SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2005; **35**:345–359
Published online 21 December 2004 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe.639

Scalability evaluation of the *Yima* streaming media architecture

Roger Zimmermann*,†, Cyrus Shahabi, Kun Fu and Shu-Yuen Didi Yao

Integrated Media Systems Center and Department of Computer Science, University of Southern California, Los Angeles, CA 90089–2561, U.S.A.

SUMMARY

Over the last decade research has been pursued on all aspects of streaming media. While many theoretical results have been reported in the literature, few performance results of large-scale systems have been published. In this report we specifically explore the scalability aspects of our Yima streaming media architecture in an end-to-end test environment. With Yima, it was our goal to design and implement an architecture that would scale in performance from small to large systems. Some of the design features include (1) a multi-node cluster architecture based on commodity hardware and custom software, (2) media type independence (support ranges from 500 Kb s^{-1} MPEG-4 to 45 Mb s^{-1} HDTV, at both variable and constant bitrates), (3) fine-grained online scale up/down capabilities, and (4) a client-controlled rate smoothing protocol. We briefly discuss the design and implementation of these capabilities of Yima and then thoroughly evaluate its scalability through several sets of experiments. Our results show that Yima scales linearly (within the range of our test parameters) as a function of the cluster size and also as a function of available resources such as network bandwidth and CPU performance. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: streaming media; continuous media; multimedia servers

1. INTRODUCTION

We report on the implementation and evaluation of a scalable real-time streaming media architecture called *Yima*‡, which enables 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.

*Correspondence to: Roger Zimmermann, Integrated Media Systems Center and Department of Computer Science, University of Southern California, Los Angeles, CA 90089–2561, U.S.A.
†E-mail: rzimmerm@imsc.usc.edu

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

Contract/grant sponsor: National Science Foundation; contract/grant numbers: EEC-9529152 (IMSC ERC) and IIS-0082826

Received 10 June 2003
Revised 21 June 2004
Accepted 21 June 2004

CM is distinguished from traditional textual and record-based media in two ways. First, the retrieval and display of CM are subject to real-time constraints. Second, CM objects are large. A high-definition MPEG-2 stream with a 19.4 megabits per second (Mb s^{-1}) bandwidth requirement such as the *Tournament of Roses* broadcast on New Year's day requires 145 Mb min^{-1} of storage or about 26 GB for 3 h. Popular examples of CM are video and audio objects, while less familiar examples are haptic, avatar and application coordination data [1].

The first research reports on the design of CM servers appeared about a decade ago, followed by a steady stream of publications on this topic until today. Many of the investigations focused on algorithms and simulations while only a few resulted in prototype implementations. Examples are Streaming-RAID [2], the Oracle Media Server [3], the UMN system [4], Tiger [5], Fellini [6], Mitra [7] and RIO [8]. These first-generation CM servers were primarily addressing the design of different data placement paradigms, buffer management mechanisms, and retrieval scheduling techniques to optimize for high throughput and/or low startup latency time.

While contributing to the state-of-the-art, these early prototypes have been at a disadvantage in two respects. First, since they were implemented concurrently during the same time frame, each one of them could not take advantage of the successes and failures of the other projects. Second, almost all of these research prototypes were completed before the industry's standardizations for streaming CM over IP networks. Hence, each prototype has its own proprietary media content format, client (and codec) implementation and communication/network protocol. Some of these prototypes focused solely on the server design and never reported on their network and client configurations. They mainly assumed a very fast network and constant bitrate media types in their corresponding research publications. From a practical point of view, these environment assumptions are not realistic.

In this paper we describe and evaluate an end-to-end implementation of the distributed Yima architecture. We assess its scalability through numerous experiments with several parameter sets (e.g. different amounts of resources such as network bandwidth). The focus of our implementation has been on providing a high-performance, scalable system that builds upon and extends the latest research results and is fully compatible with open industry standards. For example, 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 the expensive shared-memory architecture of UMN (based on SGI's Onyx), we employed a shared-nothing approach on commodity personal computer hardware.

The remainder of this paper is organized as follows. First we describe some of the details of the fully distributed Yima architecture in Section 2. In Section 3, we evaluate the scalability of the Yima architecture through several sets of experiments with a complete end-to-end implementation. For example, we study the scenarios when different resources of a node (i.e. CPU and network) become a bottleneck. Finally, Section 4 concludes the paper and discusses our future plans in this area.

2. SYSTEM ARCHITECTURE

2.1. Overview of the Yima architecture

A detailed description of the Yima architecture design is provided in [9]. Here we summarize some of its features. Yima is designed as a completely distributed system (with no single point of failure

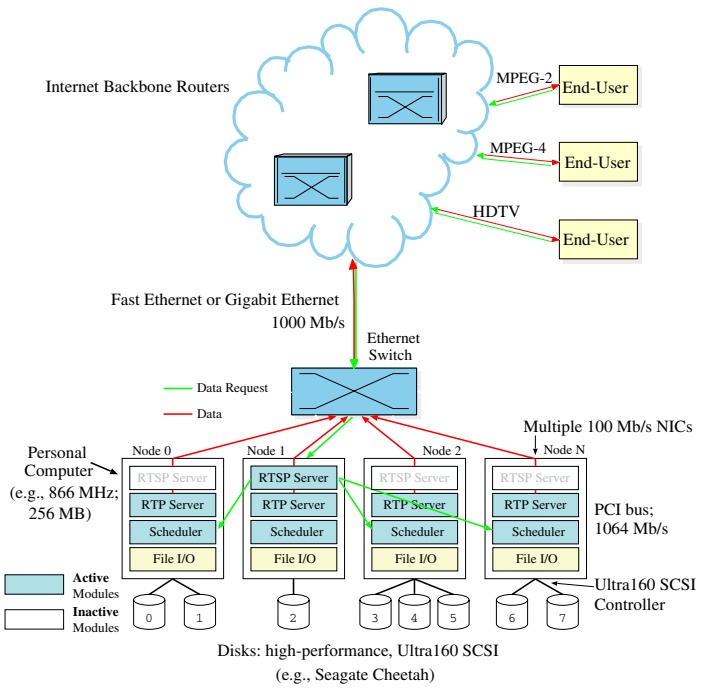

Figure 1. The Yima multi-node hardware and software architecture. Each node is based on a commodity PC and connects to one or more disk drives and the network. Four software modules run on each node.

