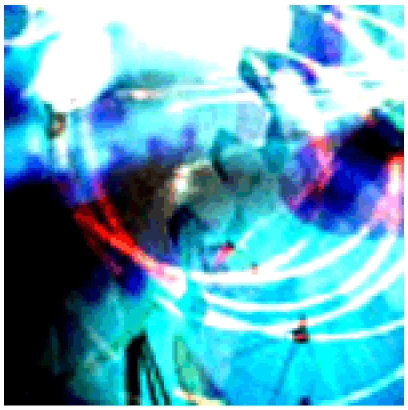
IEEE DISTRIBUTED SYSTEMS ONLINE 1541-4922 © 2004 IEEE Published by the IEEE Computer Society
Vol. 5, No. 1; January 2004

Storage System Support for Continuous-Media Applications, Part 1: Requirements and Single-Disk Issues

Pål Halvorsen, *University of Oslo*
Carsten Griwodz, *University of Oslo*
Vera Goebel, *University of Oslo*
Ketil Lund, *University of Oslo*
Thomas Plagemann, *University of Oslo*
Jonathan Walpole, *Oregon Health & Science University*



A multimedia storage system plays a vital role for the performance and scalability of multimedia servers. To handle the server load imposed by increased user access to on-demand multimedia streaming applications, new storage system solutions are needed.

Internet use and the amount of data that users download and stream from Internet servers are rapidly increasing. Analysts predict that by 2005, the World Wide Web and applications such as news- and video-on-demand (VoD) will constitute approximately 50 percent of the total available data stored.[1] Although mid-priced PCs can handle the load that multimedia applications impose on the client system, the potentially high number of concurrent users downloading or streaming data from media-on-demand servers is a challenge for a multimedia server's storage system.

Multimedia storage systems store and retrieve data from storage devices and manage related issues including data placement, scheduling, file management, continuous data delivery, memory buffering, and prefetching. For high-data-rate multimedia systems, storage systems have long been viewed as a primary bottleneck for two reasons. First, multimedia applications have a much higher storage system load than previous applications. Second, storage devices have become only marginally faster compared to increased processor and network performance. This increasing speed mismatch has fueled a search for new storage structures and file system storage and retrieval mechanisms.

Developers who design and implement multimedia storage systems must consider several issues, including

- What kind of storage device to use
- How to order the requests
- Where to put data
- How to manage memory
- How to deal with overload situations
- What kind of metadata (index) structures to use

The decisions they make for each subcomponent often depend on the expected access patterns, but to avoid conflicting designs, they must also consider the choices they make for other components.

Although researchers have explored multimedia storage techniques and research directions elsewhere,[1,2] such discussions often address particular system components. Also, many recent technological developments have not yet been addressed. In this two-part series, we offer an overview of current multimedia storage system technologies, including a general classification of real-time and continuous-media storage technologies, as a starting point for further study (for a comprehensive reference list, see www.ifi.uio.no/~paalh/publications/dsonline2004). We also consider emerging trends in storage systems, proposed system properties, and how different mechanisms fit together. In this part, we address streaming-system requirements and single-disk issues such as disk scheduling and data placement.

STORAGE AND RETRIEVAL OF MULTIMEDIA DATA

A multimedia storage system must account for both storage and bandwidth capacity for optimal device utilization, and developers have proposed different mechanisms to optimize the storage system in the multimedia context and achieve continuous media support. Compared to discrete data, continuous-media data entails new requirements related to timeliness, large file sizes, high data rates, and the need for low latency and (intra and inter) file synchronization. Researchers often characterize multimedia applications as soft real-time applications because they require timely behavior. In applications playing out continuous media streams, for example, each task requires periodic operations that give a certain data amount at each time interval. However, traditional periodic tasks have a fixed frequency and a constant time, whereas multimedia periodic behavior is more complex because period frequency and length can vary. Moreover, user interactions make future resource requirements hard to predict—a trend that will likely continue as developers increasingly use multimedia data to present information in a user-friendly way.

