UNIVERSITY OF OSLO
Department of Informatics

Storage Systems Support for Multimedia Applications

Research Report No. 307,
ISBN 82-7368-259-5,
ISSN 0806-3036

Pål Halvorsen, Carsten Griwodz,
Ketil Lund, Vera Goebel,
Thomas Plagemann

Storage Systems Support for Multimedia Applications

Pål Halvorsen[†], Carsten Griwodz[†], Ketil Lund[‡], Vera Goebel[†], Thomas Plagemann[†]
[†]IFI, University of Oslo, Norway
[‡]UniK, University of Oslo, Norway
Email: {paalh, griff, goebel, plageman}@ifi.uio.no, ketillu@unik.no

Abstract

Lately, on-demand streaming multimedia applications have become very popular. Contemporary personal computers can handle the load imposed by such multimedia applications on the client side, but the potentially high number of concurrent users accessing a server represents a generic problem. The multimedia storage system is responsible for storage and retrieval of multimedia data from storage devices, and plays a vital role for the performance and scalability of multimedia servers. It deals with issues related to data placement, scheduling, file management, continuous data delivery, buffer management, prefetching, etc., and with the particular demands of multimedia applications, such as real-time characteristics, large file sizes, high data rates, and several data sources. Performing these tasks and supporting these requirements appropriately are burdened by an increasing speed mismatch between processors and the most prolific and affordable storage devices, – magnetic disks –, and by the introduction of new requirements in new multimedia scenarios.

In this article, we give a survey of storage system support for continuous media applications and discuss issues related to integration of different mechanisms for the future multimedia storage systems.

1 Introduction

In the last years, there has been a tremendous growth in the use of Internet services. In particular, the world-wide web and applications like News- and Video-on-Demand have become very popular where the estimated fraction digital multimedia data of the total available data stored will increase to about 50% in 2005 [38]. Thus, the number of users, as well as the amount of data each user downloads or streams from servers in the Internet, is rapidly increasing. Today, contemporary mid-price personal computers are capable of handling the load that such multimedia applications impose on the client system. However, the potentially (very) high number of concurrent users that download data from Media-on-Demand (MoD) servers represents a generic problem for this kind of client-server applications.

When designing and implementing a storage system for such a system, several questions must be asked, e.g., 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 meta-data (index) structures to use, etc. The decisions made for each subcomponent are often dependent on the expected access patterns, but also on the choices made for the other components, i.e., some design alternatives might be contradicting. The multimedia storage system is responsible for storage and retrieval of multimedia data from storage devices. Thus, it deals with issues for example related to data placement, scheduling, file management, continuous data delivery, memory buffering, prefetching, etc. However, the storage system has for a long time been viewed upon as one of the main bottlenecks in high data rate multimedia systems. This is because compared to the increased performance of processors and networks, storage devices have become only marginally faster[1]. The effect of this increasing speed mismatch is the search for new storage structures, and storage and retrieval mechanisms with respect to the file system. Additionally, in a

[1]From year 1990 to 2000, the high performance disks have improved for example capacity from about 1 GB to 35 GB, spindle speed from about 5400 RPM to about 15.000 RPM, seek time from about 5.5 ms to 2 ms, and transfer rate from about 28 Mbps to 392 Mbps [26, 42, 88]. However, the Intel processors, for example, have improved from 486 (about 25-50 MHz) to pentiumIV (about 1.4-1.5 GHz) in the same period where the data buss width is also increased from 32 to 64 bits [21, 41]. Thus, disks have had large improvements in capacity and data density, but the disk bandwidth improvements are an order of magnitude smaller than the CPU improvements.

Research Report No. 307, Department of Informatics, University of Oslo, Norway (ISBN 82-7368-259-5, ISSN 0806-3036)

multimedia scenario, new requirements are introduced as continuous media data are different from discrete data, e.g., they may have *real-time characteristics*, *large file sizes*, *high data rates*, and *several data sources*.

Early work on storage system issues addressed different solutions for mechanisms providing like fairness between tasks, increased total system performance, etc. However, as one can see, multimedia system support raises several additional requirements compared to systems handling discrete data only. There are different ways to achieve continuous media support. Multimedia applications are often characterized as (soft) real-time applications, because they require timely behavior. Thus, each task require periodic operations giving a certain amount of data each time interval. However, variable bit rate (VBR) coding schemes like MPEG complicates the periodic behavior. Video frames are (usually) generated in a fixed frequency, but the size of each frame and even the frame patterns in the group of picture (GOP) varies [29]. Within an MPEG GOP, we have I-, B-, and P-frames all having different dependencies and sizes, and the size of each coded frame is dependent of the amount of scene changes between each frame, e.g., an action scene might give larger frames compared to a constant landscape view scene. Moreover, user interactions also makes it hard to predict future resource requirements, and this trend will probably continue as multimedia data is more frequently used to represent information in a user-friendly way. Thus, a multimedia storage system must take into account the storage capacity as well as bandwidth capacity for optimal utilization of the devices.

