Multimedia Storage Servers: A Tutorial

D. James Gemmell
Simon Fraser University

Harrick M. Vin
University of Texas at Austin

Dilip D. Kandlur
International Business Machines

P. Venkat Rangan
University of California at San Diego

Lawrence A. Rowe
University of California at Berkeley

Real-time processing of multimedia data is required of those who offer audio and video on-demand. This tutorial highlights the unique issues and data storage characteristics that concern designers.

Recent advances in computing and communication make on-line access to multimedia information—like books, periodicals, images, video clips, and scientific data—both possible and cost-effective. The architecture for these services consists of multimedia storage servers connected to client sites via high-speed networks (see Figure 1). Clients can retrieve multimedia objects from the server for real-time playback. Furthermore, access is interactive because clients can stop, pause, and resume playback and, in some cases, perform fast-forward and rewind operations.

Some media (such as audio and video) are classified as *continuous* because they consist of a sequence of media *quanta* (such as audio samples or video frames), which convey meaning only when presented in time. The design of services to support continuous media (CM) differs significantly from that of services to support only traditional textual and numeric data because of two fundamental CM characteristics:

- Real-time storage and retrieval: CM recording devices (such as video cameras) generate a continuous stream of media quanta that must be stored in real time. CM playback is essentially recording in reverse: The media quanta must be presented using the same timing sequence with which they were captured. Any deviation from this timing sequence can lead to artifacts such as jerkiness in video motion, pops in audio, or possibly complete unintelligibility. Furthermore, media components can be combined in a fashion requiring synchronization. For example, a slide presentation must synchronize audio (music and commentary) with images.
- *High data transfer rate and large storage space:* Digital video and audio playback demands a high data transfer rate (see Table 1), so storage space is rapidly filled. Thus, a multimedia service must efficiently store, retrieve, and manipulate data in large quantities at high speeds.

Originally published in *Computer,* May 1995 , pp. 40–49.

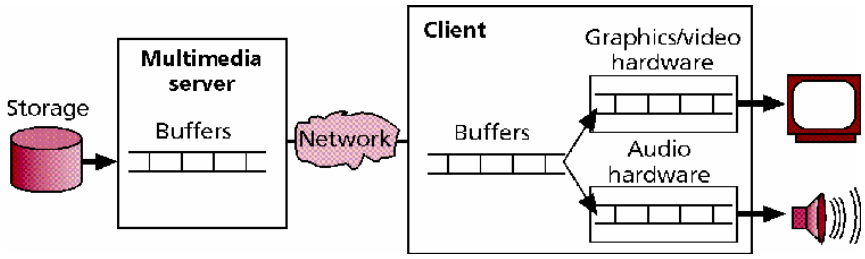

Figure 1. Data flow for a multimedia network server.

Table 1. Storage space requirements for uncompressed digital multimedia data.

Media Type	Specifications	Data Rate (per second)
Voice-quality audio	1 channel, 8‐bit samples at 8 kHz	64 Kbits
MPEG-encoded audio	Equivalent to CD quality	384 Kbits
CD-quality audio	2 channels, 16-bit samples at 44.1 kHz	1.4 Mbits
MPEG-2-encoded video	640 x 480 pixels/frame, 24 bits/pixel	0.42 Mbytes
NTSC-quality video	640 x 480 pixels/frame, 24 bits/pixel	27 Mbytes
HDTV-quality video	1,280 x 720 pixels/frame, 24 bits/pixel	81 Mbytes

Consequently, the critical components in the design of multimedia services are

1. *multimedia storage servers* that support continuous media (CM) storage and retrieval, and
2. *network subsystems* that synchronously deliver media information, on time, to the client sites.

Our focus is to survey the design issues of digital multimedia storage servers. We describe the design issues of multimedia, and we assume a network subsystem (or transmission channel) that delivers CM information according to its real-time specifications. The network can be simply telephone lines of sufficient bandwidth, which clients can call for server access.

Continuous media recording and retrieval

Video digitization yields a sequence of continuously recorded video frames; audio digitization yields a sequence of continuously recorded audio samples. Because media quanta, as we mentioned earlier, convey meaning only when presented continuously in time, a multimedia server must ensure that the recording and playback of each media stream proceeds at its real-time data rate. During recording, for example, a server must continuously store the data produced by an input device (such as a microphone or camera) to prevent buffer overruns at the device. During playback, on the other hand, the server must retrieve data from the disk at a rate that prevents an output device (such as a speaker or video display) from starving. Although semantically different, both operations are mathematically equivalent with respect to their real-time performance requirements.[1] For simplicity, we discuss techniques only for retrieving media information for real-time playback, although real-time recording can be similarly analyzed.

Single-stream playback

A media stream's continuous playback is a sequence of periodic tasks with deadlines. Tasks correspond to retrievals of media blocks from disk, and deadlines correspond to the scheduled playback times. Although it's conceivable that multimedia systems could fetch media quanta from disk just in time to be played, in practice the retrieval is likely to be bursty. Media blocks will need to be buffered when retrieval gets ahead of playback.

The server's challenge, consequently, is to supply the stream buffers with enough data to ensure that the playback processes do not starve[2] (see Figure 2). Continuous playback can be assured simply by buffering the entire stream before initiating the playback; how-

ever, this requires a large buffer and will cause a lengthy latency for initiating playback of large files. Efficiently servicing a single stream is thus a threefold problem: preventing starvation while minimizing the buffer space requirement and the initiation latency. These two minimization problems are, in fact, the same—minimizing one will minimize the other.[2] Furthermore, because disk data transfer rates are significantly higher than a single stream's real-time data rate, even a small buffer will let conventional file and operating systems support continuous storage and retrieval of a few media streams. (Consider that the maximum throughput of modern disks is around 3 to 4 Mbytes per second while that of an MPEG-2-encoded video stream is 0.42 Mbytes and uncompressed CD-quality stereo audio is about 0.2 Mbytes.)