Continuous media: A brief overview

Continuous media is characterized by a timely presentation, such as displaying a video frame every 40th millisecond for a 25 frame per second video playout. The requirements for storing continuous-media data on disk and retrieving it vary, however, according to the target application's data access pattern and the encoding and file formats.

The motion-JPEG (MJPEG) encoding format, for example, separately compresses each video frame, while MPEG might replace portions of a frame with a reference to a similar portion of another frame elsewhere in the sequence. So, MPEG provides better compression ratios. However, its frames must be processed in groups, so the server should therefore handle them in groups. More recent encoding formats include capabilities such as layering, as in scalable MPEG (SPEG), and fine-grained scalability, as in MPEG-4, which make it possible to adapt to available network bandwidth, client capabilities, and server resources. However, applications that apply these encoding formats can use only part of the frames stored on disk. Thus, although these formats maintain the frame rate, they decrease quality. However, using scalable encoding formats also creates a new storage system research challenge: how to manage internally structured media data.

The file format itself contains encoded media data and other information. With simple file formats such as MPEG-1 system streams, this additional information provides only basic facts, such as how many separate streams the file contains. More recent file formats, such as MPEG-4, QuickTime, or audio video interleave (AVI) contain information about how the contained streams relate to each other, such as that two particular streams must be played together. So, media files form external and internal structures in relation to each other. Recent file formats also support conditions that require the delivery of different encoded data based on interaction with client applications. Such file formats can also refer to other files' encoded data in a similar way as it refers to encoded data included in the file itself. Handling these additional demands also creates challenges for future research.

System requirements

In today's multimedia applications, such as the on-demand class of applications, data is typically written once and read many times. Such applications are also more complicated than traditional applications and entail new properties:

- The application data structure is moving away from linear data, such as that in traditional movies, to branched, nonlinear data that lets users choose different paths through the presentation.

- As applications move from unidirectional to bidirectional, they will have an increased amount of interaction.

The current trend is that applications are basically moving away from analog distribution of linear data (such as TV broadcasts) through digital, personalized retrieval of linear data (true VoD, for example) and retrieval of branched data (as in interactive VoD), toward interaction with variable data (games, virtual words, and so on).[3] The result is several application types with slightly different requirements. In general, however, multimedia applications have several common characteristics.

Timeliness. Retrieving, computing, and presenting continuous media are time-dependent activities, and their timeliness requirements are often expressed by quality-of-service (QoS) parameters. Some applications, such as interactive action games, require low latency: data must be read before a given deadline or as soon as possible, often at the cost of storage device bandwidth utilization. In a VoD application that streams continuous media over the network, however, it might be more important to focus on data rates over deadlines and serve requests in rounds. This is because, in such applications, other subsystems might introduce much larger delays and jitter than the storage system itself, and focusing on data rates enables a more efficient request schedule. So, algorithms for storing and retrieving such data must consider the application's periodic or time constraints and provide additional buffers to smooth the data stream.

Large file sizes and high data rates. Compared to text and graphics, video and audio have very large storage space and playout-rate requirements. Because the file system must store a range of information—from small, discrete data units such as text files to large, continuous data units such as video and associated audio—it must manage the disk data to efficiently use limited device capacity. For example, uncompressed CD-quality stereo audio requires storage and delivery of 44,100 16-bit samples per second per stereo channel—approximately 1.4 Mbits per second (Mbps). Compressed video with low but acceptable quality requires at least approximately 1 Mbps using MPEG-1, for example, while an MPEG-2 DVD-quality video stream requires approximately 3.5 Mbps on average.

Multiple data streams. A multimedia system often supports different media simultaneously, such as intrastream and interstream synchronization. Not only must the system ensure that each media gets a sufficient resource share, it also must consider tight relations between different streams arriving from different sources or files. Movie retrieval, for example, requires processing and synchronizing of audio and video.