In [38], several existing techniques (e.g., scheduling, prefetching, and caching) and strategic research directions (e.g., exploiting data access pattern) are pointed out for storage systems where it is believed that the fraction of storage systems dealing with digital multimedia data will increase heavily in the future. In this article, we address several of these techniques and give an overview over recent developments in storage system support for multimedia applications, i.e., both means for increasing the performance and supporting the necessary time sensitive operations of multimedia servers. Several issues of real-time retrieval from disks for multimedia streaming have been explained in [34], and many research groups have looked into real-time and continuous media storage systems mechanisms. It is therefore not possible to address all proposed solutions in detail in a single paper. Instead, this survey is meant to serve as an "entry-point" for further studies where we give some examples and pointers (references) to more literature. Additionally, we try to (theoretically) integrate the different mechanisms for a future multimedia storage system by discussing the different properties of the mechanisms proposed in literature and see how they fit together.

The rest of this paper is structured as follows: Section 2 summarizes multimedia application requirements, and Section 3 presents various disk scheduling algorithms. In Section 4, we address several aspects of placing data on storage devices. Section 5 describes different buffer management mechanisms, and extensions to traditional file meta-data structures are presented in 6. Issues for designing an system integrating several of the proposed mechanisms are addressed in Section 7. Finally, we summarize and give an outline for further research in Section 8.

2 Requirements

In this section, we briefly discuss the requirements that multimedia applications impose. Typically, data is written once and read many times sequentially, i.e., at least in the class of on-Demand applications. Furthermore, the classification and evolution of multimedia applications often can be done via two parameters or axes:

- The **data structure** going from linear data like traditional movies to branched, non-linear data where the user may choose different paths through the presentation.

- Coming applications have an increasing amount of **interaction** going from unidirectional applications to bidirectional applications.

The common direction is that we are going from analog distribution of linear data (TV broadcasts) through digital, personalized retrieval of linear data (true VoD, etc.), retrieval of branched data (interactive VoD), towards interaction with variable data (games, virtual words, etc.). As one can see, there exist several types of applications with slightly different requirements, but in general, a multimedia application additionally has the following characteristics/requirements compared to traditional applications:

1. **Real time characteristics:** The retrieval, computation, and presentation of continuous media is time-dependent. The data must be presented (read) before a well-defined deadline with small jitter only. Thus, algorithms for the storage and retrieval of such data must consider time constraints, and additional buffers to smooth the data stream must be provided.

2. **Large file sizes and high data rates:** Compared to text and graphics, video and audio have very large storage space and playout rate requirements. Since the file system has to store information ranging from small, unstructured units like text files to large, highly structured data units like video and associated audio, it must manage the data on disk in a way that efficiently uses the limited storage device capacity. For example, uncompressed CD-quality stereo audio requires storage and delivery of 44100 16-bit samples per second per stereo channel (about 1.4 Mbps) [102]. Low but acceptable quality compressed video requires at least about 1 Mbps for example using MPEG-1, and an MPEG-2 DVD quality video stream requires about 3.5 Mbps in average [67].

3. **Multiple data streams:** A multimedia system must support different media simultaneously. Not only must it ensure that all of them get a sufficient share of the resources, it also must consider tight relations between different streams arriving from different sources or files. The retrieval of a movie, for example, requires the processing and synchronization of both audio and video.

Thus, in addition to traditional requirements like *efficiency* (high performance) and *fairness*, a multimedia system introduces several new requirements [68, 87]. For the first class of applications, playing back a linear data object sequentially, requirements like *high throughput* and *no synchronization skew* are important (a user can normally tolerate a short startup delay), but as we go to more interactive applications requirements like *low latency* becomes increasingly more important. It is therefore important to tune the multimedia storage system for high performance to support an optimal (maximum) number of clients, to give required data rates in order to support continuous playout of time-dependent data, and to support interactive applications by having high responsiveness. In the rest of this survey, we address some of the approaches and mechanisms proposed for disk-based storage systems to approach the demanding multimedia application requirements.

3 Disk Scheduling

The disk is an exclusive, non-preemptable device, i.e., it serves one request at a time. Thus, requests must be sorted and multiplexed in the temporal domain according to system characteristics to achieve maximum performance. Originally, disk scheduling was employed to reduce latency, or increase throughput or efficiency [33], but with the advent of real-time and multimedia applications, additional requirements emerged, which the existing disk scheduling algorithms were unable to meet [79]. For instance, playing back continuous multimedia like audio and video from disk requires periodic retrieval of data, and each requested piece of data must be delivered within a certain deadline to ensure continuous presentation. In a multi-user system, distributing the bandwidth of the storage subsystem also becomes important. With the introduction of interactive, mixed-media applications, such as Learning-on-Demand (LoD) applications [112], the requirements on the disk scheduler become even more complex. In other words, disk scheduling algorithms for multimedia applications must optimize special multimedia data criteria in addition to the traditional criteria, i.e., QoS has become a central issue in disk scheduling [33, 101]. Nevertheless, the performance goals are still of importance, since continuous multimedia data often impose considerable requirements with respect to disk throughput. Thus, the disk scheduler now has to help ensuring high throughput, QoS support, and low latency in a multi-user environment characterized by non-homogeneous and varying workloads. In addition, these requirements are partly contradicting, meaning that trade-offs must be made. For instance, the need for low latency is generally in contradiction to the need for efficient disk reads. The reason for this is that while higher throughput is achieved by minimizing disk head movement through ordering of the requests, low latency is achieved by serving requests immediately and often out of order, i.e., without taking the placement of the data into consideration. Thus, long seek operations, and thereby reduced throughput, may be the result.

