Divyesh Jadav and Alok Choudhary
Syracuse University

Designing and Implementing High-Performance Media-on-Demand Servers

High-performance computers are the best choice for on-demand multimedia servers. However, implementing them poses serious challenges.

Of the many potential interactive multimedia applications (see the sidebar), *media-on-demand* has generated the most excitement. MOD will let users receive services from remote resources interactively, at their own pace. The quintessential example of these services is *video-on-demand* (VOD). Consumers will be able to order and view movies of their choice, at their convenience, from their home, using a remote control. In effect, they would possess a virtual VCR with all the traditional functions of fast-forward, pause, play, and so on, without having to go to the video rental store to get the cassette.

This scenario, though achievable, remains just a concept. Although recent advances in networking, processing, and storage have brought this concept closer to reality, large-scale deployment of MOD services remains a complex problem. We believe that using high-performance computers as MOD servers offers the best solution. However, implementing these servers poses its own challenges. This article will examine such a server's requirements and implementation problems.

The server's place in the system

Given that an Asynchronous Transfer Mode (ATM) network will form the backbone of an interactive MOD system, strategically located servers will store the data. Figure 1 shows one possible configuration. At the top of the system's hierarchy is the consumer. The consumer's interface device, a multimedia terminal, connects to a medium-capacity neighborhood

1063-6552/95/$4.00 © 1995 IEEE

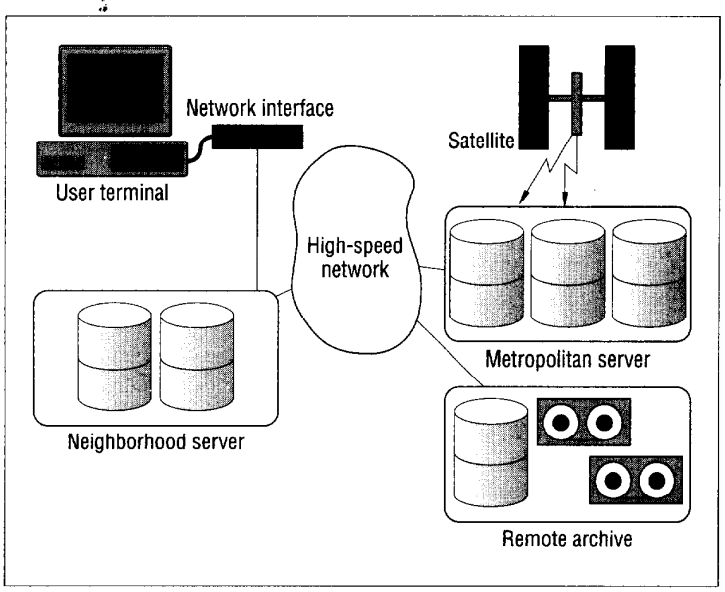

Figure 1. A hierarchical media-on-demand system.

server. The neighborhood server downloads requested programs from a more powerful metropolitan server, which may store hundreds of titles. If the metropolitan server does not carry a requested title, the neighborhood server will have to fetch the title from a remote server in another city. The remote server could be a similar (metropolitan) server or a central archive storing thousands of titles. (For more information on network and consumer requirements, see the sidebar, "Other requirements for a media-on-demand system.")

The consumer's cost for retrieving multimedia data streams increases proportionately with the data's depth in the hierarchy. Like a computer's memory hierarchy, this hierarchy of networked servers caches the requested data closest to the consumer. This procedure minimizes costly fetches from storage agents that are lower in the hierarchy.

Interactive multimedia applications

We can classify multimedia services by their degree of interactivity. For example, A. Gelman and his colleagues classify video-supply services into broadcast, pay-per-view, quasi video-on-demand, near VOD, and true VOD, in ascending order of interactivity.[1] Although VOD is the most visible interactive multimedia application, there are many other possible applications (see Table A).

Interactive multimedia services such as entertainment-on-demand will give consumers tremendous flexibility:

- Users will have more viewing choices; they won't be restricted to the programs shown on a few channels. Moreover, they won't be forced to watch commercials.
- Users will control when they view. They won't need to tailor their viewing schedules around the broadcasting schedule.
- Users will not have to travel to use a service. This has tremendous implications for shopping and education.
- Users will proceed at their own pace. For example, digitized lectures can let students review difficult topics and skip simple ones.

Interactive multimedia applications fall into two broad categories: *persistent* and *nonpersistent*. Persistent refers to the nature of the data at the source or destination. For example, movies-on-demand is persistent (as are all media-on-demand applications), and videoconferencing is nonpersistent.

Reference

1. A. Gelman et al., "A Store and Forward Architecture for Video-on-Demand Service, *Proc. IEEE ICC*, Vol. 27, No. 3, 1991, pp. 1–5.

Table A. Some applications of interactive multimedia technology.

Application	Description
Entertainment-on-demand	Viewing multimedia presentations for pleasure, using VCR-like controls. Includes movies-on-demand, news-on-demand, and interactive video games.
Home shopping	Browsing catalogs and selecting, ordering, and paying for merchandise interactively.
Distance learning	Undergoing self-education by designing content and pace of "lectures."
Digital libraries	Browsing, reading, and "checking out" (downloading) conventional library materials in digital form.
Home office	Uploading and downloading phone messages, faxes, memos, files, and folders from home.

Other requirements of a media-on-demand system

Media-on-demand systems also require specialized hardware for data transport and presentation.

Transport

The first requirement for a fully operational large-scale distributed multimedia environment is a high-speed wide-area network. The Internet's growing popularity suggests that it could be the backbone of the information superhighway. (It grew by 81% in 1994 to some 3.5 million hosts in 154 countries.[1]) However, it is woefully inadequate for the high volume of multimedia traffic.

