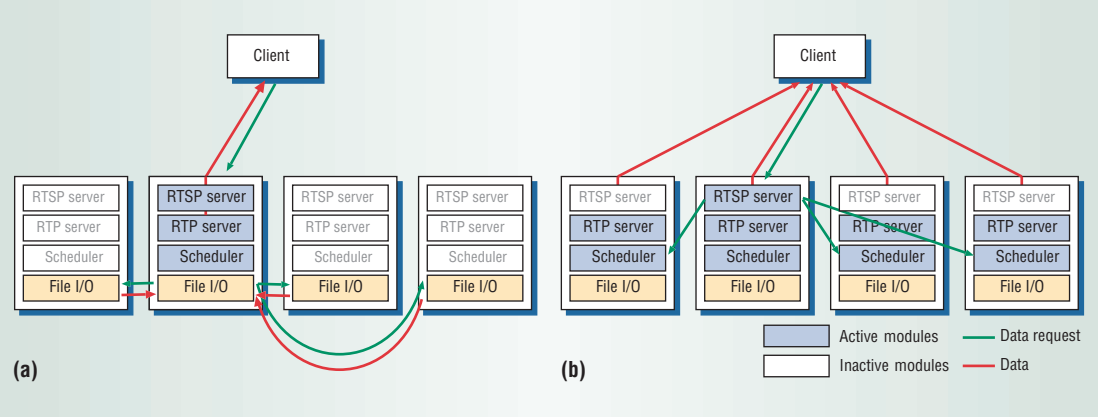

Figure 2. Client session view of the Yima server. (a) Data is sent through the master node in Yima-1, and (b) data is sent from all the nodes in Yima-2.

Yima uses a pseudorandom number generator to produce a random, yet reproducible, number sequence to determine block locations. Because some blocks must move to the added disks when the system scales up, Yima cannot use the previous pseudorandom number sequence to find the blocks; therefore, Yima must derive a new random number sequence. We use a composition of random functions to determine this new sequence. Our approach—termed Scaling Disks for Data Arranged Randomly (Scaddar)—preserves the sequence's pseudorandom properties, resulting in minimal block movements and little overhead in the computation of new locations.[8] The Scaddar algorithm can support disk scaling while Yima is online.

Multinode server architecture

We built the Yima servers from clusters of server PCs called nodes. A distributed file system provides a complete view of all the data on every node without requiring individual data blocks to be replicated, except as required for fault tolerance.[7] A Yima cluster can run in either a master-slave or bipartite mode.

Master-slave design (Yima-1). With this design, an application running on a specific node operates on all local and remote files. Operations on remote files require network access 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 Figure 2a shows, the distributed file system consists 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) module, and a real-time protocol (RTP) module. Each Yima-1 node runs the distributed file system, while certain nodes also run the Yima-1 media-streaming server. A node running only the file I/O module has only slave capabilities, while a node that runs both components has master and slave capabilities.

A master server node is a client's point of contact during a session. We define a session as a complete RTSP transaction for a CM stream. When a client wants to request a data stream using RTSP, it connects to a master server node, which in turn brokers the request to the slave nodes. If multiple master nodes exist in the cluster, this assignment is decided based on a round-robin domain name service (RR-DNS) or a load-balancing switch. A pseudorandom number generator manages the locations of all data blocks.

Using a distributed file system obviates the need for applications to be aware of the storage system's distributed nature. Even applications designed for a single node can to some degree take advantage of this cluster organization. The Yima-1 media streaming server component, based on Apple's Darwin Streaming Server (DSS) project (http://www.opensource.apple.com/projects/streaming/), assumes that all media data reside in a single local directory. Enhanced with our distributed file system, multiple copies of the DSS code—each copy running on its own master node—can share the same media data. This also simplifies our client design since it sends all RTSP control commands to only one server node.

Finally, Yima-1 uses a pause-resume flow-control technique to deliver VBR media. A stream is sent at a rate of either R_N or zero megabits per second, where R_N is an estimated peak transfer rate for the movie. The client issues pause-and-resume commands to the server depending on how full the client buffer is. Although the pause-resume design is simple and effective, its on-off nature can lead to bursty traffic.