In our study of disk scheduling, we have chosen to classify the existing algorithms according to the purpose for which they are designed, and we four main classes:

- **Performance-oriented algorithms** which only focus on optimizing performance, i.e., increasing throughput and reducing latency. Some well known examples are *first come first served (FCFS)* [104], *shortest seek time first (SSTF)* [104], *SCAN* (*elevator*) [27], *LOOK* [65], *VSCAN* [31], and *shortest access time first (SATF)* [49].

- **Real-time algorithms** which are intended for use in real-time environments, i.e., servicing disk requests within given deadlines. Examples here include *earliest deadline first (EDF)* [58], *SCAN-EDF* [78], and *priority SCAN (PSCAN)* [16].

- **Stream-oriented algorithms** handling retrieval of continuous data streams. Several proposed algorithms exist, e.g., the scheduler in the continuous media file system (CMFS) [3, 69], *grouped sweep scheduling (GSS)* [119], *BubbleUp* [19], *T-scan* [22], *batched SCAN (BSCAN)* [53], *buffer-inventory-based dynamic scheduling (BIDS)* [72], and *greedy-but-safe earliest-deadline-first (GS_EDF)* [108].

- **Mixed-media algorithms** which recognize that different disk requests may have different requirements with respect to service level. *Cello* [95, 96], *massively-parallel and real-time storage* (MARS) [13], *fair mixed-media scheduling (FAMISH)* [80], *deadline sensitive SCAN* [39], the disk scheduler in *Fellini* [62], *delta L* [11], and *adaptive disk scheduler for mixed-media workloads (APEX)* [60] are some examples.

In addition, some priority-based disk scheduling algorithms exist. Support for priorities is, however, orthogonal to the classification above, and we will therefore present those algorithms within the existing four classes. As the two last classes of algorithms are primarily designed for applications supporting multimedia data (and the first two may have several shortcomings), we look a bit closer into these.

With the introduction of continuous media like audio and video, new requirements for data delivery arose, i.e., pure performance-oriented and pure deadline-driven algorithms will fail during high workloads. Therefore, several *stream-oriented* disk scheduling algorithms have been proposed, i.e., schedulers that are primarily optimized for handling retrieval of continuous data streams. Compared to the real-time scheduling algorithms, these algorithms often focus less on deadlines, and instead rely on periodicity of the requests (the requests are typically served in fixed-length rounds) and fair allocation of disk bandwidth. These stream-oriented algorithms normally try to consider efficiency, and to a certain extent work-conservation. Furthermore, they typically offer a statistical real-time or throughput only guarantee, based on periodic requests, but with no support for request dropping, priorities, or deadlines. However, given that these algorithms are targeted at delay-sensitive data, the lack of support for deadlines is compensated by careful load control usually through admission control and by the fact that the load is uniform, e.g., all requests are for video data.

During the last years applications presenting both discrete and continuous media, and thus providing different service classes, have emerged. Disk scheduling for mixed-media workloads has therefore become an active research area. Common in most of these algorithms is that they have a hierarchical (typically two-level) design, where one level ensures efficient usage of the disks, while another level handles QoS and differentiation of service classes. Some of these algorithms rely on the proportional share allocation paradigm, while still offering QoS guarantees. This is possible since they use a fixed set of service classes (i.e., queues), and the weight of each queue is carefully controlled. Thereby, the relative share of the bandwidth for a queue becomes equal to the absolute share, and QoS guarantees can be given. The number of service classes increased from best effort only to two (discrete and continuous) and even to an arbitrary number of service classes supporting different services to be more flexible. Furthermore, some of these schedulers are priority-based, device idle time is often minimized by using work-conservation, and to avoid overload situations some include admission control.

As shown in the previous subsections, there are a lot of different disk scheduling algorithms suitable for different purposes and workload scenarios. The two last classes of algorithms, i.e., stream-oriented and mixed-media, are most appropriate for managing multimedia data. Stream-oriented algorithms are primarily targeted for periodic access for a continuous playout of an audio or video stream, whereas mixed-media aim for both discrete and continuous multimedia data requests at the same time. The stream-oriented algorithms mainly address playout of continuous media, i.e., requirements like high guaranteed throughput of a stream. However, as the requests often are served in rounds, low latency support is neglected. In some mixed media algorithms, separate high-level queues can be applied to support low latency interactions by inserting these requests into the head of the low-level scheduling queue.

There exists schedulers aimed for most purposes. However, new disk devices have a lot of "intelligence" [116], and one does not necessarily know exactly where a block is stored. The devices therefore often have a build-in hardware version of SCAN to optimize performance, and future disk scheduling research could maybe exploit these new capabilities.

4 Data Placement

Data placement policies aim to improve the efficiency of the storage system by proper placement of data elements on the storage device(s). In this section, we describe placement policies for both single and multiple devices. e.g., single device block allocation, striping, replication, and load balancing.