or bottleneck) on a multi-node, multi-disk platform as shown in Figure 1. It separates physical disks (used to store the data) from the concept of logical disks (used for retrieval scheduling) to support fault tolerance [10] and heterogeneous disk subsystems [11]. Data blocks are pseudo-randomly placed on all nodes and non-deterministic scheduling is performed locally on each node. Yima includes a method to reorganize data blocks in real time for online addition/removal of disk drives [12]. Additionally, a flexible rate-control mechanism between clients and the server supports both variable and constant bitrate media types [13]. Further included are techniques to resolve stream contentions at the server for ensuring inter-stream synchronization as proposed in [14], as well as optimization techniques such as Super-Streaming [15]. Network congestion control has been investigated extensively by many researchers. In our Yima architecture, we assume that the network provides enough streaming bandwidth.

Yima follows the open industry standards proposed by the Internet Streaming Media Alliance (ISMA; http://www.ism-alliance.org) with some extensions for higher bitrate media. It supports the RTP and RTSP communication standards for IP-based networks. Content-wise, Yima can stream MPEG-1, MPEG-2, and MPEG-4 video formats. The clients can be either off-the-shelf QuickTime

players or our own Windows and Linux clients that handle advanced multi-channel audio and video HDTV playback [9].

The Yima system has been evaluated in many different networking environments. For example, when broadband first became available we successfully streamed NTSC-quality MPEG-4 content from a Yima server located on the University of South California (USC) campus to a residential location connected via ADSL [16]. Yima is also the streaming engine of the Remote Media Immersion (RMI) system developed at the Integrated Media Systems Center (IMSC) at USC. RMI is a platform to deliver very high-quality content such as high-definition video and immersive, multi-channel, uncompressed audio across different networking environments (e.g. Internet2) [17]. Additionally, we have reported on our design and evaluation of packet loss error recovery techniques with Yima based on extensive experiments in both LAN and WAN environments [18,19].

2.2. Multi-node architecture

The design of Yima is based on a *bipartite* model. From a client's viewpoint, the scheduler, the RTSP and the RTP server modules are all centralized on a single master node. Yima expands on decentralization by keeping only the RTSP module centralized (again from the client's viewpoint) and parallelizing the scheduling and RTP functions as shown in Figure 1. Hence, every node retrieves, schedules, and sends data blocks that are stored locally directly to the requesting client, thereby eliminating a potential bottleneck caused by routing all data through a single node. The elimination of this 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 members of a group.

Although the advantages of the bipartite design are clear, its realization introduces 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 (e.g. for lost packet retransmission requests). Second, to implement the single control point for client requests as well as the synchronized decentralized scheduler and RTP server for each node while maintaining load balance across all server nodes under all kinds of VBR stream load is a challenge. Third, a flow control mechanism is needed to prevent client buffer from overflow or starvation. Fourth, UDP-based RTP packets may get lost during transmission, hence a new efficient loss recovery scheme is required. Lastly, an effective on-line data reorganization technique is also desired.

2.3. Data placement and disk scheduling

There are two basic techniques to assign the data blocks of a media object, in a load balanced manner, to the magnetic disk drives that form the storage system: in a *round-robin* sequence [20], or in a *random* manner [21]. Traditionally, the round-robin placement utilizes a cycle-based approach for scheduling of resources to guarantee a continuous display, while the random placement utilizes a deadline-driven approach. In general, the round-robin approach provides high throughput with little wasted bandwidth for video objects that are retrieved sequentially. This approach can employ optimized disk scheduling algorithms (such as *elevator* [22]) and object replication and request migration [23] techniques to reduce the inherently high startup latency. The random approach has several benefits as described

in [24], such as (1) support for multiple delivery rates with a single server block size, (2) support for interactive applications, and (3) support for data reorganization during disk scaling [12].

One potential disadvantage of random data placement is the need for a large amount of meta-data: the location of each block 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 pseudo-random number generators, a seed value initiates a sequence of random numbers which can be reproduced by using the same seed. File objects are split into fixed-size blocks and each block is assigned to a random disk. Block retrieval is similar. Hence, Yima needs to store only the seed for each file object, instead of locations for every block, to compute the random number sequence.

2.4. Communication protocol

Each client maintains contact with one RTSP module for the duration of a session to relay control related information (such as `PAUSE` and `RESUME` commands). 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 server node running an RTSP module. 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.

2.5. Variable bitrate smoothing

In order to avoid bursty traffic and to accommodate variable bitrate media, the client sends slowdown or speedup signals to adjust the data transmission rate from the server. By periodically sending these signals to the Yima server, the client can receive a smooth flow of data by monitoring the amount of data in its buffer. If the amount of buffered data decreases (increases), the client will issue speedup (slowdown) requests. Thus, the amount of buffered data can remain close to constant to support the consumption of variable bitrate media. This mechanism will complicate the server scheduler logic, but the standard deviation of bursty traffic is reduced by up to 81% as demonstrated in [13].

2.6. Transmission error recovery

There has been considerable work in the area of error recovery techniques that can be applied to real-time streaming applications. Example techniques include error concealment, forward error correction (FEC), and retransmission-based error control [18]. To solve the error recovery problem in Yima's fully distributed *bipartite* architecture, we utilize a retransmission-based error control (RBEC) mechanism[§]. Because data are randomly placed and all server nodes send data to clients independently, a client may

[§] Another possible solution is the use of forward error correction (FEC). However, FEC always adds a constant percentage of bandwidth overhead irrespective of the network condition. As pointed out by Dempsey *et al.* [25], if the packet loss rate is very low and timely retransmission can be performed with a high probability of success, an RBEC approach is an attractive solution. It imposes little overhead on network resources and can be used in conjunction with other error control schemes, such as FEC or error concealment.

not know which server node to ask for a lost packet retransmission. With RBEC, 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. We term these local sequence numbers (LSNs) as opposed to the global sequence number (GSN) that orders all packets. This mechanism requires packets to contain an LSN along with a GSN. Experiments [18,19] show that the clients need little computation to locate missing packets, which enables Yima to utilize the benefits of random data placement in cluster environments. Please note that our proposed technique can be combined with other existing error control techniques, such as FEC and error concealment to support both unicast and multicast applications. A more extended discussion of our proposed technique can be found elsewhere [18].