So, in addition to traditional requirements such as availability, efficiency and performance, and fairness, a multimedia system introduces several new, possibly contradicting requirements.[4] For early multimedia applications that play back a linear data object sequentially, requirements such as high throughput and no synchronization skew are important. Here, users can typically tolerate a short startup delay, but, as we move to more interactive applications, requirements such as low latency become increasingly important. We must therefore tune multimedia storage systems for high performance to support

- An optimal (maximum) number of clients
- The required data rates to support continuous playout of time-dependent data
- Highly responsive, interactive applications

Researchers have proposed several approaches and mechanisms to meet these demands. We now address these developments in some key areas of multimedia storage system research.

DISK SCHEDULING

A disk is an exclusive, nonpreemptable device: it serves one request at a time. So, to achieve maximum performance, the disk scheduler must sort and multiplex requests in the temporal domain on the basis of system characteristics. Originally, applications relied on disk scheduling to reduce latency or increase throughput or efficiency,[5] but real-time and multimedia applications include additional requirements that existing disk-scheduling algorithms cannot meet.[6] For instance, playing back continuous multimedia such as audio and video from disk requires periodic data retrieval, and the system must deliver each requested piece of data within a certain deadline to ensure continuous presentation. In a multiuser system, distributing the storage subsystem's bandwidth is also important.

With the introduction of interactive, mixed-media applications such as learning-on-demand, which integrates video, audio, and slides, the disk scheduler requirements are even more complex. Thus, QoS is a central issue in disk scheduling.[5] Performance goals are nevertheless still important—continuous multimedia data often imposes considerable requirements on disk throughput. So, the disk scheduler must now help ensure high throughput, QoS support, and low latency in a multiuser environment characterized by variable, heterogeneous workloads. In addition, these requirements partially conflict, and trade-offs are necessary. For example, the need for low latency generally conflicts with the need for efficient disk reads. While ordering requests minimizes disk head movement and thus achieves higher request throughput, low latency is achieved by serving requests immediately, without considering data placement and optimal request sorting. The result can thus be long seek operations and reduced throughput.

Algorithm classification

We can classify existing disk-scheduling algorithms in different ways, but we have chosen to classify them according to their designated purpose.

Performance-oriented algorithms focus only on optimizing performance. By sorting requests to minimize disk arm movement, for example, these algorithms can increase throughput, reduce latency, or do both. Well-known examples here include first-come, first-served (FCFS); shortest seek time first (SSTF); Scan (elevator); LOOK; VScan; and shortest access time first (SATF).

Real-time algorithms are intended for real-time environments, servicing disk requests on deadline. Examples here include earliest deadline first (EDF), Scan-EDF, and priority Scan (PScan).

Stream-oriented algorithms handle continuous-data-stream retrieval and often rely on periodic behavior, load control, and fair sharing of I/O bandwidth. Several proposed algorithms exist, including the continuous media file system (CMFS) scheduler, grouped sweep scheduling (GSS), BubbleUp, T-Scan, batched Scan (BScan), buffer-inventory-based dynamic scheduling (BIDS), and greedy-but-safe EDF (GS_EDF).

Mixed-media algorithms recognize that different disk requests might have different service-level requirements. These algorithms usually have two scheduling levels that manage different services and disk efficiency, respectively. Examples here include Cello, the massively parallel and real-time storage (MARS) scheduler, fair mixed-media scheduling (FAMISH), deadline-sensitive Scan, Fellini's disk scheduler, delta L, and the adaptive disk scheduler for mixed-media workloads (APEX).

Additionally, priority-based disk-scheduling algorithms also exist. However, support for priorities is orthogonal to our classification, so we have presented these algorithms within the existing four classes.

Analysis

With continuous media such as audio and video, pure performance-oriented algorithms such as Scan and real-time, deadline-driven algorithms such as EDF fail during high workloads. To overcome these algorithms' shortcomings, developers have designed several stream-oriented disk-scheduling algorithms that are primarily optimized for handling continuous-data-stream retrieval. Compared to real-time scheduling algorithms, these algorithms often focus less on deadlines and more on request periodicity (requests are typically served in fixed-length rounds) and fair allocation of disk bandwidth. Stream-oriented algorithms typically consider performance, and, to a certain extent, work conservation. Furthermore, they often offer a statistical real-time or throughput-only guarantee, based on periodic requests, but do not support request dropping, priorities, or deadlines. Given that the algorithms target delay-sensitive data, the lack of deadline support is compensated for by careful load control, typically through admission control and load uniformity (all requests are for video data, for example). Finally, because stream-oriented algorithms mainly address periodic access, they usually neglect low-latency support.