4.1 Block Allocation

As mentioned above, the seek overhead is usually the dominant factors in disk access time. Optimizing block placement according to disk mechanics and retrieval patterns to minimize average data retrieval time has been an active research area for a long time, e.g., [40], and several placement policies have been proposed:

- **Scattered (random):** data elements were initially placed on arbitrary disk blocks regardless of disk characteristics and access patterns, but this policy is often regarded as inappropriate due to possibly large numbers of intra-file seeks in multimedia streaming scenarios having I/O requests spanning several blocks. However, random placement policies are used for a mixed media, heterogeneous disk scenario [8, 54, 83]. It is shown that using random placement of replicas on random disk gives performance equally to conventional striping schemes (at the cost of replicating data) [84]. Additionally, under highly multiplexed workloads, the disk head will typically read only a single (large) disk block for the particular file. Thus, one approach to achieve disk efficiency is to use large blocks and make no particular effort to store them contiguously, but rather rely on replication and wide striping to handle bottlenecks due to highly multiplexed nodes [46].

- **Contiguous:** this policy tries to store all blocks of a file contiguously on disk, i.e., no intra-file seeks is necessary when reading the data sequentially. When reading a single multimedia file, contiguity of the data blocks is therefore considered advantageous. Thus, the contiguous placement policy is a nice solution for read-only (updates may require a rewrite of the whole file) and exclusive (accesses to different files give large seeks) device accesses. However, a multimedia server often must replace data due to popularity, and it usually has multiple concurrent clients. A *pure* contiguous block placement strategy may therefore not be desirable. Nevertheless, some variations of this exist. For example, blocks can be tried stored contiguously in the case of multimedia files whereas with other types of files no attempt is done to optimize the placement, i.e., the blocks are mixed without any overhead [76, 77]. An additional abstraction can be added to assist multimedia applications, as in the case of the data type dependent modules of Symphony [91]. Small, page-sized blocks that are stored contiguously allow an efficient mapping and unmapping of disk space into main memory, and allows finely granular decisions for buffering [37, 62].

- **Extent-based:** this is an adaption of the contiguous policy where an extent is a physically linear series of blocks and can therefore be read without repositioning the head [43]. The files are partitioned into multiple extents (giving better support for interleaved operations), and each field in the meta data structure (inode) usually points to a series of blocks (start offset and number of adjacent blocks) reducing number of inode lookups when reading. Several of file systems use this allocation policy, e.g., the XFS [98, 118], JFS [99, 103], and *Minorca* [113] file systems.

- **Cylinder-based:** the disk is organized in cylinder groups, and blocks are allocated close to each other to avoid long intra-file seeks (similar to extents) if possible, i.e., 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 and if possible at rotational optimal positions. To avoid fragmentation and different loaded cylinder groups, the data block allocation is shifted to a different cylinder group every few MB. This is for example used in the *Fast File System (FFS)* [64].

- **Log-structured:** a sequential, append-only log as is used as the only on-disk structure, i.e., all file system data is stored in a single, contiguous log. The idea relies on newer, larger buffer caches that will usually hold the required data blocks in memory when requested and assumes that allocation of new blocks is a bottleneck. Thus, to optimize writes, the next available block is allocated giving a minimum seek time when writing. This is for example used in the *log-structured file system (LFS)* [18, 63, 81, 89].

- **Zoned:** disk characteristics and average position of the disk arm is used to find a suitable block. For example, the *organ-pipe* placement policy tries to put the most frequently accessed blocks, regardless of which file it belongs to, close to the center of the disk. This would for example mean placing and periodically rearranging popular video clips into the center cylinders of the disk as proposed in several schemes [1, 32, 82, 111]. To compensate for the zoned disks, *skewed organ-pipe* policy, moves the most frequently accessed blocks slightly outward on the disk platter according to the capacity differences between the zones. This policy is used in [105]. Other zoned placement policy examples include the *near constant transfer time (NCTT)* [51] and *track-pairing* [7].

- **Constrained:** these policies tries to take advantage of sequential access patterns and thereby reduce seek overhead by restricting the average distance between consecutive blocks in number of cylinders [3]. For

example, *REBECA* [36] partitions the disk in regions and place consecutive multimedia objects in the next adjacent region to minimize seek time at the cost of start-up latency. The *strand-based* model [76,77] derives the storage granularity and scattering parameters by using device characteristics and playback rate, and data retrieved in an operation is stored contiguously (as a strand), and each strand is placed on disk according to a maximum calculated seek time between consecutive strands, i.e., a maximum scatter value, to guarantee continuous retrieval and data playout.

Block allocation mainly address disk efficiency and not latency, i.e., many data placement strategies have been proposed to reduce seek time and achieve a high disk bandwidth. Which one is best depends on access pattern, but there are some important general observations. In a server, multiple streams will be concurrently played out, and the order of the I/O operation, both with respect to current playout position and media object, is very hard to predict. Thus, schemes using strict assumptions about access patterns may be optimized for a certain sequence of I/O operations, but may fail if there are only small differences according to the assumptions. Additionally, the probability of reading a whole file continuously (as one long I/O operation) is very small, i.e., a pure continuous block allocation will give large seeks over whole files for each access t to a different file. What's important is to have as small seeks as possible per operation, i.e., data retrieved as one operation should be stored continuously or at least close. This favors techniques like extent-based (or cylinder based) placement[2], e.g., one extent per typical I/O operation. How large each extent should be and where each extent is placed on disk is another issue. For example, in a variable bit rate (VBR) scenario, conventional fixed-sized clusters correspond to varying amounts of time, depending on the achieved compression [5]. Alternatively, the system can store data in clusters that correspond to a fixed amount of time, with a variable cluster size, to better support round based retrieval. Constant retrieval time of each constant time, variable sized segment can for example be achieved using the different zones on a disk, e.g., like NCTT [51]. Additionally, compressed data might not correspond to an even number of disk sectors, which introduces the problem of packing data [34].