Multistream retrieval

In practice, a multimedia server must process retrieval requests for several streams simultaneously. Even when multiple streams access the same file (such as a popular movie), different streams might access different parts of the file at the same time.

A simple way to guarantee meeting the real-time requirements of all streams is to dedicate a disk head to each stream and treat each disk head as a single-stream system. This however limits the total number of streams to the number of disk heads. Because disk data rates significantly exceed those of single streams, the number of streams that can be serviced simultaneously can generally be increased by multiplexing a disk head among several streams. In doing so, the server must meet the continuous playback requirements of all streams through carefully scheduling disk requests so that no individual stream starves. Furthermore, the server must ensure that it can in fact schedule disks by limiting the number of streams serviced at any given time.

Disk scheduling

Servers traditionally employ disk-scheduling algorithms—such as first come, first served; shortest seek time first; and Scan—to reduce seek time and rotational latency, to achieve high throughput, or to provide fair access to each stream. Real-time constraints, however, reduce the direct application of traditional disk-scheduling algorithms to multimedia servers.

The best-known algorithm for real-time scheduling of tasks with deadlines is the *earliest deadline first* algorithm. This algorithm schedules the media block with the earliest deadline for retrieval. Scheduling of the disk head based solely on EDF policy, however, is likely to yield excessive seek time and rotational latency, and poor server-resource utilization can be expected.

One variant of this basic algorithm combines Scan with EDF and is called the Scan-EDF scheduling algorithm.[3] The Scan algorithm scans the disk head back and forth across the disk's surface and retrieves a requested block as the head passes over the surface. By limiting the amount of backtracking that the disk head does, Scan can significantly reduce seek latencies. Scan-EDF services the requests with earliest deadlines first, just like EDF; however, when several requests have the same deadline, their respective blocks are accessed with the Scan algorithm. Clearly, the Scan-EDF technique's effectiveness depends on the number of requests having the same deadline. When deadlines for media block retrieval are batched (for example, by initiating media strand playbacks only at certain intervals), Scan-EDF is reduced to Scan only.

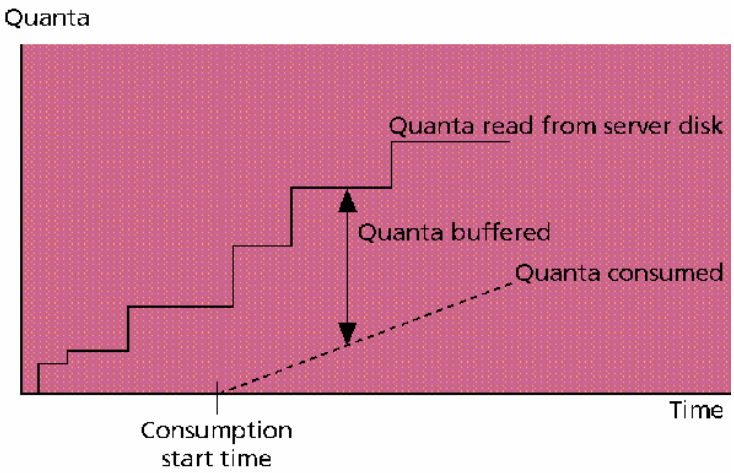

Figure 2. Ensuring continuous retrieval of a media stream from disk.

Originally published in *Computer*, May 1995 , pp. 40–49.

Scan-EDF is a unique disk-scheduling algorithm for CM because it does not intrinsically batch requests. All other algorithms typically process requests in *rounds*. During each round, the multimedia server retrieves a media block sequence of arbitrary length (even zero) for each stream. Processing requests in rounds is more than a convenience; it also exploits the periodic nature of CM playback.

Each round still requires a disk-scheduling algorithm, the simplest of which is the *round-robin* algorithm. This services streams in a fixed order that does not vary from one round to the next. Round-robin scheduling's major drawback is that it, like EDF, does not exploit the relative positions of the media blocks being retrieved during a round. For this reason, data-placement algorithms that inherently reduce latencies are sometimes used in conjunction with round-robin scheduling.

Round length and latency trade-offs

Applying the Scan algorithm to reduce round latencies is simple. For CM servers, minor alterations to Scan can minimize both the seek latencies and the round length.[4] In addition to round-length minimization, latencies between successive stream retrievals are also an issue for the CM server. In the round-robin algorithm, the order in which streams are serviced is fixed across rounds. Therefore, the maximum latency between retrieval times of streams' successive requests is bounded by a round's duration. With Scan, the relative order for servicing streams depends solely on the placement of blocks being retrieved; thus a stream can receive service at the beginning of one round and at the end of another round.

The latency between successive stream retrievals has several implications for playback initiation delay and buffer requirements. For round-robin, playback can be initiated immediately after all blocks from the stream's first request have been retrieved. With Scan, however, playback must wait until the end of the round. To prevent output device starvation, round-robin needs enough buffer space to satisfy data consumption for one round, while Scan needs enough to satisfy consumption for nearly two rounds. However, because Scan's rounds are shorter, there is a trade-off between round length and latency between successive stream retrievals.

To exploit this trade-off, a disk-scheduling algorithm known as the *grouped sweeping scheme* (GSS) partitions each round into groups. Each stream is assigned to a group, and the groups are serviced in a fixed order in each round. The Scan disk-scheduling algorithm is used in each group. If all streams are assigned to the same group, GSS reduces to Scan. On the other hand, if each stream is assigned to its own unique group, GSS degenerates to round-robin. By optimally deriving the number of groups, the server can balance the reduction of round length against the latency of successive stream retrievals (see Figure 3).