2.7. Data reorganization

Yima incorporates a unique online storage scalability feature for the addition of disks to increase storage and/or bandwidth or the deletion of disks when either capacity needs to be conserved or old disk drives are retired. Our approach is an efficient randomized technique to reorganize continuous media blocks, called SCADDAR [12]. With SCADDAR, disk additions or deletions can be done online with minimum overhead in terms of the number of media blocks needing to be redistributed while still maintaining the randomized uniform distribution of the blocks. The SCADDAR approach is based on a series of REMAP functions which can derive the location of a new block using only its original location as a basis.

We have conducted streaming experiments while performing disk scaling operations, such as removing a disk. During the switch-over period, the system continues to perform well [26]. Note that *a disk removal* differs from *a disk crash* in that a disk crash generally happens unexpectedly. If it is possible to predict when a disk might crash in the future (for example, through Self-Monitoring, Analysis and Reporting Technology, SMART), a disk removal operation can be initiated in advance. Otherwise, a fault tolerant design is required to provide continuous streaming service while surviving disk crashes [10,27].

3. SCALABILITY EXPERIMENTS

We have implemented the features described in the previous section in our Yima streaming media prototype. It was our goal to design and implement an architecture that would scale in performance from small to large systems. In this section we assess its scalability in an end-to-end test environment.

A computer system is scalable if it can *scale up* to accommodate performance demands and/or *scale down* to reduce cost [28]. Scalability can be classified into two categories: (1) *size scalability*: scaling up by increasing the number of server nodes; (2) *scale up in resources*: scaling up by adding resources such as memory, cache, disks, or network bandwidth. We present the results of two sets of experiments. First, we compare a single node server with two different network interface bandwidths: 100 Mb s^{-1} versus 1 Gb s^{-1}. These experiments show that the system can *scale up in resources*.

In the second set of experiments we increased a server cluster from one to two and then four nodes. The goal of every cluster architecture is to achieve close to a linear performance scale up when system resources are increased. However, achieving this goal in a real-world implementation is very challenging. Our experiments show the *size scalability* of the Yima system. We start by describing our measurement methodology. Table I lists the terms used in this section.

Table I. List of terms used repeatedly in this section and their respective definitions.

Term	Definition
$\mathcal{N}$	The number of concurrent clients supported by Yima server
$\mathcal{N}_{\max}$	The maximum number of sustainable, concurrent clients
μ_{idle}	Idle CPU in percentage
μ_{system}	System (or kernel) CPU load in percentage
μ_{user}	User CPU load in percentage
B_{avgNet}	Average network bandwidth per client (Mb s^{-1})
B_{net}	Network bandwidth (Mb s^{-1})
B_{disk}	The amount of movie data accessed from disk per second (termed disk bandwidth) (MB s^{-1})
B_{cache}	The amount of movie data accessed from server cache per second (termed cache bandwidth) (MB s^{-1})
$B_{\text{avgNet}}[i]$	The B_{avgNet} measured for the ith server node in a multi-node experiment
$B_{\text{net}}[i]$	The B_{net} measured for the ith server node in a multi-node experiment
$B_{\text{disk}}[i]$	The B_{disk} measured for the ith server node in a multi-node experiment
$B_{\text{cache}}[i]$	The B_{cache} measured for the ith server node in a multi-node experiment
$R_{\Delta r}$	The number of rate changes per second

3.1. Methodology of measurement

One of the challenges when stress testing a high-performance streaming media server is the necessary support of a large number of clients. For a realistic test environment, these clients should not be simulated, but rather be real viewer programs that run on various machines across a network. To keep the number of client machines manageable we ran several client programs on each machine. Since decompressing multiple MPEG-2 encoded DVD-quality streams requires a very high CPU performance, we changed our client software to not actually decompress the media streams. Such a client is identical to its real counterpart in every respect, except that it does not render any video or audio. Instead, this emulation client consumes data according to a movie trace data file, which contains the pre-recorded consumption behavior of a real client with respect to a particular movie. Thus, by changing the movie trace file, each emulation client can behave like, for example, a DVD stream (5 Mb s^{-1}, VBR), an HDTV stream (20 Mb s^{-1}, CBR), or an MPEG-4 stream (800 Kb s^{-1}, VBR). For all the experiments in this section, we chose trace data from the DVD movie *Twister* (see Figure 2) as the consumption load. The average bandwidth requirement for this DVD movie is approximately 5.33 Mb s^{-1}. For each experiment, we started clients in a staggered manner (the incoming streaming request arrival rate is 0.5 per minute). On the server side, we recorded the following statistics every 2 s: CPU load (μ_{idle}, μ_{system} and μ_{user}), disk bandwidth (B_{disk}), cache bandwidth (B_{cache}), $R_{\Delta r}$, the total network bandwidth (B_{net}) for all clients, the number of clients served, and the average network bandwidth per client, B_{avgNet}.

The server nodes were run with disabled admission control policies to allow us to push them into overload and hence find the maximum sustainable throughput used by many client sessions.

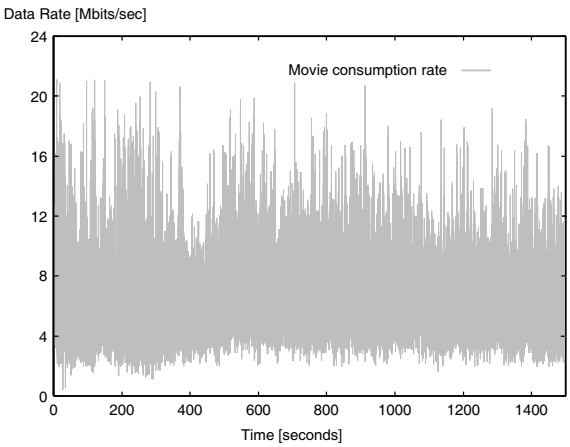

Figure 2. The consumption rate of a segment of the movie *Twister* encoded with a variable bitrate MPEG-2 algorithm.