4.2 Journaling

Journaling provides fast crash recovery by maintaining a journal of write operations [18, 43, 100, 103]. All write operations are logged in such journals in the order in which they were performed. In case of a system reboot after a crash, all write operations that are reported in the journal are validated, the ones that are missing are invalidated, which leaves the file system in a consistent state. The amount of logging data for one write operation is independent of the number of blocks written in the operation, so the overhead for writing large amounts of data at once is much lower than that for frequent small writes. Usually, logging information is kept in memory until a block-size amount has been collected, and then flushed to disk. Thus, because writing of the journal and writing the actual data is interleaved, and keeping the journal on the same disk increases the seek time. Writing the journal to a separate disk maintains the write performance of an equivalent unjournalized file system. Although this variation increases the failure probability of the entire system by adding a dependency on another disk, a regular file system check is still possible in this case. Due to the faster file system checking after crashes and the low overhead for large files, journaling is a feature that is also found in multimedia file systems.

Journaling applies to a general requirement of service availability as it reduces restart times. Which type of journals that is best suited depends how it places the journal information as it should preferably not affect the performance of the application I/O operations.

4.3 Multiple Disks Issues

The bandwidth capacity of a single disk would also strongly limit the number of concurrent users in a multimedia scenario. One approach approach to overcome the bandwidth bottleneck is to scatter different segments of a file across multiple disks using some kind of *data striping* or *data interleaving* [30, 33]. Another approach is to use *replication* to distribute several copies of a file to different disks.

4.3.1 Striping and Interleaving

For many years, *striping* [59, 74] (also called *wide* or *full striping*) has been the primary technique when reading multimedia data from disks where data blocks are spread over all devices in the storage system and all devices are accessed in parallel during an I/O operation increasing the effective transfer rate. The *redundant arrays of*

[2] Or in some cases, even a random placement with a large block size might be sufficient.

inexpensive disks (RAID) technology [20,74] has addressed both performance and reliability issues. The *streaming RAID* [109] uses a grouping approach where where stripes are grouped in consecutive segments to achieve good performance. Furthermore, in the case of multimedia server environments, block-interleaved schemes like RAID-5 are more cost effective compared to bit-interleaved schemes like RAID-3 [71].

However, disk performance has improved considerably, and new compression algorithms for multimedia data have to some extent reduced bandwidth requirements[3]. Therefore, several refinements to the simple striping scheme have been proposed, which serve a request without necessarily involving all disks in the array. Data *interleaving* [33] (also called *compound striping* [25]) is a technique where the media files are stored across a set of disks, and the simplest form store each successive block is on successive disks in a round-robin manner. Other approaches to interleaving are *staggered striping* [6], where data is striped over possible overlapping disk groups, and *scalable stream pumping* [117], where all data blocks are stored on successive disks and successive zones.

Two important parameters when implementing a disk array for a multimedia storage server are the stripe unit size and the degree of striping for optimal resource utilization. In [110], experiments using a fixed- and variable-sized block placement policy are performed to determine optimal stripe unit size. To achieve maximum utilization, [94] proposes a scheme where the array is partitioned and data is striped across single-disk partitions. Results presented in [35] suggest similar policies, because one-disk stripe groups have better performance with respect to throughput and wasted bandwidth. Thus, using small stripes (as long as playout rate can be serviced) is more efficient with respect to total storage system I/O bandwidth, because only one or a few disks have to pay the overhead of moving the disk head for a particular request and concurrent requests can therefore be served in parallel.

4.3.2 Replication

A performance increase of the servers themselves beyond single file retrieval optimization can be achieved by replication, i.e., to guarantee availability and to overcome limits in the number of concurrent accesses to individual titles. Several schemes are proposed:

- **Static:** Content files are duplicated explicitly, by storing the file on multiple machines and providing the user with a choice of access points. This is frequently done in the Internet today. Automatic replication of the relatively small and frequently accessed read-only system files for load balancing among file servers has been proposed in [85], and a static placement policy that uses estimated load information for the placement of video objects is proposed in [25]. This static placement policy is complementary to the proposed replication, as it reduces, but cannot eliminate, dynamic imbalances.
- **Dynamic segment:** Read-only, equally-sized segment of a movie can be dynamically replicated according to the number prespecified threshold for the number of concurrent read requests [23].
- **Threshold-based dynamic:** Whole continuous media files are replicated, and all disks of the system and the probability of new requests is taken into account to determine whether a movie should be replicated or a replica should be deleted [57]. A number of variations is proposed, e.g., *injected sequential*, *piggybacked sequential*, *injected parallel*, *piggybacked parallel* and *piggybacked and injected parallel* replication.
- **Partial:** This is based on the observation that if there were a number of consecutive requests for the same video, and if the blocks read in by the first request were copied to another disk, it would be possible to switch the following requests to the partial replica just created [25]. This technique was introduced for load balancing in multimedia systems.
- **Random:** Replicas are created and deleted randomly, and in its basic application, differences in access frequency to the different files is not taken into account. This policy has for a long time been viewed upon as inappropriate, but as user behavior is hard to predict several studies have tried random placement and replication, e.g., [8, 54, 83].