Reading and buffering requirements

As mentioned, nearly all multistream CM retrieval approaches involve processing stream requests in rounds. Another almost universal practice found in the literature is to ensure that production matches consumption in each round. During a round, the amount of data retrieved for a stream is at least equal to the amount consumed by the stream's playback. This means that, on a round-by-round basis, data production never lags consumption, and there is never a net decrease in the amount of buffered data. Algorithms having this property are referred to as *workahead-augmenting*[1] or *buffer-conserving*.[4]

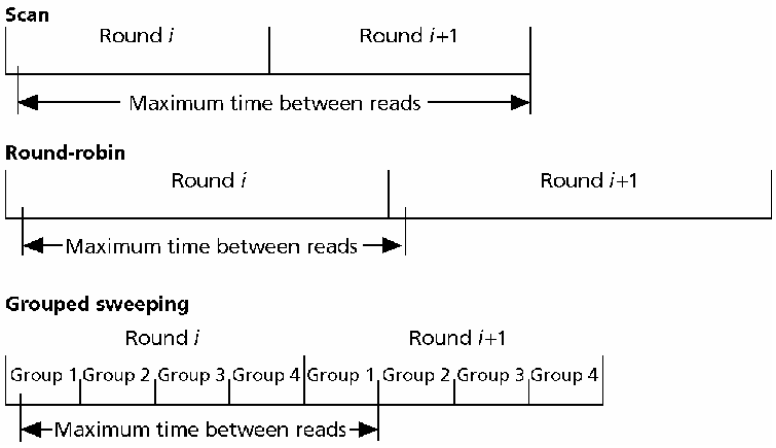

Figure 3. Trade-off between round length and time between service for Scan, round-robin, and grouped sweeping scheme.

Originally published in *Computer*, May 1995 , pp. 40–49.

An algorithm could conceivably be developed that proceeds in rounds but is not buffer-conserving. Such an algorithm would allow production to fall behind consumption in one round and compensate for it in a later round, although this would be more complex. Furthermore, while buffer conservation is not a necessary condition for preventing starvation, it can be used as a sufficient condition. For instance, before initiating playback, if enough data is prefetched to meet the consumption requirements of the longest possible round and if each round thereafter is buffer-conserving, it is clear that starvation is impossible.

To ensure continuous playback of media streams, a sufficient number of blocks must be retrieved for each client during a round to prevent the output device's starvation for the round's entire duration. To determine this number, the server must know the maximum duration of a round. As round length depends on the number of blocks retrieved for each stream, some care should be taken that unnecessary reads are not performed. In particular, a simple scheme that retrieves the same number of media blocks for each stream will be inefficient because the stream with the maximum consumption rate will dictate the number of blocks to read. This will cause streams with smaller consumption rates to read more than they need. To minimize round length, the number of blocks retrieved for each stream during each round should be proportional to the stream's consumption rate.[1–2, 5,–6]

Managing buffers for maximum consumption

Naturally, a server must manage its buffers to leave sufficient free space for the next reads to be performed. On a per-stream basis, the most suitable buffer model is a first-in, first-out queue. Using a FIFO, contiguous files, and round-robin scheduling, the buffer size can approximate the size of the maximum required read. In this case, each stream's FIFO can simply be "topped up" in each round (that is, enough data is read to fill the FIFO). In contrast, a Scan strategy would require at least a double-buffered scheme, where each buffer is the size of a maximum read. This is because Scan can schedule the reads for a stream so that the stream is serviced last in one round and first in the next (back to back). If a buffer is not completely empty when it is time for reading, the topping-up strategy can still be used.

With a topping-up strategy, the amount read for a stream in each reading period will vary with the amount of free buffer space. When files are not stored contiguously but are split into blocks, variable read amounts might mean that the data to be retrieved is split across two blocks, causing an extra intrafile seek. One solution to this problem uses three block-sized buffers. With three buffers, the only time a whole block cannot be read is when at least two buffers are full; buffering is otherwise sufficient so that reading is not necessary until the next round. Other solutions with fewer than three buffers are also possible.[4]

Admission control

Given streams' real-time performance requirements, a multimedia server must employ admission control algorithms to determine whether a new stream can be serviced without affecting streams already being serviced. So far we have assumed that stream performance requirements include meeting all real-time deadlines; however, some applications can tolerate missed deadlines. For example, a few lost video frames or a pop in the audio can occasionally be tolerated, especially if such tolerance is rewarded with a reduced cost of service. To guarantee that a server meets all real-time deadlines, worst-case assumptions must be made by the server regarding seek and rotational latencies, although the seek time and rotational latency incurred are generally shorter than those in a worst case. A multimedia server might therefore accommodate additional streams through an admission control algorithm that exploits the statistical variation in media block-access times from disk (or statistical variations in compression ratios, where applicable).

In admitting streams, CM servers can offer three broad quality-of-service categories:

- *Deterministic*. All deadlines are guaranteed to be met. For this level of service, the admission control algorithm considers worst-case scenarios in admitting new streams.
- *Statistical*. Deadlines are guaranteed to be met with a certain probability. For example, a client subscribes to a service that guarantees meeting 90 percent of deadlines over an interval. To provide such guarantees, admission control algorithms must consider the system's statistical behavior while admitting new streams.
- *Background*. No guarantees are given for meeting deadlines. The server schedules such accesses only when there is time left over after servicing all guaranteed and statistical streams.