With the Yima-1 architecture, several major performance problems offset the ease of using clustered storage, such as a single point of failure at the master node and heavy internode traffic. These drawbacks motivated the design of Yima-2, which provides a higher performing and more scalable solution for managing internode traffic.

Bipartite design (Yima-2). We based Yima-2's bipartite model on two groups of nodes: a server group and a client group.

Yima-2 offers the flexibility of delivering any data type while still being compatible with the MPEG-4 industry standard.

With Yima-1, the scheduler, RTSP, and RTP server modules are all centralized on a single master node from the viewpoint of a single client. 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 Figure 2b shows. In Yima-2, every node retrieves, schedules, and sends its own local data blocks directly to the requesting client, thereby eliminating Yima-1's master-node bottleneck. These improvements significantly reduce internode traffic.

Although the bipartite design offers clear advantages, its realization imposes several new challenges. First, clients must handle receiving data from multiple nodes. Second, we replaced the original DSS code component with a distributed scheduler and RTP server to achieve Yima-2's decentralized architecture. Last, Yima-2 requires a flow-control mechanism to prevent client buffer overflow or starvation.

With Yima-2, each client maintains contact with one RTSP module throughout a session for control information. For load-balancing purposes, each server node can run an RTSP module, and the decision of which RTSP server to contact remains the same as in Yima-1: RR-DNS or switch. However, contrary to the Yima-1 design, a simple RR-DNS cannot make the server cluster appear as one node since clients must communicate with individual nodes for retransmissions. Moreover, if an RTSP server fails, sessions are not lost. Instead, the system reassigns the sessions to another RTSP server, with no disruption in data delivery.

We adapted the MPEG-4 file format as specified in MPEG-4 Version 2 for the storage of media blocks. This flexible-container format is based on Apple's QuickTime file format. In Yima-2, we expanded on the MPEG-4 format by allowing encapsulation of other compressed media data such as MPEG-2. This offers the flexibility of delivering any data type while still being compatible with the MPEG-4 industry standard.

To avoid bursty traffic caused by Yima-1's pause/resume transmission scheme and still accommodate VBR media, the client sends feedback to make minor adjustments to the data transmission rate in Yima-2. By sending occasional slowdown or speedup commands to the Yima-2 server, the client can receive a smooth data flow by monitoring the amount of data in its buffer.

CLIENT SYSTEMS

We built the Yima Presentation Player as a client application to demonstrate and experiment with our Yima server. The player can display a variety of media types on both Linux and Windows platforms. Clients receive streams via standard RTSP and RTP communications.

Client buffer management

A circular buffer in the Yima Presentation Player reassembles VBR media streams from RTP packets that are received from the server nodes. Researchers have proposed numerous techniques to smooth the variable consumption rate R_C by approximating it with a number of constant-rate segments. Implementing such algorithms at the server side, however, requires complete knowledge of R_C as a function of time.

We based our buffer management techniques on a flow-control mechanism so they would work in a dynamic environment. A circular buffer of size B accumulates the media data and keeps track of several watermarks including buffer overflow WM_O and buffer underflow WM_U. The decoding thread consumes data from the same buffer. Two schemes, pause/resume and Δp, control the data flow.

Pause-resume. If the data in the buffer reaches WM_O, the client software pauses the data flow from the server. The playback will continue to consume media data from the buffer.

When the data in the buffer reaches the underflow watermark WM_U, the stream from the server resumes. However, the buffer must set WM_O and WM_U with safety margins that account for network delays. Consequently, if the data delivery rate (R_N) is set correctly, the buffer will not underflow while the stream is resumed.

Although the pause/resume technique is a simple and effective design, if pause and resume actions coincide across multiple sessions, bursty traffic will become a noticeable effect.

Client-controlled Δp. Δp is the interpacket delivery time the schedulers use to transmit packets to the client. Schedulers use the network time protocol (NTP) to synchronize time across nodes. Using a common time reference and each packet's time stamp, server nodes send packets in sequence at timed intervals.