Replication address requirements like availability and latency, and can be an important means to increase performance if used appropriately. As more copies of a data element exists on several devices, the systems is more reliable to failures, and long waiting times for an overloaded device can be avoided by accessing another replica. Which mechanism to chose might depend on several factors like frequency of popularity shift, size of files or objects, etc. Thus, the applied replication mechanism should take these parameters into account, because it is expensive to replicate and copy multimedia data elements.

[3] At least compared to older video steams with the same quality. Though, the number of streams have increased making the total bandwidth requirement higher.

4.3.3 Load Balancing

Using data interleaving where only a few disks are accessed for each I/O operation may cause load imbalance as all concurrent requests may come to the same group of disks where others are left unused. Replication (described above) is one way to deal with load imbalance resulting in low overall performance and high latencies, but several other approaches are proposed. For example, the problem assigning media streams to disks to achieve a balanced disk array load is addressed in [25, 94, 114]. To determine imbalance across partitions, [94] presents a model to determine which partition sizes that best utilize the resources. The *DASD dancing* load balancing policy [114, 115] determines if the most frequently accessed files can be played out from memory, how to best assign and replicate them to striped disk groups, and to shift existing streams to another replica stored on another disk group if a disk group becomes overloaded. The *generalized staggered distributed data layout (G-SDCL)* data layout policy [14] tries to avoid hot spots in the disk array (or in the pool of storage nodes) supporting arbitrary playout modes in various speeds. Data is interleaved using round-robin, but for each round, the cyclic layout is staggered, i.e., the first data segment in each round is not stored on the same disk. The *prime round-robin (PRR)* placement scheme [56] is similar to G-SDCL, and both these schemes try to avoid load imbalance by introducing a rounding distance based on a (prime) number to distribute disk accesses evenly at any retrieval speed.

5 Buffer Management

In multimedia applications, due to the high data consumption rate data is often replaced before it might be reused [12]. Thus, using a complex, computational expensive *caching* or *page replacement* algorithm might be wasted, and a traditional algorithm as described in [28, 104] might be applied. Nevertheless, in some multimedia applications where data often might be reused, proper replacement algorithms may increase performance, i.e., increasing system throughput and latency of new requests, and in multimedia scenarios, we classify the algorithms into two classes:

- **Block-based:** block-based caching and replacement algorithms is usually implemented in todays systems using a variant of *least recently used (LRU)*, where each block is evaluated independently when a block is to be replaced. In the context of multimedia systems, algorithms like *least/most relevant for presentation (L/MRP)* [66] are proposed where parameters like presentation mode and presentation point are considered. Other variations of L/MRP include *Q-L/MRP* [44] and *MPEG-L/MRP* [9].

- **Stream-based:** this policy tries to minimize disk accesses based on the observation that if there were a number of consecutive requests for the same video they would issue the same requests in a short period of time. Caching is then performed by keeping data for a stream which follows temporarily close to another stream of the same object in memory. Some examples are the *(generalized) interval caching* [24, 25], *distance* [70] and *SHR* [50].

Even though data might be replaced before it can be reused due to high data rates, one should try to gain benefit from caching in order to reduce disk operations. On the server side, a complex, block-based algorithm will often be too CPU intensive [44], but stream-oriented algorithms making cache decisions based on the distance between clients might be appropriate. Additionally, as the number of heterogeneous devices with different capabilities increases, support for strided access as a generalization of the stream-oriented approach can be an interesting research direction. This is for example useful in systems having multi-protocol support and scalable content, e.g., layer dropping in *Priority Progress Streaming* [55], .

Another aspect of buffer management is when data is retrieved form the storage devices. Usually, demand paging will be inappropriate in high data rate, low latency applications like multimedia streaming systems. Therefore, *prefetching* data from disk to memory is better suited to support continuous play-back of time-dependent data types. Prefetching is a mechanism to preload data from slow, high-latency storage devices such as disks to fast, low-latency storage like main memory. This reduces the response time of a data read request dramatically and increases the disk I/O bandwidth as more data is usually retrieved in one operation. Obviously, knowledge (or estimations) about application behavior might be used for both replacement and prefetching, and many of the existing file systems optimize disk accesses by using a read-ahead mechanism if sequential reads are performed. In the case of multimedia presentations, a prefetching mechanism can very often take advantage of the sequential access pattern and several mechanisms are proposed. For example, L/MRP [66] calculates the relevance values to maximum (one) for a given interval in front of the current playout position, and all data units which have a relevance value of one are prefetched into memory. A similar read-ahead mechanism is presented in [2] retrieving

data before it is requested if the system determines that the accesses are sequential. In [106], assuming a linear playout of the continuous data stream, the data needed in the next period (determined by a trade-off between the maximum concurrent streams and the initial delay) is prefetched into a shared buffer. Additionally, models for preloading data according to the loading and consuming rate and the available amount of buffers are described in [77, 120]. How to prefetch data in a multimedia scenario, is of course again dependent of access patterns. Thus, a simple read-ahead-like mechanism can often be sufficient in a pure playout application like NoD, VoD, or LoD. The amount of data data to prefetch should be a trade-off between data rate, retrieval rate and available buffers.