To implement deterministic service, resources are reserved in worst-case fashion for each stream. Before admitting another stream and lengthening the round's duration, the server must ensure that buffering for existing streams is adequate to prevent starvation.[1, 6] Some schemes dynamically change the stream buffer spaces based on the current round's length. Alternately, all stream buffer spaces can be allocated assuming a

Originally published in *Computer*, May 1995 , pp. 40–49.

maximum round length and, for admission, the new round length need only be compared to the maximum.[4]

Statistical service implementation resembles that of deterministic service, but instead of the server's computing the change to round length based on worst-case values, the computation is based on statistical values. For instance, the computation can use an average rotational-delay value that would be expected to occur with a certain probability based on a random distribution of rotational delays.

In servicing streams during a round, deterministic streams must be guaranteed service before any statistical streams, and all statistical streams must similarly be serviced before any background streams. Missed deadlines should be distributed fairly so that the same streams are not dropped each time.

Dealing with service guarantees and deadlines

When variable-rate compression is used, a media block will decompress into a variable amount of media quanta. Therefore, the number of blocks that must be retrieved will vary according to the compression ratio achieved for each block. In a like manner when dealing with variable disk latencies, deterministic service for such data could use worst-case compression figures, and statistical service could use probabilistic figures. With compressed data, a further option is to record the compression ratios achieved. Deterministic service could then be based on actual rather than worst-case figures, and statistical service could be based on the files' actual statistics rather than on the statistics for the compression algorithm in general.

For background and statistical traffic, different strategies are available to resolve missed deadlines. For example, although it might be desirable not to skip any data blocks to ensure that the information received is intelligible, this technique would lengthen the playback duration of media streams. On the other hand, if the playback of multiple media streams is being temporally coordinated, dropping media blocks might be preferable.

Techniques that dynamically vary media resolution levels to accommodate an overloaded server significantly depart from these simplistic schemes. For example, audio quality can be degraded simply by transmitting only the higher order bits. Similarly, some compression schemes can be made scalable—that is, data is encoded so that subsets of the media stream can be extracted and decoded to achieve lower resolution output. To deal with missed deadlines, techniques for varying resolution are generally similar to those used for implementing fast forward.

Managing digital multimedia storage

A multimedia server must divide video and audio files into blocks while storing them on disk. Each data block can occupy several physical disk blocks. Techniques for managing disk storage include optimally placing data blocks on disk, using multiple disks, adding tertiary storage to gain additional capacity, and building storage hierarchies.

Placing data blocks for optimal service

A file's blocks can be stored contiguously or scattered about the storage device. Contiguous files are simple to implement but subject to fragmentation. They also can necessitate enormous copying overheads during insertions and deletions to maintain contiguity. In contrast, scattered placements avoid fragmentation and copying overheads. Contiguous layouts are useful in read-only systems, such as video-on-demand, but not for read-write servers.

Contiguous placement. For continuous media, the choice between contiguous and scattered files relates primarily to intrafile seeks. When reading from a contiguous file, only one seek is required to position the disk head at the start of the data. However, when reading several blocks in a scattered file, a seek could be incurred for each block read. Furthermore, even when reading a small amount of data, it is possible that half of the data might be stored in one block and the other half in the next block, thereby incurring intrafile seeks.

Intrafile seeks can be avoided in scattered layouts if the amount read for a stream always evenly divides a block. One approach to achieve this result is to select a sufficiently large block size and read one block in each round. This technique has several advantages, especially for large video servers. It improves disk throughput substantially, thereby increasing the number of streams that can be served by the disk. Furthermore, since a file system has to maintain indexes for each media block, choosing a large block size also reduces the overhead for maintaining indexes.

Constrained placement. If more than one block is required to prevent starvation prior to the next read, intrafile seeks are necessary. Instead of avoiding intrafile seeks, another approach is to reduce them to a reasonable bound. This is referred to as the *constrained placement* approach.[1, 7] Constrained placement techniques ensure that the separation between successive file blocks is bounded. The bound on separation is generally not enforced for each pair of successive blocks but only on average over a finite sequence of blocks (see Figure 4).

Originally published in *Computer,* May 1995 , pp. 40–49.

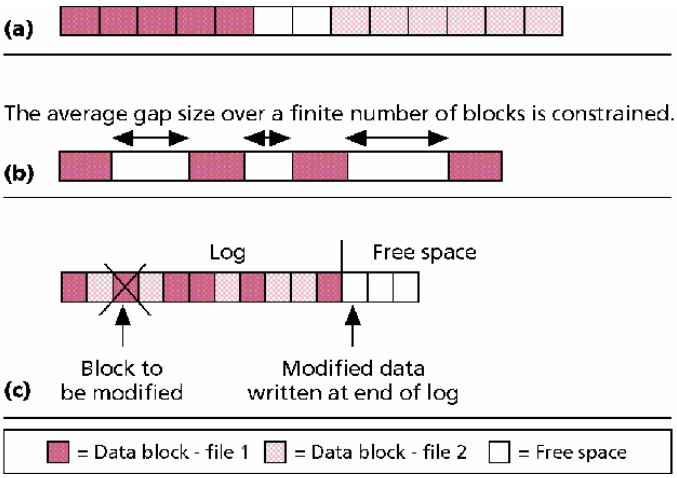

Figure 4. Data placement schemes for multimedia storage: (a) contiguous placement, (b) constrained placement, and (c) log-structured placement.

Constrained placement is particularly attractive when the block size must be small (for example, when using a conventional file system with block sizes tailored for text). Implementing constrained placement can require elaborate algorithms to assure that separation between blocks conforms to the required constraints. Furthermore, for constrained latency to yield its full benefits, the scheduling algorithm must immediately retrieve all blocks for a given stream before switching to any other stream. An algorithm like Scan, which orders blocks regardless of the stream they belong to, highly reduces the impact of constrained placement.[4]

Log-structure placement. One way to reduce disk seeks is to adapt "log-structured" file systems.[5] When modifying blocks of data, log-structured systems do not store modified blocks in their original positions. Instead, all writes for all streams are performed sequentially in a large contiguous free space (see Figure 4). Therefore, instead of requiring a seek (and possibly intrafile seeks) for each stream writing, only one seek is required prior to a batch of writes. This leads to a dramatic performance improvement during recording.