Network protocols that have been candidates for carrying multimedia data include the 100-Mbps Ethernet standard, Distributed Queue Dual Bus (DQDB), Fiber-Distributed Data Interface (FDDI), and ATM. The first three have bandwidths of the order of 100 Mbps. Although this is an improvement over Ethernet's original 10 Mbps, it is still inadequate for high-volume multimedia data. ATM is rapidly emerging as the front-runner.[2]

Presentation

Because MOD servers will probably store data in compressed form, the consumer (user) will require sophisticated equipment. Figure A shows one possible configuration. Data decompression occurs at the user's *multimedia terminal.* The terminal is an intelligent computer with hardware such as a microphone, a high-resolution graphics display, stereo speakers, and a cable decoder.

The *cable decoder* is the interface to the high-speed wide-area network. It incorporates tens of Kbytes of buffer space and compression and decompression hardware.[3] Generally, a user session may consist of multiple media streams. *Streaming* restores the temporal relationship among the elements of a single stream before delivery to a multimedia device at the terminal. *Synchronization* restores the temporal relationship among multiple streams. Accordingly, the decoder connects through buffers to *stream handlers*, and the entire unit operates under the control of a *synchronizing/streaming manager*. The stream handler consists of devices such as *codecs* (coder/decoders) and associated software. The decompressed, streamed, and synchronized data are played out on the multimedia output devices. The user generates feedback in the form of interrupts, requests, and terminations through an interface control similar to the familiar TV remote control.

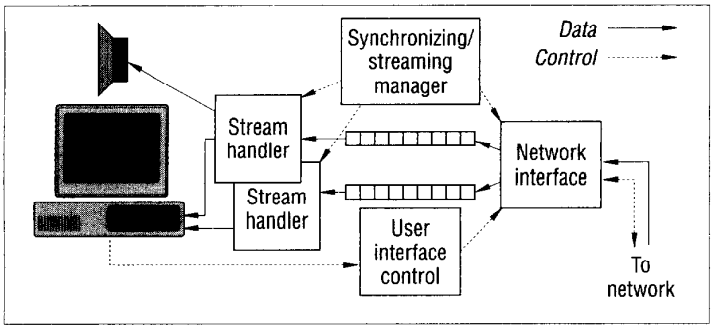

Figure A. Hardware required at user site.

References

1. S.J. Lowe, "Technology 1995: Data Communications," *IEEE Spectrum*, Vol. 32, No. 1, Jan. 1995, p. 26.
2. J. Lane, "ATM Knits Voice, Data on Any Net," *IEEE Spectrum*, Vol. 31, No. 2, Feb. 1994, pp. 42–45.
3. T.S. Perry, "Technology 1994: Consumer Electronics," *IEEE Spectrum*, Vol. 31, No. 1, Jan. 1994, pp. 30–34.

MOD server requirements

A workstation server can support only a few tens of simultaneous multimedia streams. Because of the large workload and multimedia data's inherent complexity (see the "Multimedia data" sidebar), a neighborhood or metropolitan server requires a more powerful machine, one that

- *has high storage capacity.* A 2-hour MPEG-1-encoded movie requires approximately 1 Gbyte of storage. MPEG-1 requires a minimum playback rate of 1.5 Mbits per second. The newer MPEG-2 standard, which will be used for most National Television Systems Committee (NTSC)-type full-motion video applications, requires a minimum playback rate of 4 Mbps. So, a 2-hour-long MPEG-2-encoded movie will require nearly 3.5 Gbytes of storage. Compressed HDTV-quality video will require even more storage. Thus, the server should be able to store at least hundreds of gigabytes of data.
- *stores a large number of titles.* If a neighborhood server serves 100 homes, each requiring independent connectivity, it should simultaneously store at least 100 different titles. If a server stores only a few titles, it may incur costly downloading from remote servers. The consumer will also experience increased response time.
- *can sustain a maximum number of streams.* The more streams a server can simultaneously source, the more consumers it can serve. The server can employ various caching optimizations when multiple consumers request the same data. So, the number of different streams that the server can simultaneously source is important.
- *delivers minimum response time.* A crucial factor that will determine the success of on-demand services is the response time that consumers experience. A consumer might tolerate a long response time during setup, but not when restarting a paused stream. Also, the difference between average and worst-case response times should be low.

Multimedia data

Multimedia data processing is difficult because such data differs markedly from the unimedia data (text) that conventional computers are built to handle.[1] Multimedia data's characteristics include

- *Multiple data streams.* A multimedia object can consist of text, audio, video, and image data. These data types have very different storage space and retrieval rate requirements. The design choices include storing data of the same type together, or storing data for the same object together. In either case, multimedia data adds a new dimension to the mechanisms used to store, retrieve, and manipulate the data.
- *Real-time retrieval requirements.* Video and audio data must be presented to the user, and hence retrieved and transported, in real-time. In addition, compound objects (objects consisting of more than one media type) usually require synchronizing two or more data types as the object plays out.
- *Large data size.* Typically, a video or audio object is much larger than a text object. For example, a 2-hour movie might require over 1 Gbyte of storage. So, retrieval and transportation mechanisms must not only be fast; they must also have large storage capacity and transfer bandwidth, respectively.

These features strain the capacities of technology designed for unimedia data. Consequently, we need new approaches and techniques in all three areas of computing: storage, processing, and communications.

The relative infancy of multimedia computing exacerbates the difficulties of processing this data. Companies are jumping quickly onto the multimedia bandwagon and producing a plethora of different formats and standards. This only adds to the confusion.