The client fine-tunes the delivery rate by updating the server with new Δp values based on the amount of data in its buffer. Fine-tuning is achieved by using multiple watermarks in addition to WM_O and WM_U, as Figure 1 shows.

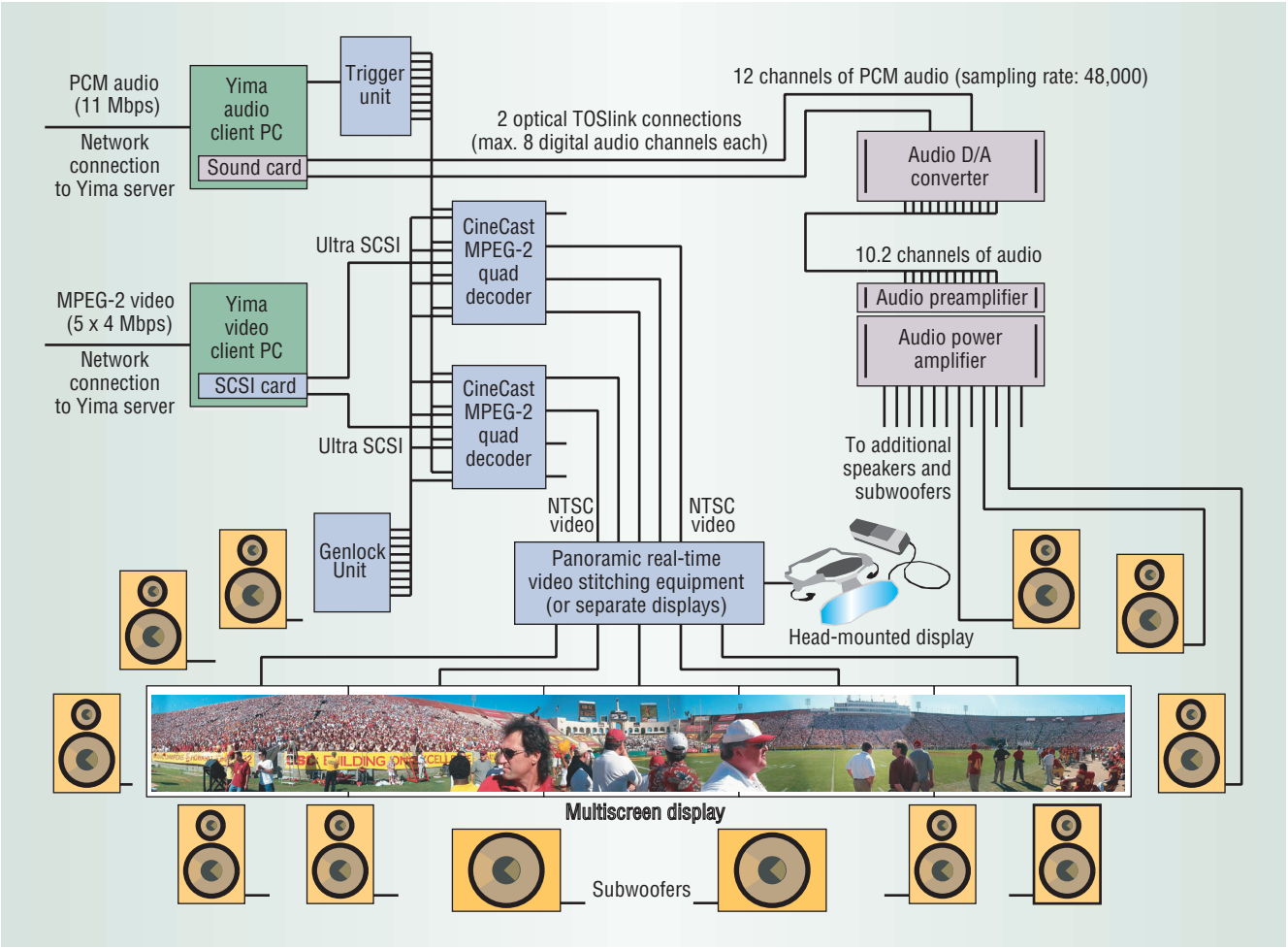

Figure 3. Panoramic video and 10.2-channel audio playback system block diagram. One Yima client renders five channels of synchronized video in a mosaic of 3,600 × 480 pixels while another Yima client renders 10.2 channels of synchronized audio (0.2 refers to two low-frequency channels, or subwoofers).