A log-structured approach, however, does not guarantee any improvement in playback performance and is more complex to implement because modified blocks may change position. Consequently, log-structured file systems are best suited for multimedia servers that support extensive editing and are inappropriate for systems that are primarily read-only (for example, video-on-demand servers, which could likely implement writes in non-real time).

Special placement considerations apply when the media is encoded with variable bit-rate compression. Conventional fixed-sized clusters correspond to varying amounts of time, depending on the compression achieved. Alternately, the system can store data in clusters that correspond to a fixed amount of time, with a variable cluster size. Furthermore, compressed media quanta might not correspond to an even number of disk sectors, which raises questions about "packing" data.[2] With scalable compression, data must be carefully placed and managed to ensure efficient extraction of low-resolution subsets.

Data striping and data interleaving

If an entire multimedia file is stored on one disk, the number of concurrent accesses to that file are limited by disk throughput. One approach to overcome this limitation is to maintain multiple copies of the file on different disks, but this is expensive because it requires additional storage space. A more effective approach is to scatter the multimedia file across multiple disks. This scattering can be achieved by using two techniques: data striping and data interleaving.

RAID (redundant array of inexpensive disks) technology has popularized the use of parallel access to an array of disks. Under the RAID scheme, data is "striped" across each disk (see Figure 5). Physical sector 1 of each disk is accessed in parallel as a large logical sector 1. Physical sector 2 of each disk is accessed as logical sector 2, and so on. In this configuration, the disks in the set are *spindle synchronized* and operate in lock-step parallel mode. Because accesses are performed in parallel, logical and physical blocks have identical access times. Therefore, the transfer rate is effectively increased by the number of drives involved.

Originally published in *Computer*, May 1995 , pp. 40–49.

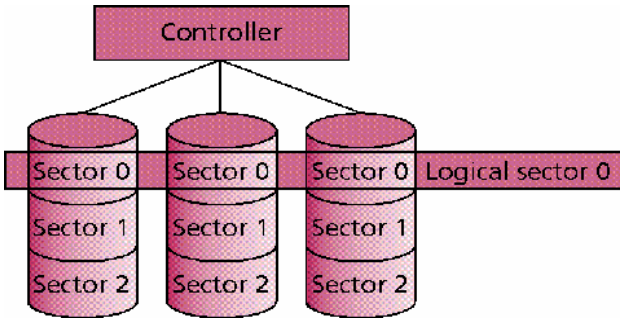

Figure 5. Striped data accessed in parallel.

With their increased transfer rates, disk arrays are a good solution to the problem of the CM's high bandwidth requirements. However, striping cannot improve the seek time and rotational latency incurred during retrieval. The throughput of each disk in the array is still determined by the ratio of the useful read time to the total (read plus seek) time. As with the single disk configuration, disk throughput can be improved by increasing the size of the physical block. However, this would also increase the logical block size and consequently lengthen start-up delays and enlarge buffer space requirements per stream.

In *data interleaving*, the blocks of the media file are interleaved across the disk array for storage, with successive file blocks stored on different disks. A simple interleave pattern stores the blocks cyclically across an array of *N* disks. The disks in the array are not spindle synchronized, and they operate independently.

Two data retrieval methods can be used with this organization. The first one follows the data striping model, whereby one block is retrieved from each disk in the array for each stream in every round. This method ensures a balanced load for the disks but requires more buffer space per stream. In the second method, data is extracted from one of the disks for a given stream in each round (see Table 2). Data retrieval for the stream thus cycles through all disks in *N* successive rounds. In each round, the retrieval load must be balanced across the disks to maximize throughput of *N* disks. Because each stream cycles through the array, this load can be balanced by staggering the streams. With staggering, all streams still have the same round length, but each stream considers the round to begin at a different time, so that their requests are staggered rather than simultaneous.

A combination of data striping and data interleaving can scatter the media file across a large number of disks attached to a networked cluster of servers. This technique lets a scalable video server be constructed that can serve many streams from a single copy of the media file. Moreover, redundancy techniques can be applied to the media file to increase availability and throughput.

Tertiary storage and hierarchies

Although the performance of fixed magnetic disks makes them desirable for CM applications, their high cost per gigabyte makes them impractical as the sole storage medium for a large-scale server (such as a video-on-demand server with hundreds of feature-length titles, each being several Gbytes in size even with MPEG-2 compression). For large-scale servers, economics will dictate the use of large tertiary storage devices such as tape and optical jukeboxes.

Table 2. Reading interleaved data (method 2).

Round	Disk 1	Disk 2	Disk 3
1	File A, block 1	File B, block 1	File C, block 1
2	File C, block 2	File A, block 2	File B, block 2
3	File B, block 3	File C, block 3	File A, block 3
4	File A, block 4	File B, block 4	File C, block 4

Tertiary storage devices are highly cost-effective and offer enormous storage capacities by means of robotic arms that serve removable tapes or disks to a few reading devices (see Table 3). However, their slow random access—due to long seeking and loading times—and relatively low data transfer rates make them inappropriate for CM playback. Consequently, large-scale servers will need to combine the cost-effectiveness of tertiary storage with the high performance of fixed magnetic disks. The storage subsystem will need to be organized as a hierarchy and the magnetic disks used as a cache for the tertiary storage devices.