Nevertheless, researchers have taken encouraging steps toward standardization and some measure of order. They now realize that the best way to manage video and image data is to compress them at the source, and decompress them at the destination. This realization has resulted in compression standards. The Joint Photographic Experts Group (JPEG) standard covers image compression, and the Moving Pictures Experts Group (MPEG) standard covers video compression. Asynchronous Transfer Mode (ATM) is becoming the protocol of choice for large-scale distributed multimedia networks. Significant progress has occurred toward the definition of a digital High Definition Television (HDTV) standard. Similarly, the World Wide Web (WWW) has considerably eased the problem of Internet navigation. However, multimedia data processing requires more standards for storage mechanisms, scripting languages, graphical user interfaces, and database representations and manipulations.

Reference

1. P.V. Rangan and H.M. Vin, "Efficient Storage Techniques for Digital Continuous Multimedia," *IEEE Trans. Knowledge and Data Engineering*, Vol. 5, No. 4, Aug. 1993, pp. 564–573.

- *meets quality-of-service requirements.* The QOS requirements of consumers affect the use of server resources. For example, a family viewing a feature film may not mind if an occasional frame is dropped. On the other hand, a medical student viewing a recording of open-heart surgery would demand absolute fidelity of playback. The server should adapt to myriad QOS requirements.
- *is cost effective.* This factor will also govern the blooming or bludgeoning of interactive multimedia services. Installation charges should be no more than a few hundred dollars per customer. Monthly service charges should approximate charges for common antenna television (CATV) services, where a group of subscribers shares a common antenna.
- *exploits user access patterns.* The server must be able to trap and exploit dynamic user behavior. For example, if 10 consumers request the same movie in 3 minutes, it will be prohibitively expensive to start 10 different streams for them. The server must find an alternative. We'll discuss this issue further in the next section.
- *handles real-time and non-real-time traffic.* A high-performance multimedia server's primary function is to serve multiple real-time data streams simultaneously. However, it must also serve non-real-time data satisfactorily. It would encounter such data when downloading new programs from satellites and remote servers, handling billing and accounting, and communicating with intelligent personal agents.[1]
- *provides reliability and availability.* Like any other kind of server, a multimedia server must be reliable. As the volume handled by a server increases, so does the difficulty of guaranteeing reliability. Servers must employ special hardware and software mechanisms to provide fault tolerance for terabytes of data. The server also must have minimal down time, because consumer requests are asynchronous and might arrive at any time.
- *provides fast signal processing.* The server might have to compress video and image data and encode audio data before storage. Consequently, it requires sophisticated scalar and floating-point arithmetic performance and specialized hardware for compression and signal processing.

To meet these requirements, a natural choice is a high-performance computer consisting of multiple processors connected by a high-speed interconnection network. However, parallel computers must overcome

two obstacles. First, their technical complexity and their small user community compared to that of PCs make parallel computers expensive.

Second, I/O constitutes a severe bottleneck in parallel computers. Until recently, most parallel computers concentrated on minimizing the time to handle workloads such as in scientific computing. So, the emphasis was on performing fast arithmetic and handling vector operands. However, multimedia applications require fast data retrieval and real-time guarantees. Therefore, the computer must also be optimized for fast I/O.

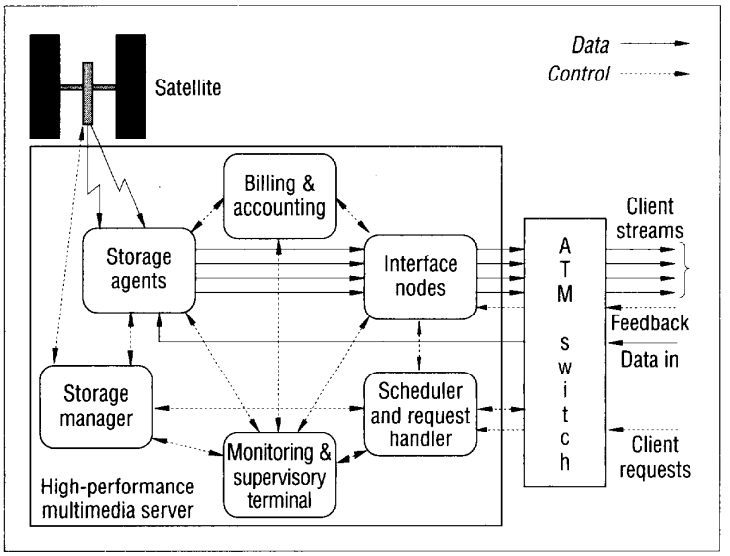

Figure 2. Logical model of a high-performance multimedia server.

Figure 2 shows a logical model of a high-performance multimedia server, based on these requirements. The server connects to a high-speed wide-area network through an ATM switch, and can communicate with orbiting satellites. It consists of six logical modules: *interface nodes*, the *scheduler and request handler*, the *monitoring and supervisory terminal*, the *storage manager*, *storage agents*, and the *billing and accounting module*. Although it would be natural to map each module to one or more nodes of a parallel computer, it is not necessary.

Consumers interact with the server through the interface nodes and the scheduler and request handler. Consumer requests arrive at the scheduler. If the request is for a new stream, the scheduler runs an *admission control policy* that determines if sufficient server resources are free to accept the request and guarantee performance.

The admission control policy executes in the following sequence. The scheduler determines the location of the requested data from the storage manager. Storage agents store the multimedia data. The scheduler asks the identified storage agents if they can accept the request. If they can without violating the real-time requirements of the in-service streams, the admission control policy seeks an interface node to service the stream. If it finds such a node, the agents accept the request, the scheduler and the request handler allocate resources, and playback starts. If the policy cannot find a node, the server cannot accept the request at that time. If the consumer's request is to resume a paused stream, the server takes appropriate actions to resume service.

An interface node services accepted requests. It accepts the schedule from the scheduler, and periodically retrieves data from the storage agents at the required rate for each stream in service. The monitoring and supervisory terminal performs overall coordination and handles errors. The billing and accounting module charges clients, based on the required QOS and actual resource use.