In recent years, developers have introduced applications that present both discrete and continuous media—and thus provide different service classes. Disk scheduling for mixed-media workloads has therefore become an active research area. Most of these algorithms have a two-level hierarchical design, with the lower level ensuring efficient disk usage and the higher handling QoS and service-class differentiation. (The number of service classes increases from the traditional, best-effort class to two classes—discrete and continuous—or even to an arbitrary number of service classes.) Some algorithms rely on the proportional-share-allocation paradigm, while still offering QoS guarantees. This is possible because they use a fixed set of service classes, or queues, and carefully control each queue's weight. The relative share of a queue's bandwidth thereby equals the absolute share, and QoS guarantees are possible if system efficiency remains constant. Some of these schedulers are also priority based and minimize device idle time using work conservation. To avoid overload situations, some include admission control. Finally, to support low latency in stream-oriented schedulers, some mixed-media algorithms use separate queues and give low-latency requests priority in the low-level queue.

Although much work has been done on disk scheduling support for multimedia applications, some areas for disk scheduling research are still open or unfinished. Some schedulers, for example, address mechanisms for admission control and service-class resource assignment that automatically adapt to varying conditions and workload mixes.[7] However, this topic remains an important research issue. Also, new disk devices are "intelligent," and the system does not necessarily know exactly where a block is stored. The devices therefore often contain a built-in hardware version of a Scan-like scheduling algorithm to optimize performance. Future disk-scheduling research should exploit these new capabilities; fine-grained request sorting, for example, might not be necessary.

DATA PLACEMENT

Data placement policies are related to disk scheduling in that disk request sorting is typically based on block placement in the storage device. Such policies aim to improve the storage system's efficiency by properly placing data elements according to device mechanics and the often-predictable (sequential) retrieval patterns of multimedia applications.

Placement policies

Optimizing block placement to minimize average data retrieval time has long been an active research area, and researchers have proposed several placement policies.

Scattered (random) placement places data elements on arbitrary disk blocks regardless of disk characteristics and access patterns.

Contiguous placement aims to store all file blocks contiguously. It might, for example, store blocks on a disk's adjacent sectors first, then on another track in the same cylinder, and finally on adjacent cylinders. So, assuming no other interleaved I/O operations to the disk exist, track-to-track seeks are the only intrafile seeks required to read data sequentially.

Extent-based placement refines the contiguous policy to better support interleaved I/O operations. An extent is a physically linear series of blocks that can be read without head repositioning, but each extent is smaller than a whole file, decreasing the length of seeks for data in other files. Additionally, each metadata structure (inode) field usually points to a series of blocks (start offset and number of adjacent blocks) reducing the number of indirect pointers to follow to find the block address. Several file systems use this policy, including the XFS, journaled file system (JFS), new technology file system (NTFS), and Minorca file system.

Cylinder-based placement allocates blocks close to each other to avoid long intrafile seeks (similar to extents). This policy organizes the blocks in cylinder groups, which consist of several adjacent cylinders. A global policy then decides which cylinder group to place a file in, and a local policy tries to put all data blocks in the same cylinder group at rotational optimal positions (when possible). To avoid fragmentation and different-loaded cylinder groups, the data block allocation shifts to a different cylinder group every few megabytes. The fast file system, for example, uses this policy.

Log-structured placement stores all file system data in a sequential, append-only log, which is the only on-disk structure. The idea is that newer, larger buffer caches will usually hold the required data blocks in memory when requested. The policy also assumes that the disk subsystem predominantly sees write operations; that is, allocating new blocks is a bottleneck. So, to optimize writes, this policy allocates the next available block with a minimum seek time when writing. The log-structured file system (LFS), for example, uses this policy.