Therefore, client starvation would occur when the number of sessions $\mathcal{N}$ increased beyond a threshold. We defined that threshold as the maximum number of sustainable, concurrent sessions $\mathcal{N}_{\max}$. Specifically, this threshold marks the point where certain server system resources reach full utilization and become a bottleneck, for example the network bandwidth, the disk bandwidth or the CPU load.

We first assess the Yima server performance with two different network connections, and then we evaluate our prototype in a cluster scale-up experiment.

3.2. Network scale-up experiments

3.2.1. Experimental setup

We tested a single node server with two different network connections: 100 Mb s^{-1} and 1 Gb s^{-1} Ethernet. Figure 1 illustrates our experimental setup. In both cases, the server consists of a single Pentium III 933 MHz PC with 256 MB of memory. The PC is connected to an Ethernet switch (model Cabletron 6000) via a 100 Mb s^{-1} network interface for the first experiment and a 1 Gb s^{-1} network interface for the second experiment. Movies are stored on a 73 GB Seagate Cheetah disk drive (model ST373405LC). The disk is attached through an Ultra2 low-voltage differential (LVD) SCSI connection that can provide 80 MB s^{-1} throughput. RedHat Linux 7.2 is used as the operating system. The clients are based on several Pentium III 933 MHz PCs, which are connected to the same Ethernet switch via 100 Mb s^{-1} network interfaces. Each PC can support up to 10 concurrent MPEG-2 DVD emulation clients (with 5.3 Mb s^{-1} stream consumption rate for each client).

3.2.2. *Experimental results*

Figure 3 shows the server measurement results for both sets of experiments (100 Mb s^{-1} and 1 Gb s^{-1}) in two columns. Figures 3(c) and (d) present the per stream bandwidth[¶] B_{avgNet} with respect to the number of clients, $\mathcal{N}$. Figure 3(c) shows that, for a 100 Mb s^{-1} network connection, B_{avgNet} remains steady (between 5.3 and 6 Mb s^{-1}) when $\mathcal{N}$ is less than 13; after 13 clients, B_{avgNet} decreases steadily and falls below 5.3 Mb s^{-1} (depicted with a dashed horizontal line), which is the average consumption bandwidth of our test movie. Note that the horizontal dashed line intersects with the B_{avgNet} curve at approximately 12.8 clients. Thus, we consider 12 as the maximum number of clients, $\mathcal{N}_{\text{max}}$, supportable by a 100 Mb s^{-1} networking interface. An analogous result can be observed in Figure 3(d), indicating a maximum throughput of $\mathcal{N}_{\text{max}} = 35$ with a 1 Gb s^{-1} network connection.

Figures 3(a) and (b) show the CPU utilization as a function of $\mathcal{N}$ for 100 Mb s^{-1} and 1 Gb s^{-1} network connections. Both figures contain two curves: μ_{system} (kernel space CPU utilization) and $\mu_{\text{system}} + \mu_{\text{user}}$ (combined user and kernel space CPU utilization). As expected, the CPU load (both μ_{system} and μ_{user}) increases steadily as $\mathcal{N}$ increases. With the 100 Mb s^{-1} network connection, the CPU load reaches its maximum at 40% with 12 clients, which is exactly $\mathcal{N}_{\text{max}}$ suggested by Figure 3(c) (vertical dashed line). Similarly, for 1 Gb s^{-1}, the CPU load levels off at 80% where $\mathcal{N}_{\text{max}} = 35$ clients. Note that in both experiments, μ_{system} accounts for more than two thirds of the maximum CPU load.

Yima implements a simple yet flexible caching mechanism in the file I/O module (Figure 1). Movie data are loaded from disks as blocks (e.g. 1 MB). These blocks are organized into a shared block list maintained by the file I/O module in memory. For each client session, there are at least two corresponding blocks on this list. One is the block currently used for streaming, and the other is the prefetched, next block. Some blocks may be shared because the same data are used by more than one client session simultaneously. Therefore, a session counter is implemented for each block. When a client session requests a block, the file I/O module checks the shared block list first. If the block is found, then the corresponding block counter will be incremented and the block made available; otherwise, the requested block will be fetched from disk and added to the shared block list (with its counter set to one). We define the cache bandwidth, B_{cache}, as the amount of data accessed from the shared block list (server cache) per second.

Figures 3(e) and (f) show B_{disk} and B_{cache} as a function of $\mathcal{N}$ for 100 Mb s^{-1} and 1 Gb s^{-1} network connections. In both experiments, the $B_{\text{disk}} + B_{\text{cache}}$ curves increase linearly until $\mathcal{N}$ reaches its respective $\mathcal{N}_{\text{max}}$ (12 for 100 Mb s^{-1} and 35 for 1 Gb s^{-1}), and they level off beyond those points. For the 100 Mb s^{-1} network connection, $B_{\text{disk}} + B_{\text{cache}}$ level off at around 8.5 MB s^{-1}, which equals the 68 Mb s^{-1} peak rate, B_{net}, in Figure 3(i) with $\mathcal{N} = \mathcal{N}_{\text{max}}$. Similarly, for the 1 Gb s^{-1} network connection, $B_{\text{disk}} + B_{\text{cache}}$ level off at 25 MB s^{-1}, which corresponds to the 200 Mb s^{-1} maximum, B_{net}, in Figure 3(j) with $\mathcal{N} = \mathcal{N}_{\text{max}} = 35$. In both cases, B_{cache} contributes little to $B_{\text{disk}} + B_{\text{cache}}$ when $\mathcal{N}$ is less than 15. For $\mathcal{N} > 15$, caching becomes increasingly effective. For example, with 1 Gb s^{-1} network connection, B_{cache} accounts for 20–30% of $B_{\text{disk}} + B_{\text{cache}}$ with $\mathcal{N}$ between 35 and 40. This is because for higher $\mathcal{N}$, the probability that the same cached block is accessed by more than one client increases. Intuitively, caching is more effective with large $\mathcal{N}$.

[¶]In the paper, we use 'per stream bandwidth' and 'per client bandwidth' interchangeably.