Implementation issues

Researchers are just beginning to understand the physical implications of the requirements of high-performance multimedia servers. Although some research has simulated and modeled parallel machines as multimedia servers,[2] many issues persist, especially regarding implementation.

Storage media

Many storage devices can store persistent multimedia data, including magnetic disks, disk arrays, CD-ROMs, magnetic tape, and optical disks. The most important criteria for selecting a storage medium are high storage capacity, read/write capability, and comparable overheads for both reads and writes. Although the storage capacity of a CD-ROM is larger than that of a magnetic disk, CD-ROMs are inferior on the other counts. Consequently, magnetic disks are becoming popular for high-performance interactive servers. They are easily available at commodity prices, and they have nearly the same access overhead for both reads and writes. Also, disks that store several Gbytes are becoming available. (Of course, CD-ROMs and magnetic tapes could be used for archiving multimedia data.)

However, current magnetic disk arm-scheduling techniques are inadequate. Familiar techniques such as Scan and first-come, first-served (FCFS) evolved before

the advent of multimedia concepts, and do not guarantee real-time performance. The earliest-deadline-first (EDF) approach works for multimedia data, but makes the undesirable assumption that data accesses are preemptible. Therefore, researchers must develop new techniques. A. Reddy and J. Wyllie have proposed a disk arm-scheduling approach for multimedia data, and have characterized the disk-level tradeoffs in a multimedia server.[3,4]

Parallelism of retrieval

Magnetic disks transfer raw data at 20 to 30 Mbps, while MPEG-2-encoded data require about 4 Mbps. So, the maximum number of streams that a single disk can source is limited. More important, the use of a magnetic disk implies seek and rotational latencies from a few milliseconds to tens of milliseconds per access. Increasing the granularity of data retrieved per access amortizes these overheads over a larger quantity of data, so that the overhead cost per byte decreases. However, increasing the retrieval granularity also increases queuing delays at the disk. Using a disk array that reduces data-transfer time by employing the aggregate bandwidth of multiple disks can minimize these delays.

Allocating buffer space to a stream involves a tradeoff between reduced variations in service delay and increased buffer cost.

Closely tied to the problem of minimizing disk overheads is that of placement policy. Contiguous allocation of disk blocks for a media stream amortizes the cost of a seek and rotational delay over the retrieval of a number of media blocks. This amortization minimizes the deleterious effects of disk-arm movement on media data retrieval. However, contiguous allocation fragments disk space if one disk stores the entire stream. Moreover, in this case, the disk's data-transfer rate restricts the maximum retrieval bandwidth. This placement policy makes the disk the bottleneck when the server has to support thousands of simultaneous streams.

To avoid the problems of contiguous allocation, P.V. Rangan and his colleagues have proposed a model based on constrained block allocation.[5,6] This approach is basically noncontiguous disk allocation where the time to retrieve successive stream blocks does not exceed a stream block's playback duration.

S. Ghandeharizadeh and L. Ramos offer a different solution: *striping* media data across several multicomputer nodes.[2] The number of nodes across which the data is striped is the *stripe factor*. The effective retrieval bandwidth is proportional to the stripe factor. A further optimization uses disk arrays at each stripe node. Although a high stripe factor is desirable, two factors limit the stripe factor and the number of disk arrays.

First, in most systems, a peripheral device bus such as a Small Computer System Interconnect (SCSI) or Intelligent Peripheral Interface (IPI) bus connects disks to the rest of the components. To amortize the cost of SCSI controllers, we can use a single bus to connect multiple disks to the system. Depending on the type, a SCSI bus can support bandwidths of 10 to 20 Mbytes per second. This limits the number of disks an array can contain.

Second, with coarse-grain striping (that is, across nodes), the stripe factor directly affects the number of data and control messages being received. An increase in the stripe factor increases the number of messages generated by a data request of a stream from an interface node to storage agents. Consequently, the scheduling, copying, and buffering overheads will increase at the interface node. In summary, striping the data across the nodes of a parallel computer increases the parallelism of retrieval and load balancing at the server. However, determining the stripe factor's optimum value is an open issue.

Granularity of retrieval

Because multimedia data are continuous, a server proceeds in service rounds when retrieving data for multiple users. In other words, the server retrieves stripe fragments for each stream in order and buffers them at the interface node. From there, it sends the data at the required rate to the consumer. For any stream, the data retrieved in one fetch from the storage agents lasts for a time determined by the playback rate. Then, the server must fetch a fresh set of fragments. The granularity of data retrieved from a disk at each access is an important design parameter.

Figure 3 illustrates the importance of data granularity. It shows the average component-wise delays for retrieving data at an interface node with a stripe factor of 4, on an Intel Paragon. The server has six interface nodes and 35 storage-agent nodes. The data is MPEG-1-encoded, and the size of packets to clients is 64 Kbytes. Figure 3a shows delays for 40-Kbyte data messages from the storage agent to the interface node; Figure 3b shows