Zone-based placement uses disk characteristics and average disk-arm position to find a suitable block. For example, the organ-pipe placement policy tries to put the most frequently accessed block—regardless of which file it belongs to—close to the disk's center. The reason is that with scheduling algorithms such as Scan, the disk head is, on average, at the center and its seek times to the platter's center are thus shorter. This means placing and periodically rearranging popular video clips in the disk's center cylinders.[8,9] To compensate for the zoned disks having better capacity and bandwidth on the outer zones, the skewed organ-pipe policy moves the most frequently accessed blocks slightly outward from the disk platters' center according to the zones' different capacities.[10] Other zoned placement policy examples include near-constant transfer time (NCTT) and track-pairing.

Constrained placement policies attempt to exploit sequential access patterns to reduce seek overhead. They do this by restricting the average distance between consecutive blocks in several cylinders.[11] For example, region-based block allocation (REBECA) partitions the disk in regions and places consecutive multimedia objects in adjacent regions to minimize seek time (at the cost of start-up latency). The strand-based model derives storage granularity and scattering parameters using device characteristics and playback rate. This model tries to store data so it can be retrieved contiguously as a strand in a single operation, placing each strand on disk according to a maximum calculated seek time (the maximum scatter value) between consecutive strands to guarantee continuous retrieval and data playout.

Analysis

Block allocation research primarily addresses disk efficiency rather than latency. Which one is best depends on access patterns, but there are some important general considerations. In a server, multiple streams will play out concurrently, and the I/O operation order—both with respect to current playout position and media object—might vary slightly (servers can probably control this variation to a certain degree). Schemes with strict access-pattern assumptions can therefore be optimized for a certain I/O operations sequence, but they might fail due to small differences between the actual, varying order of I/O operation and the assumptions. So, even though a single process might access data sequentially, the probability of reading an entire large file continuously (as one long I/O operation) is minute owing to multiple concurrent file accesses (clients). Thus, a pure contiguous block allocation—which seems initially reasonable for multimedia—gives large seek times over whole files for each access to a different file. However, seeks should be as small as possible for each I/O operation (disk access). That is, data retrieved as one operation should be stored contiguously or at least close on the platter to minimize intra-operation seeks.

Techniques such as extent- or cylinder-based placement, which provide smaller contiguous file fragments, better support interleaved I/O operations with no intraoperation seeks compared to pure contiguous policies. However, several contiguous policy extensions also exist. For example, with multimedia files, some mechanisms can attempt to store blocks contiguously, whereas other file types make no attempt to optimize placement.[12,13] As in Symphony's data-type-dependent modules, a placement policy can also add an abstraction to assist multimedia applications. Finally, having small, contiguously stored page-sized blocks permits efficient mapping and unmapping of disk space into main memory, allows for finely grained buffering decisions, and this makes mapping complex access patterns easier.[14,15]

Because on-demand multimedia servers are often dominated by read-only, sequential file accesses, write-optimized policies (such as log-structured) and no-optimized policies (such as random) are often regarded as inappropriate. This is because they allow numerous intrafile seeks in multimedia streaming scenarios that have I/O requests spanning several blocks. However, when systems use replication and wide striping in a disk array to handle bottlenecks, even a random placement with many blocks might be sufficient.[16] Random placement has also been successfully used for a mixed-media scenario combined with a (heterogeneous) multidisk storage system in which data blocks are allocated randomly on a disk and replicated on random disks.[17,18]

Because the key issue is to reduce intraoperation seek time, we must determine how large each contiguous region, or extent, should be. For example, in a variable-bit-rate scenario, conventional fixed-sized clusters correspond to varying amounts of time, depending on the achieved compression.[19] Alternatively, the system can store data in variably sized clusters corresponding to fixed time periods to better support round-based retrieval. Constant retrieval time for each segment can be achieved using the different disk zones, for example, as in the NCTT policy. Additionally, when compressed data does not correspond to an even number of disk sectors, we must contend with the data-packing problem.[2] Nevertheless, each extent's size should be equal to or exceed the average I/O operation request size such that at most one (possibly track-to-track) seek is performed between two extents (assuming that one extent is stored in a track or cylinder).