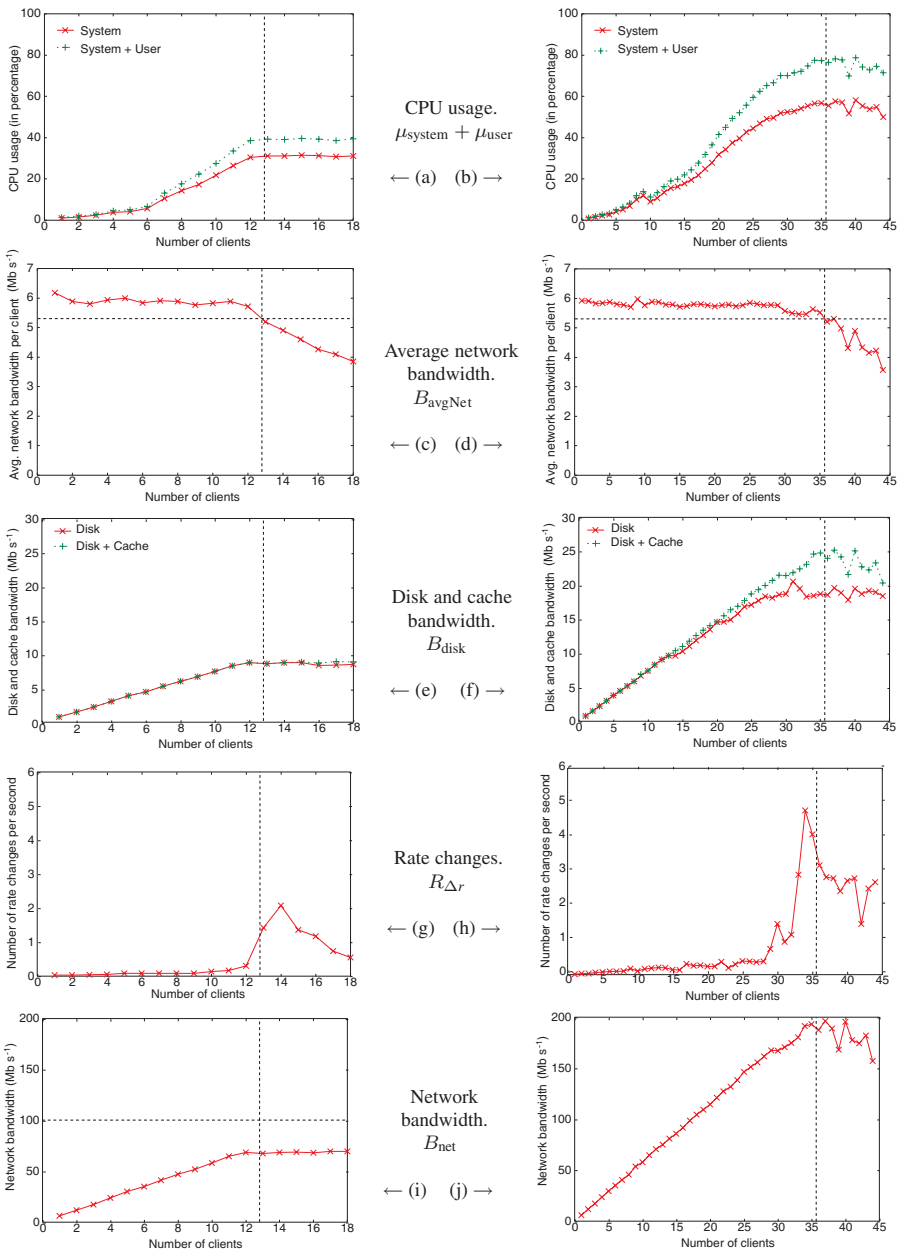

Figure 3. Yima single-node server performance with 100 Mb s^{-1} (left column) versus 1 Gb s^{-1} (right column) network connection.

Figures 3(i) and (j) show the relationship of B_{net} and $\mathcal{N}$ for both network connections. Both figures nicely complement Figures 3(e) and (f). With the 100 Mb s^{-1} connection, B_{net} increases steadily with respect to $\mathcal{N}$ until it levels off at 68 Mb s^{-1} with $\mathcal{N}_{max}$ (12 clients). For the 1 Gb s^{-1} connection, the results are similar except that B_{net} levels off at 200 Mb s^{-1} with $\mathcal{N} = 35$ ($\mathcal{N}_{max}$ for 1 Gb s^{-1} setup). Note that the horizontal dashed line in Figure 3(i) represents the theoretical bandwidth limit for the 100 Mb s^{-1} setup.

Figures 3(g) and (h) show the number of rate adjustments $R_{\Delta r}$ with respect to $\mathcal{N}$ for 100 Mb s^{-1} and 1 Gb s^{-1} network connections. Both figures suggest a similar trend: there exists a threshold T where, if $\mathcal{N} < T$, $R_{\Delta r}$ is quite small (approximately 1 per second); otherwise, $R_{\Delta r}$ increases significantly to 2 for 100 Mb s^{-1} connection and 5 for the 1 Gb s^{-1} connection. With the 100 Mb s^{-1} setup, T is reached at approximately 12 clients. For the 1 Gb s^{-1} case, the limit is 33 clients. In general, T roughly matches $\mathcal{N}_{max}$ for both experiments. Note that in both cases, for $\mathcal{N} > T$, at some point $R_{\Delta r}$ begins to decrease. This is due to client starvation. Under these circumstances such clients send a request for the maximum stream transmission rate. Because this maximum cannot be increased, no further rate changes are sent.

Note that in both the 100 Mb s^{-1} and 1 Gb s^{-1} experiments, $\mathcal{N}_{max}$ is reached when some system resources become a bottleneck. For the 100 Mb s^{-1} setup, Figures 3(a) and (e) suggest that the CPU and disk bandwidth are not the bottleneck, because neither of them reaches more than 50% utilization. On the other hand, Figure 3(i) indicates that the network bandwidth, B_{net}, reaches approximately 70% utilization for $\mathcal{N} = 12$ ($\mathcal{N}_{max}$ for 100 Mb s^{-1} setup), and hence is most likely the bottleneck of the system. For the 1 Gb s^{-1} experiment, Figures 3(f) and (j) show that the disk and network bandwidth are not the bottleneck. Conversely, Figure 3(b) shows that the CPU is the bottleneck of the system because it is heavily utilized ($\mu_{system} + \mu_{user}$ is around 80%) for $\mathcal{N} = 35$ ($\mathcal{N}_{max}$ for the 1 Gb s^{-1} setup).