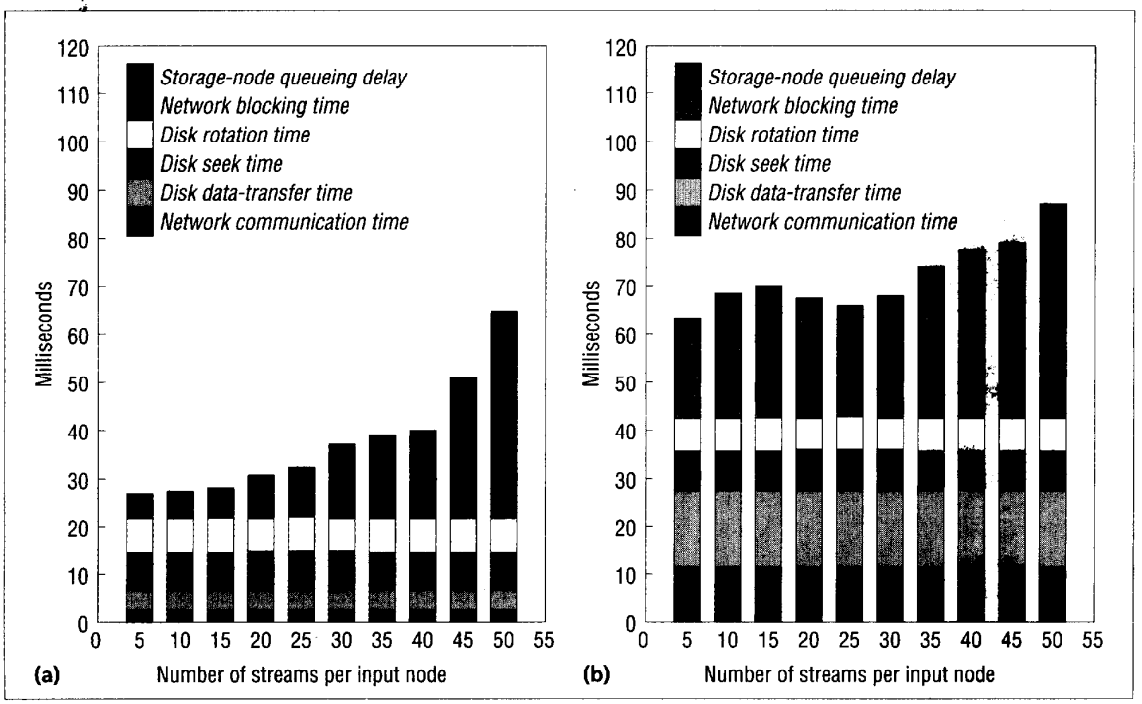

Figure 3. How varying the granularity of fetched data affects average server delays: (a) 40-Kbyte requests, (b) 160-Kbyte requests.

delays for 160-Kbyte messages. The size of data fetched from disk on each access at each storage agent is the same as the size of the data message. Overall delay comprises network communication time (without blocking), disk data-transfer time, disk seek time, disk rotation time, network blocking time, and storage-node queuing delay.

As an interface node serves more streams, the network blocking time and the storage-node queuing delay increase, but to different degrees in each graph. For a fixed stripe factor, as the granularity of data retrieved per access increases, disk seek and rotation time contribute less to the total delay, and data remains longer at the interface node. Also, the frequency of fetches from the storage nodes is lower, so the queuing delay at the storage nodes is less. However, large retrieval granularity also implies larger blocking time in the network, because of the large message size. So, changes in granularity cause tradeoffs in delay components. Moreover, queuing and blocking delay are load-sensitive.

Buffering

Increasing the granularity of data retrieved per disk access can make service time for a data request more predictable. However, in a high-performance server consisting of multiple nodes connected by a network, the granularity of data retrieval and data transfer from storage node to interface node is a design parameter, as Figure 3 shows. We can exploit small message size and large data-retrieval granularity by buffering the data at the storage nodes and interface node.

For example, Figure 4 shows the effect of retrieving 160 Kbytes from the storage node's disk, but sending only 40-Kbyte messages from the storage nodes to the interface nodes. (That is, the storage node maintains a buffer of 160 Kbytes per stream.) The other parameters are the same as for Figure 3.

Clearly, buffering effectively minimizes variations in service. Multimedia data, however, require much storage. We must therefore allocate buffer space in substantial chunks, say 64 Kbytes, as opposed to the few kilobytes that most operating systems use. More important, a high-performance server will require large main memory, approximately 100 Mbytes. So, allocating buffer space to a stream involves a tradeoff between reduced variations in service delay and increased buffer cost.

Scheduling

The real-time nature of multimedia data retrieval profoundly affects resource scheduling at the operating-system level. The server should be able to provide real-time guarantees for individual streams and all the streams combined. For a stream, the server must guarantee individual steps in the access and delivery. These include access from disks, buffering and copying (if buffering occurs), and communication to the network port. Moreover, when a server accepts a user request for

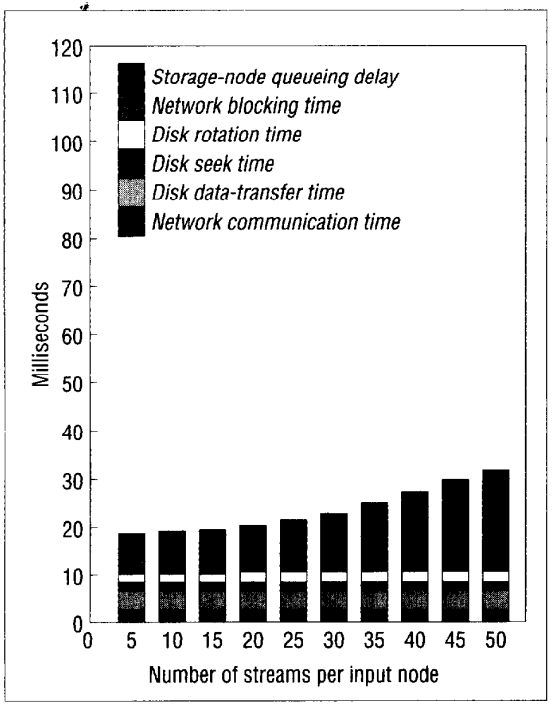

Figure 4. How varying the server node buffer size affects average server delays. The storage node buffer is 160 Kbytes; messages to the interface node are 40 Kbytes.

a new stream, it must continue to provide the promised guarantees to existing streams.

The admission control policy evaluates resource availability and determines whether a new request can be safely accepted.[6,7] It performs the necessary resource reservation to serve a stream, and keeps track of resources committed but temporarily unused because of consumer pause. How to handle consumer pause is an open question. One approach marks the resources committed to a stream as allocated until the consumer resumes. However, this approach likely will lower server utilization and throughput. The alternative releases the allocated resources when a consumer pauses and reacquires them on resumption. This approach increases response time to the consumer, and causes the admission control policy to rerun. The choice of approach depends on the pause's duration, which is not yet well understood. One compromise dedicates some server resources for handling paused streams.