When a multimedia scenario streams out various continuous-media objects, combining policies might be appropriate. For example, we might minimize intraoperation seeks using I/O request-sized extents, then place the most popular files using a popularity-based (zone-based) scheme to best utilize a modern disk's different. We could also use a constrained-based policy to achieve a worst-case interextent seek time if the popularity-based placement did not implicitly provide one.

CONCLUSION

With this data placement discussion, we conclude our look at existing solutions for handling multimedia data on a single disk. As our survey of different mechanisms shows, multimedia applications' timeliness demand requires different strategies compared to classical workloads. As for future research, in addition to multimedia disk scheduling interactions with intelligent disks, some next steps in research might include integrating existing data placement schemes, and placement of media data fragments with internal and external structural dependencies.

Part Two of our survey will cover solutions for multimedia data management for multiple disks, including techniques for striping, interleaving, replication, and load balancing. We will also discuss main memory's role in storage systems, and metadata structures' relevance for efficient multimedia data handling. We will conclude this survey by outlining existing approaches for combining all these building blocks into a complete multimedia storage system, and offer a view toward future research.

References

1. G.A. Gibson , J.S. Vitter, and J. Wilkes , "Strategic Directions in Storage I/O Issues in Large-Scale Computing," *ACM Computing Surveys* , vol. 28, no. 4, Dec. 1996, pp. 779-793.
2. J. Gemmell and S. Christodoulakis , "Principles of Delay-Sensitive Multimedia Data Storage Retrieval," *ACM Trans. Information Systems* , vol. 10, no. 1, Jan. 1992, pp. 51-90.
3. T.D.C. Little and D. Venkatesh , "Popularity-Based Assignment of Movies to Storage Devices in a Video-on-Demand System," *Multimedia Systems J.* , vol. 2, no. 6, Jan. 1995, pp. 280-287.
4. H. Schulzrinne , "Operating System Issues for Continuous Media," *Multimedia Systems J.* , vol. 4, no. 5, Oct. 1996, pp. 269-280.

5. D.J. Gemmell , et al., "Multimedia Storage Servers: A Tutorial," *Computer* , vol. 28, no. 5, May 1995, pp. 40-49. http://csdl.computer.org/comp/mags/co/1995/05/r5040abs.htm

6. 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. http://csdl.computer.org/comp/mags/co/1994/03/r3069abs.htm

7. K. Lund and V. Goebel , "Adaptive Disk Scheduling in a Multimedia DBMS," *Proc. 11th ACM Int'l Conf. Multimedia* (ACM 03), ACM Press, 2003, pp. 65-74.

8. S. Akyurek and K. Salem , "Adaptive Block Rearrangement," *ACM Trans. Computer Systems (TOCS)* , vol. 13, no. 2, May 1995, pp. 89-121.

9. R. Geist , et al., "Disk Performance Enhancement through Markov-Based Cylinder Remapping," *Proc. 30th ACM Ann. Southeast Regional Conf.* , ACM Press, 1992, pp. 23-28.

10. R. Tewari , et al., "Placement of Multimedia Blocks on Zoned Disks," *Multimedia Computing and Networking 1996* , Proc. SPIE 2667, Int'l Soc. for Optical Eng., 1996, pp. 360-367.

11. R. Anderson , Y. Osawa, and R. Govindan , "A File System for Continuous Media," *ACM Trans. Computer Systems (TOCS)* , vol. 10, no. 4, Nov. 1992, pp. 311-337.

12. P.V. Rangan and H.M. Vin , "Designing File Systems for Digital Video and Audio," *Proc. 13th ACM Symp. Operating Systems Principles (SOSP 91)* , ACM Press, 1991, pp. 81-94.