3.3. Server scale-up experiments

3.3.1. Experimental setup

To evaluate the cluster scalability of the Yima server, we conducted three sets of experiments. The server cluster consists of multiple rack-mountable Pentium III 866 MHz PCs with 256 MB of memory. We increased the number of server PCs from one to two to four, respectively, for the scale-up experiments. The server PCs are connected to an Ethernet switch (model Cabletron 6000) via 100 Mb s^{-1} network interfaces. Movies are striped over several 18 GB Seagate Cheetah disk drives (model ST118202LC, one per server node), which are attached through an Ultra2 low-voltage differential (LVD) SCSI connection that can provide 80 MB s^{-1} throughput. RedHat Linux 7.0 is used as the operating system and the client setup is the same as in Section 3.2.

3.3.2. Experimental results

The results for a single-node server have already been reported in Section 3.2. Here we will not repeat them, but refer to them where appropriate. Figure 4 shows the results for the two and four-node experiments in two columns.

Figures 4(c) and (d) present the measured per stream bandwidth B_{avgNet} as a function of $\mathcal{N}$. Figure 4(c) shows two curves representing two nodes: $B_{avgNet}[1]$ and $B_{avgNet}[1] + B_{avgNet}[2]$.

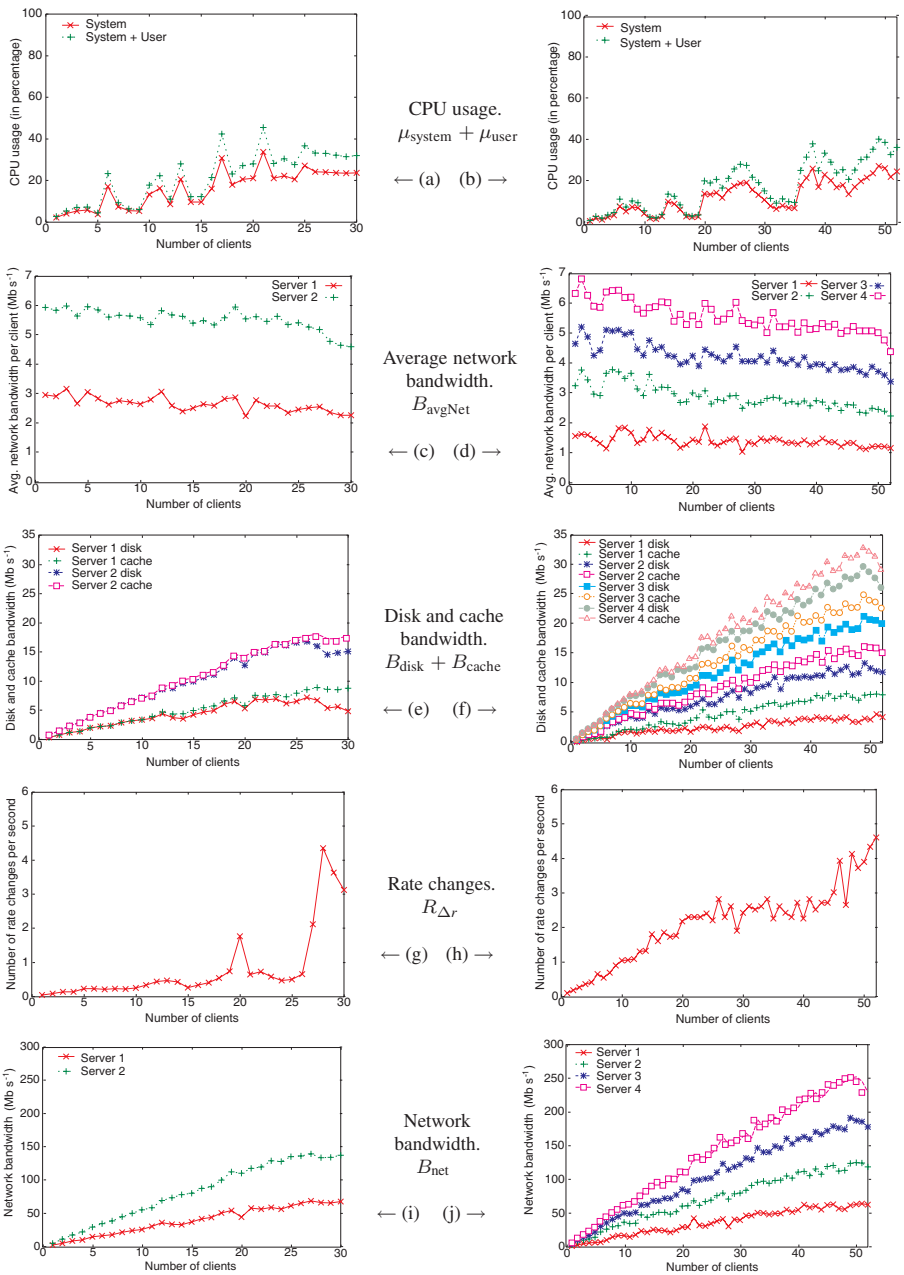

Figure 4. Yima two-node (left column) versus four-node (right column) server performance. The curves in (c, d, e, f, i, j) are cumulative. For example, in (c), 'Server 2' refers to 'Server 1+Server 2'.