Scheduling problems for multimedia data are particularly severe when the server is a parallel machine with its CPUs interconnected by a low-latency communication mechanism. Large variances typically occur in the time it takes to send a point-to-point message. We need to develop techniques that efficiently handle interrupts. We also need techniques that reduce the difference

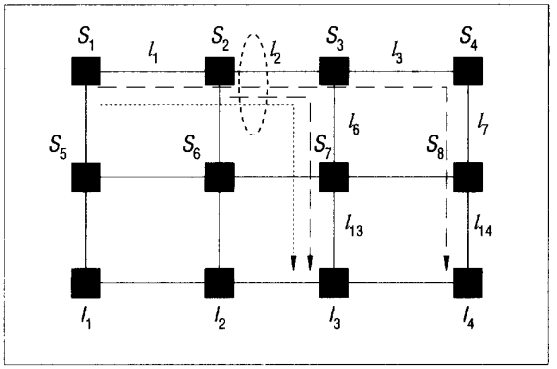

Figure 5. Interconnection network contention caused by deterministic *xy* routing.

between worst-case and average-case latency, which can contribute to variance in fine-grain parallel processors.[8]

Some delays in servicing requests are workload-dependent. One example is queuing delay caused by a single thread of control or a single service resource (such as a disk). Figure 5 shows another example, on a mesh-connected server with four interface nodes and eight storage nodes. Interface node I_3 serves two streams whose data reside on storage nodes S_1 and S_2. Node I_4 serves a stream whose data reside on S_1.

A popular technique to switch data from the input channels to the output channels of the network routers is wormhole routing.[9,10] However, wormhole routing is highly susceptible to deadlock. Various algorithms attempt to provide deadlock-free wormhole routing. The most popular for mesh-connected computers is *deterministic xy routing*. This static technique first sends packets along the mesh's x axis, and then along its y axis. Link contention, however, can cause large and unpredictable network blocking delays with this technique. For example, a link may be heavily used even when alternate paths would cause less contention, and thus smaller delays. In Figure 5, l_2 is such a link. Therefore, we need to develop dynamic (adaptive) deadlock-free routing techniques.

The QOS requirements of consumers can also greatly affect scheduling mechanisms. Depending on whether a real-time client can tolerate no loss or some loss of data, the server can provide hard or soft performance guarantees.[6] Based on the observed server load, soft performance guarantees relax the worst-case assumptions made when admitting new users. This policy lets the server support more users.

Reliability and availability

Unlike scientific computing environments, where a system's reliability and availability are desired but not critical, they are crucial in the commercial environment. If a storage provider's on-demand services are not reliable and available, it will soon be out of business. Therefore, hardware and software solutions must pro-

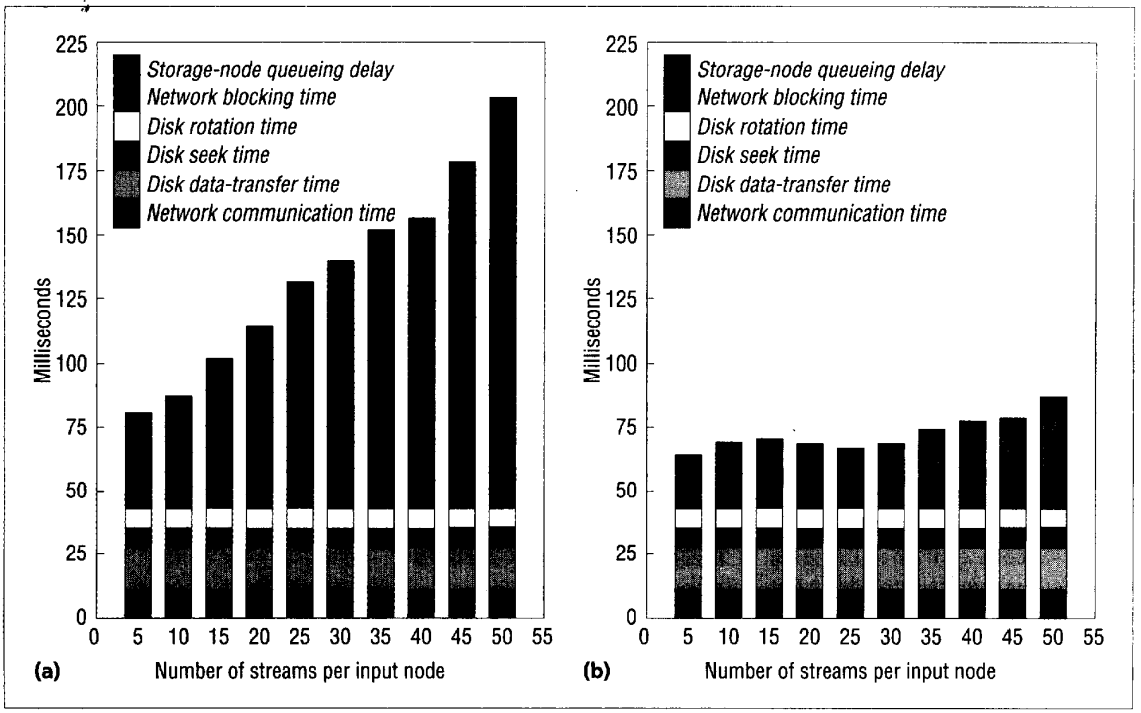

Figure 6. How the ratio of storage nodes to interface node affects average server delays for 41 nodes: (a) 4:1, (b) 6:1.

vide high reliability and availability. They must serve individual customers reliably (in terms of quality and uninterrupted service) and make the system as a whole reliable. They must also provide quick recovery from faults with minimal service degradation.