6 File System Meta-Data Structures

File systems use a data structure to hold meta-data, like file name, size, owner, permissions, etc., and data pointers to storage blocks for a file. Traditionally, UNIX-like systems have used i-nodes (having direct, indirect, double indirect, and triple indirect block pointers) and Windows systems used FAT (having an uni-directional linked list) [104]. However, these systems need many meta-data structure lookups when reading a file, i.e., one per block. Systems supporting extent-based allocation have also modified the metadata structure. Instead of having a pointer to each block, the meta-data structure points to the extent, e.g., having a pointer to the first block and a counter holding the length of the extent [43] like in XFS [118] and Minorca [113]. In Windows NTFS [100], the master file table has a record for each file where block runs are defined in the same way as extents, i.e., a start address and the number of adjacent blocks. These meta-data structures reduce the number of lookups, the chains of data structures to track allocated blocks, and reduce the size of the file system meta-data. In other words, the request latency and the required number of disk operations reading meta-data are reduced.

Additionally, a many file systems provide space for meta-data for file specific tasks. The following three approaches to an implementation and use of this meta-data can be found:

- **Application meta data:** application-relevant-only information which comprises typically information about the type of format, data rates, extended access rights, or information relevant for copyright questions. While multimedia file systems have implemented this for themselves, some standard file systems support this as well [4, 100].
- **Operational meta data:** meta-data to change the way in which API calls, and standard API calls in particular, are processed. It is similar to application meta data, but kernel modules interpret the meta-data as well and behave in a desired fashion. In Symphony [91], data type information is used to select an appropriate module, which among other things change the interpretation of block sizes. Tiger Shark [46] retrieves the default data rate of a file from its meta data and uses this information for data prefetching.
- **Integral meta data:** information integrated into the structures of the file system itself, i.e., the equivalent of the Unix VFS inode and block structures. Since timeliness of data access is the main concern of multimedia file systems, using these structures to provide application with a straight-forward access to temporal units according to its understanding is typical for this approach [76, 91]. Furthermore, the Video File Server [76] stores information for the synchronization of sub-streams in integral structures.

7 Putting It All Together

As one can see from the above section, a lot of work exists in the area of multimedia storage systems, and different solutions have been proposed for various workloads and applications. Many of the techniques presented have been developed independently, but the overall performance of a storage system can only be evaluated when the appropriate set of techniques is integrated into a complete storage system. We consider the integration of these techniques, to increase performance and flexibility for a mixed interactive workload, the main future research challenge. Storage systems that integrate several subcomponents have been developed in the past. The term chosen for such integrated systems has been inconsistent (e.g., file system, file server, storage server), but applications access all of these systems through file system abstractions. The file systems that are used for multimedia applications can be divided into three groups:

- **General file systems** such as *FAT* [100], *NTFS* [100], *Ext2* [15, 97], and *FFS* [64] are not designed for a specific application area, but are meant to support all applications. Since this generality comprises multimedia applications, the performance of general file systems provides a benchmark for specific multimedia file systems.

- **Multimedia file systems** try to address the requirements described in Section 2. Examples for such file systems are the *Video File Server* [76], *Shark* [45], *Everest* [37], *CMFS* [61], *Fellini* [62], *Symphony* [91], *Tiger Shark* [46], *Minorca* [113], *ERTFS* [48], *PMFS* [73], and *MiPFS* [17]. The main distinguishing factor of multimedia applications is thus that they have soft real-time constraints. In the design of many file systems, this demand has been addressed by focusing on scheduling. Systems have been designed exclusively for streaming [37,45,46,48,61,76], for a combination with a second non real-time class to accommodate mixed workloads [17,62], and for serving several application classes [60,91].

- **High performance file systems** such as *GPFS* [86], *CXFS* [98], *Frangipani* [107], *GFS* [75], *PPFS* [47], *Exemplar* [10], and *ELFS* [52] are primarily designed for applications that require reading and writing of large amounts of data in a very short time. Typical examples are experiments in physics and large scale simulations. In contrast to multimedia file systems, timeliness needs not be guaranteed, but overall performance must be maximized. Newly developed high performance file systems concentrate on this task, while high performance file systems that are derived from multimedia file systems [86,98] have gained scalability while continuing to support resource reservation. However, because of their high performance for large amounts of data, even those newly developed file systems can be used for multimedia applications and compete with multimedia file systems.

As multimedia applications become more interactive and include write operations, we should consider high performance file system features. For example, matrix operations is a feature that is not frequently required in multimedia applications. However, it will be useful there and important in I/O intensive access pattern found in high performance applications. It leads to strided reading and writing or other non-contiguous reading and writing that follows well-known patterns. Many high performance file systems support this ability by providing a separate API [52], or also by detection of patterns [10,86,98].