Similarly, Figure 4(d) shows four curves: $B_{\text{avgNet}}[1]$, $B_{\text{avgNet}}[1]+B_{\text{avgNet}}[2]$, $B_{\text{avgNet}}[1]+B_{\text{avgNet}}[2]+B_{\text{avgNet}}[3]$ and $B_{\text{avgNet}}[1]+B_{\text{avgNet}}[2]+B_{\text{avgNet}}[3]+B_{\text{avgNet}}[4]$. Note that each server node contributes roughly the same share to the total bandwidth B_{avgNet}, i.e. 50% in case of the two-node system and 25% for the four-node cluster. This illustrates how well the nodes are load balanced within our architecture. Recall that the same software modules are running on every server node, and the movie data blocks are evenly distributed by the random data placement technique. Similarly as in Figures 3(c) and (d), the maximum number of supported clients can be derived as $\mathcal{N}_{\max}=25$ for two nodes and $\mathcal{N}_{\max}=48$ for four nodes. Including the previous results from one node (see the 100 Mb s^{-1} experimental results in Figure 3), with two and four nodes the maximum number of client streams $\mathcal{N}_{\max}$ are 12, 25, and 48, respectively, which represents an almost ideal linear scale-up.

Figures 4(a) and (b) show the average CPU utilization on two and four-node servers as a function of $\mathcal{N}$. In both figures, μ_{system} and $\mu_{\text{system}}+\mu_{\text{user}}$ are depicted as two curves with similar trends. For two nodes the CPU load ($\mu_{\text{system}}+\mu_{\text{user}}$) increases gradually from 3% with $\mathcal{N}=1$ to approximately 38% with $\mathcal{N}=25$ ($\mathcal{N}_{\max}$ for this setup), and then levels off. With four nodes, the CPU load increases from 2% with $\mathcal{N}=1$ to 40% with $\mathcal{N}=48$ ($\mathcal{N}_{\max}$ for this setup). Note that the curves in both figures are not very smooth, which might be due to server logging activities and the measurement interval variations.

Figures 4(e) and (f) show $B_{\text{disk}}+B_{\text{cache}}$. The four curves presented in Figure 4(e) cumulatively show the disk and cache bandwidth for two nodes: $B_{\text{disk}}[1]$, $B_{\text{disk}}[1]+B_{\text{cache}}[1]$, $B_{\text{disk}}[2]+B_{\text{disk}}[1]+B_{\text{cache}}[1]$, and $B_{\text{disk}}[2]+B_{\text{cache}}[2]+B_{\text{disk}}[1]+B_{\text{cache}}[1]$. The curves exhibit the same trend as shown in Figures 3(e) and (f) for a single node. $B_{\text{disk}}+B_{\text{cache}}$ reach a peak value of 17 MB s^{-1} with $\mathcal{N}=\mathcal{N}_{\max}$ for the two-node setup and 32 MB s^{-1} for the four-node experiment. Note that $B_{\text{disk}}+B_{\text{cache}}$ for four nodes is nearly doubled compared with two nodes, which is double that of the one-node setup. In both cases, each server contributes approximately the same share to the total of $B_{\text{disk}}+B_{\text{cache}}$, illustrating the balanced load in the Yima cluster. Furthermore, similar to Figures 3(e) and (f), caching effects are more pronounced with large $\mathcal{N}$ in both the two- and four-node experiments.

Figures 4(i) and (j) show the achieved network throughput B_{net}. Again, Figures 4(i) and (j) nicely complement Figures 4(e) and (f). For example, the peak rate, B_{net}, of 136 Mb s^{-1} for the two-node setup is equal to the 17 MB s^{-1} peak rate of $B_{\text{disk}}+B_{\text{cache}}$. Each node contributes equally to the total served network throughput.

Finally, Figures 4(g) and (h) show the number of rate changes, $R_{\Delta r}$, that are sent to the server cluster by all clients. Similarly to the one-node experiment, for the two-node server $R_{\Delta r}$ is very small (approximately 1 per second) when $\mathcal{N}$ is less than 26, and increases significantly above this threshold. For the four-node experiment, a steady increase is recorded when $\mathcal{N}$ is less than 26; after that it remains constant at 2.5 for $\mathcal{N}$ between 27 and 45, and finally $R_{\Delta r}$ increases for $\mathcal{N}$ beyond 45. Note that for all experiments, with $\mathcal{N}<\mathcal{N}_{\max}$, the rate change messages $R_{\Delta r}$ generate negligible network traffic and server processing load. Therefore, our MTFC smoothing technique [13] is well suited for a scalable cluster architecture.

Overall, the experimental results presented here demonstrate that our current architecture scales linearly to four nodes while at the same time achieving an impressive performance on each individual node. Furthermore, the load is nicely balanced and remains such, even if additional nodes or disks are added to the system (with SCADDAR). We expect that high-performance Yima systems can be built with eight and more nodes. When higher performing CPUs are used (beyond our dated 866 and 933 MHz Pentium IIIs) each node should be able to eventually reach 300–800 Mb s^{-1}.

With such a configuration almost any currently available network could be saturated (e.g. 8×800 Mb $s^{-1} = 6.4$ Gb s^{-1} effective bandwidth).

4. CONCLUSIONS AND FUTURE WORK

Yima is a second-generation scalable real-time streaming architecture that builds upon results from first-generation research prototypes, and is compatible with industry standards. The fully distributed design of Yima yields a linear scale up in performance, which we successfully verified through several experiments with realistic end-to-end setups.

We plan to extend Yima in three ways. First, we are working toward enabling it with scalable real-time recording capabilities [29]. Second, we plan to extend our experiments to investigate distributed clients from more than one Yima server cluster. By co-locating two Yima server clusters at off-campus locations and two other 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 [30] that we would like to incorporate, experiment with and extend. Finally, we performed some studies on supporting other media types, in particular the immersive sensor data streams [31]. Our next step would be to store and stream immersive sensor data.

ACKNOWLEDGEMENTS

This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC) and IIS-0082826, and unrestricted cash gifts from NCR, Microsoft, Intel, Hewlett-Packard and the Okawa and Lord Foundations.