When the level of data in the client buffer reaches a watermark, the client sends a corresponding Δp speedup or slowdown command to maintain the amount of data within the buffer. The buffer smoothes out any fluctuations in network traffic or server load imbalance that might delay packets. Thus, the client can control the delivery rate of received data to achieve smoother delivery, prevent bursty traffic, and keep a constant level of buffer data.

Player media types

We have experimented with a variety of media types for our Yima player. Figure 1 shows the player's three-threaded structure. The playback thread interfaces with the actual media decoder. The decoder can be either software- or hardware-based. Table 1 lists some decoders that we incorporated.

The CineCast hardware MPEG decoders from Vela Research support both MPEG-1 and MPEG-2 video and two-channel audio. For content that includes 5.1 channels of Dolby Digital audio, as used in DVD movies, we use the Dxr2 PCI card from Creative Technology to decompress both MPEG-1 and MPEG-2 video in hardware. The card also decodes MPEG audio and provides a 5.1-channel Sony-Philips Digital Interface (SP-DIF) digital audio output terminal.

With the emergence of MPEG-4, we began experimenting with a DivX software decoder.[9] MPEG-4 provides a higher compression ratio than MPEG-2. A typical 6-Mbps MPEG-2 media file may only require an 800-Kbps delivery rate when encoded with MPEG-4. We delivered an MPEG-4 video stream at near NTSC quality to a residential client site via an ADSL connection.[10]

HDTV client

The streaming of high-definition content presented several challenges. First, high-definition media require a high-transmission bandwidth. For example, a video resolution of 1,920 x 1,080 pixels encoded via MPEG-2 results in a data rate of 19.4 Mbps. This was less of a problem on the server side because we designed Yima to handle high data rates.

The more intriguing problems arose on the client side. We integrated an mpeg2dec open source software decoder because it was cost-effective. Although it decoded our content, achieving real-time frame rates with high-definition video was nontrivial because of the high resolution. On a dual-processor 933-MHz Pentium III, we achieved

A Yima client can render up to eight synchronous streams of MPEG-2 video and 24 audio channels.

approximately 20 frames per second using unoptimized code with Red Hat Linux 6.2 and Xfree86 4.0.1 on an nVidia Quadro 2 graphics accelerator. In our most recent implementation, we used a Vela Research CineCast HD hardware decoder, which achieved real-time frame rates at data rates up to 45 Mbps.

Multistream synchronization

The flow-control techniques implemented in the Yima client-server communications protocol synchronize multiple, independently stored media streams.

Figure 3 shows the client configuration for the playback of panoramic, five-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. We encode each video channel into a standard MPEG-2 program stream. The client receives the 10.2 channels of high-quality, uncompressed audio separately.

During playback, all streams must render in tight synchronization so the five video frames corresponding to one time instance combine accurately into a panoramic mosaic of 3,600 x 480 pixels every 1/30th of a second. The player can show the resulting panoramic video on either a wide-screen or head-mounted display. The experience is enhanced with 10.2-channel surround audio, presented phase-accurately and in synchronization with the video.

Yima achieves precise playback with three levels of synchronization: block-level via retrieval scheduling, coarse-grained via the flow-control protocol, and fine-grained through hardware support. The flow-control protocol maintains approximately the same amount of data in all client buffers. With this prerequisite in place, we can use multiple CineCast decoders and a genlock timing-signal-generator device to lock-step the hardware MPEG decoders to produce frame-accurate output. All streams must start precisely at the same time.

The CineCast decoders provide an external trigger that accurately initiates playback through software. Using two PCs, one equipped with two four-channel CineCast decoders and one with a multichannel sound card, a Yima client can render up to eight synchronous streams of MPEG-2 video and 24 audio channels.