Several approaches are possible for managing such a storage hierarchy. One approach is to use the magnetic disks only as storage for the beginning segments of the multimedia files. These segments can reduce the startup latency and ensure smooth transitions in the playback.[8] When media files are to be played back, another alternative is to move the entire file from tertiary storage to the disks. A drawback with this approach is that the startup delays associated with loading the entire file will be very high for large files like videos. Fortunately, for applications like video-on-demand, relatively few titles will generally be popular at any given time, while older and more obscure titles will be seldom accessed. Thus a policy of replacing the least recently used titles to make room in the cache for requested items is likely to be effective.

Additionally, for a large class of applications, the user access pattern is often predictable well in advance. For example, an instructor may predict that recent class lectures as well as material related to an upcoming test are more likely to be accessed than other class material. Distributed hierarchical storage extends these ideas by distributing multiple magnetic disk-based caches across a network. Although distributed caches in general must deal with cache consistency, this problem will not apply to most CM applications, which will generally be read-only or will have single-user access (it's hard to imagine widespread demand for simultaneous editing of the same audio or video file among multiple users).

The architecture of a proposed distributed hierarchical storage management system will consist of several video storage servers that act as on-line cache for information stored permanently on archive servers (see Figure 6).[9, 10] In addition to maintaining one or more tertiary storage devices that contain the video files as well as the corresponding metadata, each archive server will also provide an interface to let users query the database to locate pertinent video files and schedule their retrieval.

Table 3. Tertiary storage devices in a multimedia system.

Feature	Magnetic Disk	Optical Disk	Low-end Tape	High-end Tape
Capacity	9 Gbytes	200 Gbytes	500 Gbytes	10 Tbytes
Mount time	None	20 seconds	60 seconds	90 seconds
Transfer rate (per second)	2 Mbytes	300 Kbytes	100 Kbytes	1 Mbyte
Cost	$5,000	$50,000	$50,000	$500,000–$1,000,000
Cost per Gbyte	$555	$125	$100	$50

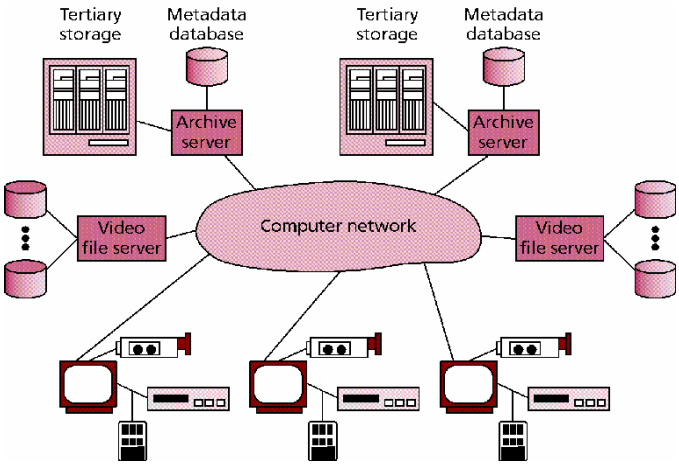

Figure 6. Architecture of a distributed hierarchical video-on-demand server.

Originally published in *Computer*, May 1995 , pp. 40–49.

Implementing a multimedia file system

Designers of a multimedia file system must concern themselves with client/server interaction, tracking data through file structures, and creating, editing, and retrieving multimedia objects.

Interfacing with the client

Multimedia storage servers can be classified as *file-system oriented* or *stream oriented*. A client of a file-system oriented server sees the multimedia object as a large file and uses typical file system operations such as open, close, and read to access the file. The client issues read requests to the server periodically to read data from the file. The server can use the open operation to enforce admission control and initiate prefetching of the multimedia file. The server can also periodically prefetch from the disk system into memory buffers to service read requests with minimal delay. In this model, the client can implement operations, such as pause and resume, by simply ceasing to issue read requests.

The client of a stream-oriented server issues commands such as *play*, *pause*, and *resume* to the server. The server uses the stream concept to deliver data continuously to the client. After the client initiates stream playback, the server periodically sends data to the user at the selected rate without further read requests from the user.

Moving data is another key issue in the client/server interface. Typically, data being transferred from one process (such as the server kernel) to another process (such as the client) is copied. For CM streams, copying is unnecessary, takes extra time, and produces extra traffic on the system bus. Because of CM's high throughput requirements, it is desirable to share memory or remap the memory into another address space to avoid data copying.

File-retrieval structures

A fundamental issue in implementing a file system is to keep track of which disk blocks belong to each file, keeping a map, essentially, of how to travel from block to block in a file. Of course this is not a concern for contiguous files. File mapping for scattered files can be accomplished in several ways, each with its own merit.

A simple solution for mapping blocks is a *linked list*, where each block contains a pointer to the next block in the file. The file descriptor only needs to contain a pointer to the first block of the file. A serious limitation of this approach, however, is that random access is highly inefficient as accessing a random block requires accessing all previous blocks.

To improve random-access performance, some conventional file systems (like DOS) have used a file allocation table, with a table entry for each block on the disk. Each table entry maintains a pointer to the next block of a file. Assuming that the entire FAT is kept in main memory, random access can be very fast. However, it might not be possible to keep a FAT in main memory for the large file systems expected in multimedia servers.

Indexes. A FAT contains information about the entire file system, but only a portion of this information relating to currently open files is needed. Storing an index for each file separately (for instance, I-nodes in Unix) can exploit this situation. These indexes can be simple lists or hierarchical structures such as binary trees (to make the process of searching more efficient). Rapid random access is still possible, but the need to keep the entire FAT in main memory is lessened.