Reliability ranges from serving one stream reliably (for example, recovering from a disk failure on which the stream data is stored) to serving a large number of streams reliably. In normal operation, scheduling and access techniques affect reliability in terms of QOS (for example, percentage of packets dropped). However, handling failures requires other measures. We can keep redundant copies (for example, data mirroring), copy popular streams on the fly, and reassign clients if the nodes serving them fail.

Also, correctly striping data can improve its availability during failures. For example, we can stripe a primary stream over several disks (instead of placing it on one disk) and stripe its copy over several different disks. Then, only a part of the data will be lost when a disk fails. This scheme can provide more time to recover without degrading the QOS. The probability that a particular stripe fragment is being served at the time of failure will be smaller by a factor equal to the number of nodes on which the stream data is striped. Similarly, rerouting the data when a node fails can improve system availability.

Generally, improvements in reliability and availability also depend greatly on the support that the hardware and system software provide. That is, the server architecture will help determine whether these techniques deliver cost-effective solutions.

Node configuration

Economic factors can limit the number of nodes available to the designer of a multimedia server. A fixed number of nodes forces a designer to make tradeoffs when designating the nodes as storage or interface nodes. Because the interface nodes actually source the client streams, their number should be large, so that the server's total streaming capacity is high. (The number of interface nodes cannot be arbitrary. The server architecture and the number of ports provided by the switch interface between the server and the WAN limit that number). On the other hand, because the storage nodes store the media data, their number should also be large.

Figure 6 depicts the tradeoffs when the ratio of storage nodes to interface nodes varies for 41 nodes. The stripe factor is 4. Packets to clients are 64 Kbytes, and data messages from the storage agent to the interface node are 160 Kbytes. In Figure 6a, the server has 33 storage nodes and 8 interface nodes; in Figure 6b, it has 35 storage nodes and 6 interface nodes. Thus, the ratios of storage nodes to interface nodes are approximately 4 to 1 and 6 to 1. The figure shows the delay components as a function of stream load.

In Figure 6a, the storage-node queuing delay is the

largest component, while in Figure 6b, the network blocking time is largest. The configuration in Figure 6a has fewer nodes to store data, and more data streams to serve. So, the storage nodes become the throughput bottleneck.

Our results show that a low storage-to-input (S-to-I) ratio produces higher average total retrieval time. The storage-node queuing delay is much higher for a low S-to-I ratio. Given a fixed number of nodes and a certain S-to-I ratio, a designer can increase the ratio to increase storage space. The server will be able to source fewer streams. However, the designer can afford to choose disks with lower performance to guarantee the same QOS to consumers at a lower net server cost.

Caching

An important operational aspect that has received little attention is the use of caching techniques to exploit knowledge of user request patterns. Because traditional cache memories are too small to hold multimedia data, these techniques use processor main memory for buffering.

Gang scheduling accumulates requests over a time interval, and avoids multiple fetches for requests for the same object during that interval.

The chief premise behind these techniques is that user requests for stored data are not uniformly distributed. For example, a neighborhood VOD server's workload will be higher in the evening and at night than during the afternoon. More important, some movies will be more popular than others. For example, on a given night, customers will more likely request a newly released movie than one released five years ago.

Given a nonuniform request rate for different movies, we can develop various optimizations that increase server throughput by caching the popular movies. At the storage level, we can replicate the data and store the copies on different node subsets. When one copy of a popular movie is inadequate to service the anticipated requests, the multiple copies can serve multiple streams. This helps achieve load balancing. However, although replication improves load balancing and lets the server handle more streams, each copy requires extra storage (of the order of Gbytes).

Also, we could reconfigure the server dynamically so that an interface node is also a storage node.[11] When consumers frequently access a media object, we would migrate that object from the nodes on which it is stored to local disks at an interface node. We would then dedicate the interface node to serving requests for that object. This increases the server's throughput, but also increases server software complexity.

Gang scheduling is another approach to increasing server throughput. It accumulates requests over a time interval (a *gang window*), and avoids multiple fetches for requests for the same object during that interval. For instance, assume that, during a gang window of 5 minutes, a server receives 10 requests for a certain object. It starts retrieving data for only one stream at the end of the gang window, and it sources 10 client streams from the same data. Gang scheduling incurs extra overhead from accumulating requests during the gang window and searching through those requests to identify repeated requests. This method delays servicing some requests to minimize the server load. So, a tradeoff exists between increased response time for consumers and reduced server workload. Consequently, the gang window's size is crucial.

This approach can cause problems. If a consumer interrupts the stream, say for pausing or fast forward, that consumer will fall out of phase with the stream being retrieved. In this case, the server should be able to establish dynamically a fresh server-interface stream for the consumer.

A different approach hardwires part or all of a popular media object's data in RAM to hide the latency of disk accesses. Yet another approach delays the freeing of media data buffer space in RAM as much as possible. The server fulfills later requests for the "stale" data directly from RAM (instead of disk). Both approaches assume that considerable buffer space is available and media data size is reasonable.

Although VOD has been one of the most visible MOD applications, much of the hype surrounding it has abated because of failures and delays in field trials. The coming years will decide whether geographically distributed interactive multimedia services such as VOD are technologically and economically feasible.

Also, much work needs to be done in real-time scheduling, parallel I/O, reliability, scalability, dynamic scheduling, and caching techniques for multimedia data stored on high-performance computers. The best architecture

What architecture?

What architecture will work best for an MOD server? Companies such as Oracle/nCube, Intel, and IBM promote distributed-memory multicomputers; others, such as Silicon Graphics, Digital, and Hewlett-Packard, promote shared-memory multiprocessor technology.