RTP/UDP AND SELECTIVE RETRANSMISSION

Yima supports the industry-standard RTP for the delivery of time-sensitive data. Because RTP transmissions are based on the best-effort user datagram protocol, a data packet could arrive out of order at the client or be altogether dropped along the network. To reduce the number of lost RTP data packets, we implemented a selective retransmission protocol.[11] We configured the protocol to attempt at most one retransmission of each lost RTP packet, but only if the retransmitted packet would arrive in time for consumption.

When multiple servers deliver packets that are part of a single stream, as with Yima-2, and a packet does not arrive, how does the client know which server node attempted to send it? In other words, it is not obvious where the client should send its retransmission request.

There are two solutions to this problem. The client can broadcast the retransmission request to all server nodes, or it can compute the server node to which it issues the retransmission request. With the *broadcast approach*, all server nodes receive a packet retransmission request, check whether they hold the packet, and either ignore the request or perform a retransmission. Consequently, broadcasting wastes network bandwidth and increases server load.

Yima-2 incorporates the *unicast approach*. Instead of broadcasting a retransmission request to all the server nodes, the client unicasts the request to the specific server node possessing the requested packet. The client determines the server node from which a lost RTP packet was intended to be delivered by detecting gaps in node-specific packet sequence numbers. Although this approach requires packets to contain a node-specific sequence number along with a global sequence number, the clients require very little computation to identify and locate missing packets.

TEST RESULTS

In extensive sets of experiments, Yima-2 exhibits an almost perfectly linear increase in the number of streams as the number of nodes increases. Yima-2's performance may become sublinear with larger configurations, low-bit-rate streams, or both, but it scales much better than Yima-1, which levels off early. We attribute Yima-1's nonlinearity to the increase of internodal data traffic.

We sent MPEG-4 data from the Yima servers in our lab to the public Internet via the University of Southern California campus network. The geographical distance between the two end points measured approximately 40 kilometers. We set up the client in a residential apartment and linked it to the Internet via an ADSL connection. The ADSL provider did not guarantee any minimum band-

width but stated that it would not exceed 1.5 Mbps. The raw bandwidth achieved end-to-end between the Yima client and servers was approximately 1 Mbps.

The visual and aural quality of an MPEG-4 encoded movie at less than 1 Mbps is surprisingly good. Our test movie, encoded at almost full NTSC resolution, displayed little degradation—a performance attributable to the low packet loss rate of 0.365 percent without retransmissions and 0.098 percent with retransmissions. The results demonstrated the superiority of Yima-2 in scale-up and rate control. They also demonstrated the incorporated retransmission protocol's effectiveness.

We colocated a Yima server at Metromedia Fiber Network in El Segundo, California, to demonstrate successful streaming of five synchronized video channels. Also, as part of a remote media immersion experiment (http://infolab.usc.edu/News/NYT-RML.html). We successfully streamed HD video at 45 Mbps from Arlington, Virginia, synchronized with 10.2-channel audio at 11 Mbps from Marina del Rey, California, to our lab at the University of Southern California.

We are exploring resource management strategies across both distributed and peer-to-peer architectures in which multiple Yima clusters would exist across geographically dispersed areas.[12] This distribution would allow a wider range of serviceable clients. We also plan to extend the support of data types to include haptic and avatar data as part of the overall research in immersive media at USC's Integrated Media Systems Center. ■

Acknowledgments

This research has been funded by the US National Science Foundation grants EEC-9529152 (IMSC ERC) and IIS-0082826. We thank our IMSC collaborators Chrysostomos L. Nikias, Ulrich Neumann, Alexander Sawchuk, Chris Kyriakakis, Christos Papadopoulos, and Albert Rizzo. We also thank the following students for helping with the implementation of certain Yima components: Mehrdad Jahangiri, Nitin Nahata, Sahitya Gupta, Farnoush Banaei-Kashani, and Hong Zhu.