A potential drawback to storing file indexes separately occurs when all open file indexes cannot be kept in main memory in their entirety, as is possible with large CM files. (For example, a server with a small selection of long videos may have all the videos open for playback at once. The open indexes would therefore map all the allocated file space.) Retrieving a CM file involves retrieving blocks of the index in real time, in addition to the blocks of the file itself. It is true that the index retrieval demands less in terms of bandwidth, but it nonetheless consumes resources. In fact, managing such small bandwidth streams might require special algorithms to keep them from using a disproportionate amount of system resources.

A hybrid solution. An obvious solution to excessively using resources is to implement a linked list so that real-time playback can follow pointers contained in the data blocks. Random seeks can be achieved quickly through the index without reserving real-time resources. This would add system overhead in keeping both the index and the link pointers up to date, but for applications that perform little editing, such as video-on-demand, the overhead might be worthwhile. (To support fast forward and rewind, it may be necessary to store extra pointers, as the blocks will not be visited in normal sequential order.)

Finally, a multimedia server must accommodate the fact that each multimedia object can contain media information in various formats (such as video, audio, and textual). Besides maintaining file maps for each media file, the server must maintain characteristics of each multimedia object, such as its creator, length, access rights and, most important, intermedia synchronization relationships.

Originally published in *Computer*, May 1995 , pp. 40–49.

Editing multimedia objects

Multimedia objects comprise media components (such as audio or video files, images, and text) that are presented to the user in a coordinated fashion. When a large media component is copied into more than one object, the copying operation consumes significant time and space. To minimize copying, the multimedia file system can consider media component files immutable and enable editing by manipulating pointers to the media component files (or portions of files). Once a media component file has no multimedia object referring to it, it can be deleted by the server to reclaim memory. A "garbage collection" algorithm that uses a reference count mechanism called *interests*, such as the one presented by Terry and Swinehart[11] in the Etherphone system, can be used for this purpose.

When performing small insertions, using a pointer may not be worthwhile (or feasible) in terms of maintaining continuous playback. Also, CM files might need small deletions. As small as such operations might be, the naive approach—simply rewriting the file from the edited point on—can be extremely time-consuming for large CM files. If the section being inserted or deleted is an integral number of blocks, then the file map could simply be modified, but usually the section will not be such a convenient size. It is possible to perform insertion and deletions in time, proportional to the size of the insertion/deletion rather than to the whole file, by implementing a scheme where blocks must be filled to a certain minimum level to support continuous retrieval. The insertion/deletion will consist of some number of full blocks plus a remaining partially filled block. All the blocks are then inserted/deleted by modifying the file map, and then data is distributed among adjacent blocks of the file to meet the required fill level.

Interactive control functions. A multimedia server must also support interactive control functions such as pause/resume, fast forward, and fast backward. The pause/resume operations pose a significant challenge for buffer management because they interfere with the sharing of a multimedia stream among different viewers.[12] The fast-forward and fast-backward operations can be implemented either by playing back media at a higher rate than normal or by continuing playback at the normal rate while skipping some data. Since the former approach can significantly increase the data rate, its direct implementation is impractical. The latter approach, on the other hand, can also be complicated by the presence of interdata dependencies (for example, compression schemes that store only differences from previous data).

Achieving fast forward. Several approaches can achieve fast forward through data skipping. One method is to create a separate, highly compressed (and lossy) file. For example, the MPEG-2 draft standard proposes the creation of special, highly compressed *D* video frames that do not have any interframe dependency to support video browsing. During retrieval, when fast-forward operation is required, the playback would switch from the normal file (which could itself be compressed but still maintain acceptable quality levels) to the highly compressed file. This option is attractive because it does not require any special storage methods or file postprocessing. It does however require additional storage space and, moreover, the resulting output has poor resolution because of the high compression.

Another way to achieve fast forward is to categorize each block as either relevant or irrelevant to fast forward. During normal operation, both types of blocks are retrieved, and the media stream is reconstructed by recombining the blocks either in the server or in the client station. Alternatively, during fast-forward operation only the fast-forward blocks are retrieved and transmitted.

Scalable compression schemes are readily adapted to this sort of use, although the drawback here is that it poses additional overheads for splitting and recombining blocks. Furthermore, with compression schemes that store differences from previous data, most data will be relevant to fast forward. For example, the *I* and *P* frames of MPEG are much larger than the average frame size. This means that the data rate required during fast-forward operations would be higher than normal.

Chen, Kandlur, and Yu[13] offer a different solution for fast-forward operations on MPEG video files. Their method performs block skipping through an intelligent arrangement of blocks (called segments) that takes into account the interframe dependencies of the compressed video. Entire video segments are skipped during fast-forward operations, and the viewer sees normal resolution video with gaps. Their solution also addresses the placement and retrieval of blocks on a disk array using block interleaving.

Conclusion

Multimedia storage servers differ from conventional storage servers to an extent that requires significant changes in design. Graphical user interfaces have already tremendously influenced computing, calling for faster and more efficient hardware, and for specialized algorithms. Multimedia interfaces, and CM in particular, are even more revolutionary because they introduce real-time demands and consume system resources in unprecedented quantities.

Originally published in *Computer,* May 1995 , pp. 40–49.

Commercially available multimedia server products underscore progress made thus far. For example, in the LAN environment, products like IBM's LANServer Ultimedia serve video and audio to suitably equipped PCs. In the video-on-demand arena, there are products like Oracle's Media Server, which is slated to deliver approximately 25,000 video streams.

The study of multimedia systems continues to flourish and to confirm that merely tacking multimedia onto conventional systems is inadequate. If multimedia is to succeed, fundamental changes must be made with respect to real-time issues: The services supported by an operating system or network must be expanded, data must be stored and retrieved for real-time retrieval rates and to meet client expectations, and user interfaces must be rethought once again to fulfill multimedia's promise of interactivity.