13. P.V. Rangan and H.M. Vin , "Efficient Storage Techniques for Digital Continuous Multimedia," *IEEE Trans. Knowledge and Data Eng.* , vol. 5, no. 4, Aug. 1993, pp. 564-573. http://csdl.computer.org/comp/trans/tk/1993/04/k0564abs.htm

14. C. Martin , et al., "Multimedia Information Storage and Management," *The Fellini Multimedia Storage Server* , S.M. Chung, ed., Kluwer Academic Publishers, 1996.

15. S. Ghandeharizadeh , et al., "A Scalable Continuous Media Server," *Multimedia Tools and Applications* , vol. 5, no. 1, July 1997, pp. 79-108.

16. R. Haskin , "Tiger Shark-A Scalable File System for Multimedia," *IBM J. Research and Development* , vol. 42, no. 2, 1998, pp. 185-197.

17. Y. Birk , "Random Raids with Selective Exploitation of Redundancy for High-Performance Video Servers," *Proc. Int'l Workshop Network and O.S. Support for Digital Audio and Video* (NOSSDAV), 1997, pp. 13-23.

18. J.R. Santos and R.R. Muntz , "Performance Analysis of the Rio Multimedia Storage System with Heterogeneous Disk Configurations," *Proc. 6th ACM Int'l Conf. Multimedia* (MM 98), ACM Press, 1998, pp. 303-308.

19. T.C. Bell , et al., "The MG Retrieval System: Compressing for Space and Speed," *Comm. ACM* , vol. 38, no. 4, Apr. 1995, pp. 41-42.

Pål Halvorsen is an associate professor in the University of Oslo's Department of Informatics. His research activities focus mostly on distributed-multimedia-system support, including operating systems, storage and retrieval, communication, and distribution. He received his doctoral degree in computer science from the University of Oslo. Contact him at the Dept. of Informatics, Univ. of Oslo, PO Box 1080 Blindern, N-0316 Oslo, Norway; paalh@ifi.uio.no.

Carsten Griwodz is an associate professor in the University of Oslo's Department of Informatics. His research interests include improved mechanisms and algorithms for media servers and distribution systems. He received his doctoral degree from the Darmstadt University of Technology. Contact him at the Dept. of Informatics, Univ. of Oslo, PO Box 1080 Blindern, N-0316 Oslo, Norway; griff@ifi.uio.no.

Vera Goebel is a professor in the University of Oslo's Department of Informatics. Her research interests include multimedia database systems and operating systems, multimedia middleware, quality of service, and interactive distributed multimedia applications. She received her PhD in computer science from the University of Zurich. Contact her at the Dept. of Informatics, Univ. of Oslo, P.O. Box 1080 Blindern, N-0316 Oslo, Norway; goebel@ifi.uio.no.

Ketil Lund works as a post-doctoral researcher at the UniK-University Graduate Center. His research interests include multimedia database systems, disk scheduling, quality of service, and video on demand. He received his PhD in computer science from the University of Oslo, where his doctoral work focused on disk scheduling for multimedia database management systems. Contact him at UniK-Univ. Graduate Center, PO Box 70, N-2027 Kjeller, Norway; ketillu@unik.no.

Thomas Plagemann is a professor in the University of Oslo's Department of Informatics. His research interests include multimedia middleware, protocol architectures, quality of service, operating system support for distributed multimedia systems, content distribution infrastructures, and interactive distance learning. He received his Doctor of Technical Science from the Swiss Federal Institute of Technology (ETH) Zurich. Contact him at the Dept. of Informatics, Univ. of Oslo, P.O. Box 1080 Blindern, N-0316 Oslo, Norway; plageman@ifi.uio.no.

Jonathan Walpole is a full professor in the Computer Science and Engineering Department and is the director of the Systems Software Laboratory at the OGI School of Science and Engineering at Oregon Health & Science University. His research focuses on adaptive systems software and its application in distributed, mobile, and multimedia computing environments. He received his PhD in computer science from Lancaster University, UK. Contact him at the Dept. of Computer Science and Eng., Oregon Graduate Inst. of Science and Technology, 20000 N.W. Walker Rd., Beaverton, OR 97291-1000; walpole@cse.ogi.edu.