References

1. D.J. Gemmell et al., "Multimedia Storage Servers: A Tutorial," *Computer*, May 1995, pp. 40-49.
2. A. Bonhomme, "Survey of Video Servers," hyperlinked resource page, including bibliography, http://www.ens-lyon.fr/~abonhomm/video/survey.html (current May 2002; last update June 2001).
3. S. Berson et al., "Staggered Striping in Multimedia Information Systems," *Proc. 1994 ACM Sigmod Int'l Conf. Management of Data*, ACM Press, New York, 1994, pp. 79-90.
4. J.R. Santos and R.R. Muntz, "Performance Analysis of the RIO Multimedia Storage System with Heterogeneous Disk Configurations," *Proc. 6th ACM Int'l Multimedia Conference* (ACM MM 98), ACM Press, New York, 1998, pp. 303-308.
5. S. Ghandeharizadeh et al., "On Minimizing Startup Latency in Scalable Continuous Media Servers," *Proc. Multimedia Computing and Networking* (MMCN 97), SPIE-Int'l Society Optical Engineering, Bellingham, Wash., 1997, pp. 144-155.
6. J.R. Santos, R. Muntz, and B. Ribeiro-Neto, "Comparing Random Data Allocation and Data Striping in Multimedia Servers," *Int'l Conf. Measurement and Modeling of Computer Systems* (Sigmetrics 2000), ACM Press, New York, 2000, pp. 44-55.
7. R. Zimmermann and S. Ghandeharizadeh, "Continuous Display Using Heterogeneous Disk-Subsystems," *Proc. 5th ACM Int'l Multimedia Conf.* (ACM MM 97), ACM Press, New York, 1997, pp. 227-236.
8. A. Goel et al., "Scaddar: An Efficient Randomized Technique to Reorganize Continuous Media Blocks," *Proc. 18th Int'l Conf. Data Eng.* (ICDE 02), IEEE CS Press, Los Alamitos, Calif., 2002, pp. 473-482.
9. J. Hibbard, "What the $%@# is DivX;-)?" *Red Herring Magazine*, Jan. 2001, pp. 60-64.
10. R. Zimmermann et al., "Yima: Design and Evaluation of a Streaming Media System for Residential Broadband Services," *Proc. VLDB 2001 Workshop Databases in Telecommunications* (DBTel 01), Springer-Verlag, Berlin, 2001, pp. 116-125.
11. C. Papadopoulos and G.M. Parulkar, "Retransmission-Based Error Control for Continuous Media Applications," *Proc. 6th Int'l Workshop Network and Operating Systems Support for Digital Audio and Video* (NOSSDAV 96), Springer-Verlag, Heidelberg, 1996, pp. 5-12.
12. C. Shahabi and F. Banaei-Kashani, "Decentralized Resource Management for a Distributed Continuous Media Server," to be published in *IEEE Trans. Parallel and Distributed Systems*, vol. 13, no. 6, June 2002.

***Cyrus Shahabi** is an assistant professor and director of the Information Laboratory (http://infolab.usc.edu) in the Computer Science Department at*

the University of Southern California. He is also director of the Information Management Research Area at the Integrated Media Systems Center, an NSF Engineering Research Center at USC. His research interests include multidimensional databases, multimedia servers, and data mining. Shahabi received a PhD in computer science from USC. He is a member of the IEEE and the ACM. Contact him at shahabi@usc.edu.

***Roger Zimmermann** is a research assistant professor in the Computer Science Department at the University of Southern California and director of the Media Immersion Environment Research Area at the Integrated Media Systems Center. His interests include novel database architectures for immersive environments, video-streaming technology, cluster and distributed computing, and fault-resilient storage architectures. Zimmermann received a PhD in computer science from USC. He is a member of the IEEE and the ACM. Contact him at rzimmerm@usc.edu.*

***Kun Fu** is a doctoral candidate in computer science at the University of Southern California. His research interests include multimedia servers, real-time data distribution, and parallel computing. Fu received an MS in engineering science from the University of Toledo. Contact him at kunfu@cs.usc.edu.*

***Shu-Yuen Didi Yao** is a doctoral candidate in computer science at the University of Southern California. His research interests include scalable storage architectures, multimedia servers, video streaming, and fault-tolerant systems. Yao received an MS in computer science from USC. He is a member of the ACM. Contact him at didiyao@cs.usc.edu.*