Acknowledgments

The authors acknowledge the coordination and detailed suggestions of Arturo Rodriguez. This work was partially supported by MPR Teltech Ltd. and the British Columbia Science Council.

References

1. 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.
2. D.J. Gemmell and S. Christodoulakis, "Principles of Delay Sensitive Multimedia Data Storage and Retrieval," *ACM Trans. Information Systems*, Vol. 10, No. 1, 1992, pp. 51-90.
3. A.L. Narasimha Reddy and J.C. Wyllie. "I/O Issues in a Multimedia System," *Computer*, Vol. 27, No. 3, Mar. 1994, pp. 69-74.
4. D.J. Gemmell and J. Han, "Multimedia Network File Servers: Multichannel Delay Sensitive Data Retrieval," *Multimedia Systems*, Vol. 1, No. 6, Apr. 1994, pp. 240- 252.
5. P. Lougher and D. Shepherd, "The Design of a Storage Server for Continuous Media," *The Computer J.*, Vol. 36, No. 1, Feb. 1993, pp. 32-42.
6. H.M. Vin and P. Venkat Rangan, "Designing a Multi-User HDTV Storage Server," *IEEE J. Selected Areas in Comm.*, Vol. 11 No. 1, Jan. 1993, pp. 153-164.
7. P. Venkat 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.
8. T. Mori et al., "Video-on-Demand System using Optical Mass Storage System," *J. Applied Physics*, (Japanese) Vol. 1, No. 11B, Nov. 1993, pp. 5,433-5,438.
9. C. Federighi and L.A. Rowe, "The Design and Implementation of the UCB Distributed Video-On-Demand System," *Proc. IS&T/SPIE 1994 Int'l Symp. Electronic Imaging: Science and Technology*, Int'l Soc. for Optical Eng., P.O. Box 10, Bellingham, Wash., 98227-0010, 1994, pp. 185-197.
10. L.A. Rowe, J. Boreczky, and C. Eads, "Indexes for User Access to Large Video Databases," *Proc. IS&T/SPIE 1994 Int'l. Symp. Electronic Imaging: Science and Technology*, Int'l Soc. for Optical Eng., P.O. Box 10, Bellingham, Wash., 98227-0010, 1994, pp. 150-161.
11. D.B. Terry and D.C. Swinehart, "Managing Stored Voice in the Etherphone System," *ACM Trans. Computer Systems*, Vol. 6, No. 1, Feb. 1988, pp. 3-27.
12. A. Dan, D. Sitaram and P. Shahabuddin, "Dynamic Batching Policies for an On-Demand Video Server," *Proc. ACM Multimedia 94*, ACM Press, New York, 1994, pp. 15-24.
13. M. Chen, D.D. Kandlur, and P.S. Yu, "Support for Fully Interactive Playout in a Disk-Array-Based Video Server," *Proc. ACM Multimedia 94*, ACM Press, New York, 1994.

D. James Gemmell is a PhD candidate at Simon Fraser University. His research interests include delay-sensitive multimedia systems, focusing on server storage and retrieval.

Gemmell received a BSc degree from Simon Fraser University in 1988 and an MSc degree from the University of Waterloo in 1990, both in computer science.

Harrick M. Vin is an assistant professor of computer science and the director of the Distributed Multimedia Computing Laboratory at the University of Texas at Austin. His research interests are multimedia systems, high-speed networking, mobile computing, and large-scale distributed systems. He has co-authored more than 35 papers in leading journals and conferences on multimedia systems.

Vin received a BTech degree in computer science and engineering in 1987 from the Indian Institute of Technology in Bombay. He received an MS in 1988 from Colorado State University and a PhD in 1993 from University of California at San Diego, both in computer science.

Dilip D. Kandlur is on the research staff at the IBM T. J. Watson Research Center in Yorktown Heights, N.Y., where he has worked on designing and implementing the Multimedia Multiparty Teleconferencing system. His research interests include video/audio support for desktop collaboration, multimedia networking, and multimedia storage management.

Kandlur received a BTech degree in computer science and engineering from the Indian Institute of Technology in Bombay in 1985. He received MSE and PhD degrees, also in computer science and engineering, from the University of Michigan, Ann Arbor, in 1987 and 1991 respectively. Kandlur is a member of the IEEE Computer Society.

P. Venkat Rangan founded and directs the Multimedia Laboratory at the University of California at San Diego, where he is also associate professor of computer sci-

Originally published in *Computer*, May 1995 , pp. 40–49.

ence. His research interests include multimedia on-demand servers, media synchronization, and multimedia communication and collaboration.

Rangan received a PhD degree in computer science from the University of California at Berkeley in 1988, and a BTech degree in electrical engineering from the Indian Institute of Technology in Madras in 1984, where he received the President of India gold medal. Rangan recently received the National Science Foundation Young Investigator Award. He is editor-in-chief of the ACM/Springer-Verlag *Multimedia Systems* journal. Rangan is a member of IEEE.

Lawrence A. Rowe is a coguest editor of this theme issue. His biography appears following the guest editors' introduction.

Readers can contact Gemmell at the School of Computer Sci., Simon Fraser Univ., Burnaby, B.C. Canada, V5A 1S6, e-mail dgemmell@sfu.ca; Vin at vin@cs.utexas.edu; Kandlur at IBM T.J. Watson Research Ctr., 30 Saw Mill River Rd., Hawthorne, N.Y. 10532, kandlur@watson.ibm.com; and Rangan at Computer Sci. Dept., Univ. of California at San Diego, 92093-0114, venkat@chinmaya.ucsd.edu.

Originally published in *Computer,* May 1995 , pp. 40–49.