Also, parallel machines now compete directly with volume-produced PCs and workstations. Companies such as Oracle and Silicon Graphics advocate powerful parallel computers as multimedia servers. Companies such as Microsoft and Compaq claim to achieve equivalent functionality at lower cost with servers that interconnect the bulk-produced chips used in PCs.[1] (For example, Microsoft's Tiger file system uses a high-speed communication fabric to interconnect Pentium-based nodes.)

At present, there is no clear winner among distributed-memory multicomputers, shared-memory multiprocessors, or networks of PCs/workstations. The acceptance of any one of these architectures will depend largely on the cost-effectiveness and QOS that it can provide.

Reference

1. "Microsoft Touts PC Video Servers Better Than Supercomputers," *HPCwire* (electronic magazine), article no. 4097, May 1994; send e-mail to help@hpcwire.ans.net.

must also be determined (see the "What architecture?" sidebar).

Can high-performance computers be cost-effective multimedia servers and give timely service? The answer will determine whether large-scale on-demand services mature from a "plug-and-pray" to a plug-and-play technology.

Acknowledgments

We thank A.N.L. Narasimha Reddy of IBM for many helpful suggestions, and Dave Riss and Denise Eklund of Intel SSD for technical inputs and support. The Intel Corp., the NSF Young Investigator Award CCR-9357840, and the New York State Center for Advanced Technology in Computer Applications and Software Engineering (CASE) at Syracuse University supported this work. We thank the Caltech CCSF for access to the Intel Paragon.

References

1. S. Ramanathan and P.V. Rangan, "Architectures for Personalized Multimedia," *IEEE Multimedia*, Vol. 1, No. 1, Spring 1994, pp. 37–46.
2. S. Ghandeharizadeh and L. Ramos, "Continuous Retrieval of Multimedia Data Using Parallelism," *IEEE Trans. Knowledge and Data Engineering*, Vol. 5, No. 4, Aug. 1993, pp. 658–669.
3. A. Reddy and J. Wyllie, "Disk-Scheduling in a Multimedia I/O System," *Proc. First ACM Int'l Conf. Multimedia*, ACM Press, New York, 1993, pg. 225.
4. A.L.N. Reddy and J.C. Wyllie, "I/O Issues in a Multimedia System," *Computer*, Vol. 27, No. 3, Mar. 1994, pp. 69–74.
5. P.V. Rangan and H.M. Vin, "Efficient Storage Techniques for Digital Continuous Multimedia," *IEEE Trans. Knowledge and Data Engineering*, Vol. 5, No. 4, Aug. 1993, pp. 564–573.
6. P.V. Rangan, H. Vin, and S. Ramanathan, "Designing an On-Demand Multimedia Service," *IEEE Comm.*, Vol. 30, No. 7, July 1992, pp. 56–64.
7. D. Anderson, Y. Osawa, and R. Govindan, "A File System for Continuous Media," *ACM Trans. Computer Systems*, Vol. 10, No. 4, Nov. 1992, pp. 311–337.
8. R. Mraz, "Reducing the Variance of Point-to-Point Transfers for Parallel Real-Time Programs," *IEEE Parallel & Distributed Technology*, Vol. 2, No. 4, Winter 1994, pp. 20–31.
9. L.M. Ni and P.K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *Computer*, Vol. 26, No. 2, Feb. 1993, pp. 62–76.
10. P. McKinley, Y. Tsai, and D. Robinson, "A Survey of Collective Communication in Wormhole-Routed Massively Parallel Computers," Tech. Report MSU-CPS-94-35, Dept. of Computer Science, Michigan State Univ., East Lansing, Mich., 1994.
11. D. Jadav et al., "Design and Evaluation of Data-Access Strategies in a High-Performance Multimedia-on-Demand Server," *Proc. Second IEEE Int'l Conf. Multimedia Computing and Systems*, 1995, pp. 286–291.

Divyesh Jadav is a PhD candidate in computer engineering in the Department of Electrical and Computer Engineering at Syracuse University. He is also a research assistant at the New York State Center for Advanced Technology in Computer Applications and Software Engineering (the CASE Center) at Syracuse University. His research interests include parallel and distributed databases, and multimedia servers and networks. He received his MS in computer engineering from Syracuse University in 1994, and his BE in computer engineering from the Victoria Jubilee Technical Institute, Bombay University, India, in 1992. He is a member of Phi Beta Delta, the ACM, the IEEE, and the IEEE Computer Society. He can be contacted at the Dept. of Electrical and Computer Engineering, 121 Link Hall, Syracuse Univ., Syracuse, NY 13244; divyesh@cat.syr.edu.

Alok Choudhary is an associate professor at the Department of Electrical and Computer Engineering at Syracuse University. His main research interests are in parallel and distributed processing; software development environments for parallel computers, including compilers and runtime support; parallel computer architectures; and parallel I/O for scientific and multimedia applications. He is a subject-area editor of the *Journal of Parallel and Distributed Computing*, and has been a guest editor for *JPDC* and *Computer*. He has also coauthored *Parallel Architectures and Parallel Algorithms for Integrated Vision Systems* (Kluwer Academic, 1990). He was a program co-chair for the 1993 International Conference on Parallel Processing.

He received his PhD in electrical and computer engineering from the University of Illinois, Urbana-Champaign, in 1989. He received his MS in electrical and computer engineering from the University of Massachusetts, Amherst, in 1986, and his BE (Hons.) in electrical and electronics engineering from the Birla Institute of Technology and Science, Pilani, India, in 1982. He received an IBM Faculty Development Award in 1994, the NSF Young Investigator Award in 1993, and an IEEE Engineering Foundation Award in 1990. He is a member of the IEEE Computer Society and the ACM. He can be contacted at the Dept. of Electrical and Computer Engineering, 121 Link Hall, Syracuse Univ., Syracuse, NY 13244; choudhar@cat.syr.edu.