REFERENCES

1. Shahabi C, Barish G, Ellenberger B, Jiang N, Kolahdouzan M, Nam A, Zimmermann R. Immersidata management: Challenges in management of data generated within an immersive environment. *Proceedings of the International Workshop on Multimedia Information Systems*, October 1999.
2. Tobagi F, Pang J, Baird R, Gang M. Streaming RAID-A disk array management system for video files. *First ACM Conference on Multimedia*, August 1993. ACM Press, 1993.
3. Laursen A, Olkin J, Porter M. Oracle media server: Providing consumer based interactive access to multimedia data. *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM Press, 1994; 470–477.
4. Hsieh J, Liu J, Du D, Ruwart T, Lin M. Experimental performance of a mass storage system for video-on-demand. *Journal of Parallel and Distributed Computing* (*Multimedia Systems and Technology*) 1995; **30**(2):147–167.
5. Bolosky WJ, Barrera JS, Draves RP, Fitzgerald RP, Gibson GA, Jones MB, Levi SP, Myhrvold NP, Rashid RF. The Tiger video fileserver. *6th Workshop on Network and Operating System Support for Digital Audio and Video*, Zushi, Japan, April 1996. ACM Press, 1996.
6. Martin C, Narayan PS, Özden B, Rastogi R, Silberschatz A. The Fellini multimedia storage server. *Multimedia Information Storage and Management*, ch. 5, Chung SM (ed.). Kluwer Academic Publishers: Boston, MA, 1996.
7. Ghandeharizadeh S, Zimmermann R, Shi W, Rejaie R, Ierardi D, Li T. Mitra: A scalable continuous media server. *Kluwer Multimedia Tools and Applications* 1997; **5**(1):79–108.
8. Muntz R, Santos J, Berson S. RIO: A real-time multimedia object server. *ACM Sigmetrics Performance Evaluation Review* 1997; **25**(2):29–35.
9. Shahabi C, Zimmermann R, Fu K, Yao S-YD. Yima: A second generation continuous media server. *IEEE Computer* 2002; **35**(6):56–64.
10. Zimmermann R, Ghandeharizadeh S. HERA: Heterogeneous extension of RAID. *Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, Las Vegas, NV, 26–29 June 2000. CSREA Press, 2000.

11. Zimmermann R, Ghandeharizadeh S. Continuous display using heterogeneous disk-subsystems. *Proceedings of the Fifth ACM Multimedia Conference*, Seattle, WA, 9–13 November 1997. ACM Press, 1997; 227–236.
12. Goel A, Shahabi C, Yao S-YD, Zimmermann R. SCADDAR: An efficient randomized technique to reorganize continuous media blocks. *Proceedings of the 18th International Conference on Data Engineering*, February 2002. IEEE Computer Society Press, 2002; 473–482.
13. Zimmermann R, Shahabi C, Fu K, Jahangiri M. A multi-threshold online smoothing technique for variable rate multimedia streams. *Kluwer Multimedia Tools and Applications Journal* (in press).
14. Shahabi C, Ghandeharizadeh S, Chaudhuri S. On scheduling atomic and composite continuous media objects. *Transactions on Knowledge and Data Engineering* 2002; **14**(2):447–455.
15. Shahabi C, Alshayeji M. Super-streaming: A new object delivery paradigm for continuous media servers. *Journal of Multimedia Tools and Applications* 2000; **11**(1):129–155.
16. Zimmermann R, Fu K, Shahabi C, Yao S-YD, Zhu H. Yima: Design and evaluation of a streaming media system for residential broadband services. *VLDB 2001 Workshop on Databases in Telecommunications (DBTel 2001)*, Rome, Italy, September 2001. Springer, 2001.
17. Zimmermann R, Kyriakakis C, Shahabi C, Papadopoulos C, Sawchuk AA, Neumann U. The remote media immersion system. *IEEE MultiMedia* 2004; **11**(2):48–57.
18. Zimmermann R, Fu K, Liao F. Retransmission-based error control for scalable streaming media systems. *SPIE Journal of Electronic Imaging* (in press).
19. Zimmermann R, Fu K, Nahata N, Shahabi C. Retransmission-based error control in a many-to-many client-server environment. *Proceedings of the SPIE Conference on Multimedia Computing and Networking (MMCN)*, Santa Clara, CA, 29–31 January 2003. The International Society for Optical Engineering and The Society for Imaging Science and Technology, 2003.
20. Berson S, Ghandeharizadeh S, Muntz R, Ju X. Staggered striping in multimedia information systems. *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM Press, 1994.
21. Santos JR, Muntz RR. Performance analysis of the RIO multimedia storage system with heterogeneous disk configurations. *ACM Multimedia Conference*, Bristol, U.K. ACM Press, 1998.
22. Seltzer M, Chen P, Ousterhout J. Disk scheduling revisited. *Proceedings of the 1990 Winter USENIX Conference*, Washington DC. Usenix Association, 1990; 313–324.
23. Ghandeharizadeh S, Kim SH, Shi W, Zimmermann R. On minimizing startup latency in scalable continuous media servers. *Proceedings of Multimedia Computing and Networking 1997*, February 1997; 144–155.
24. Santos JR, Muntz R, Ribeiro-Neto B. Comparing random data allocation and data striping in multimedia servers. *Proceedings of ACM SIGMETRICS 2000*, June 2000. ACM Press, 2000; 44–55.
25. Dempsey BJ, Liebeherr J, Weaver AC. On retransmission-based error control for continuous media traffic in packet-switching networks. *Computer Networks and ISDN Systems* 1996; **28**(5):719–736.
26. Shahabi C, Zimmermann R. Design and development of a scalable end-to-end streaming architecture. *Handbook for Video Databases: Design and Applications*, ch. 32, Furht B, Marques O (eds.). CRC Press LLC: Boca Raton, FL, 2003.
27. Zimmermann R, Ghandeharizadeh S. Highly available and heterogeneous continuous media storage systems. *IEEE Transactions on Multimedia Journal* 2004; (December).
28. Hwang K, Xu Z. *Scalable Parallel Computing: Technology, Architecture, Programming*. McGraw-Hill: New York, 1998.
29. Zimmermann R, Fu K, Ku W-S. Design of a large scale data stream recorder. *Proceedings of the 5th International Conference on Enterprise Information Systems (ICEIS 2003)*, Angers, France, 23–26 April 2003. ICEIS Press, 2003. Available at: http://www.iceis.org/.
30. Shahabi C, Kashani FB. Decentralized resource management for a distributed continuous media server. *IEEE Transactions on Parallel and Distributed Systems* 2002; **13**(6):710–727.
31. Shahabi C. AIMS: An immersidata management system. *VLDB First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, Asilomar, CA, 5–8 January 2003. VLDB Endowment, 2003.