A comprehensive approach to modernize multimedia file systems is also supported by Shenoy's analysis of new requirements, which include the need for integrated file systems that support a variety of applications [90]. Furthermore, he considers server-independent, self-healing, self-managing networked file systems a new goal, and predicts that more functions will be off-loaded to processors on the disk. Although storage area network (SAN) and network attached storage (NAS) products are already claiming such abilities, Haskin has pointed to limitations in current systems that led to the integrated approach followed by GPFS [86].

Some previous work focuses on one single part of the storage system only, and one large challenge is therefore to integrate suitable subcomponents into one "optimal" system for a particular workload. Typically, the newer of these storage systems address a larger breadth of workloads and take more subcomponents into account, whereas the older ones are more restricted. The integration of different mechanisms is, however, not a trivial task, because, as mentioned earlier, the decisions made for each subcomponent are often dependent on the expected access patterns, but one must also consider the properties of the mechanisms chosen for the other components. Thus, one must also make all the appropriate choices for each component or mechanism work well together. For example, the data placement on disk and disk scheduling are tightly coupled as the disk arm movement is dependent on where each data block is located on the platter. As many scheduling algorithms give shortest access time to data blocks near the center of the disk, e.g., (skewed) organ pipe placement, the most popular data could be placed according to this property of the scheduling algorithms. On the other hand, if one also applies a buffer management algorithm that tries to keep the most frequently accessed data in memory, e.g., generalized interval caching [25], L/MRP [66] etc., such a data placement strategy will in many cases be wasted, as the requested data probably resides in memory, and the disk is not accessed. Furthermore, such block reorganizations according to popularity is expensive and should be minimized in many cases. However, dynamic block reorganizations can still be useful. For example, scalability can be increased by dynamic reorganization at block granularity spanning several disks combined with replication. Overload of a single disk by frequent access to individual blocks can be prevented, and in future interactive applications that include write operations, the problem of partial file locking can be reduced.

Even though some mechanisms have contradicting properties, several mechanisms can be combined in several ways to give good storage system performance. One interesting example in our context would be to use a hierarchical mixed media scheduler where different service classes can be supported, work-conservation removes device idle time, and disk efficiency is improved sorting all the requests in a round according to the placement on the platters. As video or audio streams can benefit greatly from sequential prefetching (read-ahead), some kind of adjacent block placement should be used to minimize intra-request seeks. A pure continuous placement results in large "file-sized" seeks as several files are retrieved in parallel, i.e., a type of extent-based placement could be appropriate where the extent size should be large enough to avoid intra-request seeks, but small enough to allow efficient inter-file seeks.

If possible, one should additionally try to gain benefit from caching. However, the data rates in multimedia scenarios are high, and the benefits might be small as data might be replaced before it can be reused. A complex, block-based, CPU intensive algorithm like L/MRP should therefore not be used in a server, but stream-oriented algorithms making cache decisions based on the distance between clients might be appropriate.

With respect to index structures, the number of accesses to find the address of the requested data blocks can be reduced using some kind of "extent info" holding pointers to several continuous blocks in one entry in the structure, e.g., location of first block in the extent and a length of the extent [103, 113]. Very application-specific index structures [76], on the other hand, are hard to adapt to new application requirements and should be avoided for future integrated file systems.

To specify multimedia specific information to the file system, applications access the storage system through the system call interface to the operating system. This can be solved either by using an entirely *proprietary API* [45, 61, 62, 73, 76] or by extending the operating system's standard file system API [46, 86, 91, 98, 103, 113]. However, experience shows that multimedia file systems can be most easily commoditized when they use (extended) standard APIs as most applications is implemented on commodity operating systems, and thus simplify the extension of existing applications.

In the case of a multimedia server, multiple disks usually have to be used to store all data, increase reliability and fault tolerance, and increase performance. The disks today have each sufficient data rates to support a number of streams, i.e., striping units should be made as small as possible (as long as playout rates are achieved) due to overall disk bandwidth utilization. Thus, full striping is no longer needed to achieve high enough bandwidths, but can be used like staggered striping (together with replication) to distribute the workload on several devices.

8 Summary and Research Directions

Time-dependent multimedia data types, like audio and video, will be a natural part of future applications and integrated together with time-independent data types, like text, graphics, and images. Commodity operating systems were originally designed for best-effort applications, and their storage systems do not presently support all the requirements of multimedia systems. This paper gives a short overview over multimedia related storage system research. Proposed approaches to increase multimedia support include work in:

- **Disk scheduling** algorithms sorting requests by considering overall performance (efficiency), time dependent data (deadlines) and several service classes.
- **Data placement** policies using proper placement of data elements on the storage devices to improved performance. Issues here include block allocation, various striping techniques, replication and load balancing.
- **Buffer management** trying to use the characteristic access patterns to perform efficient replacement, caching and prefetching.
- **Meta data** management where additional multimedia specific information is added, and the data block index structure is changed to reduce the number of index structure look-ups when reading the file.
- **File systems** often integrating some of the mechanisms above, but also extending the API for multimedia specific operations.

It is not clear how future systems will support multimedia applications in a better way, but an emerging trend is to have systems supporting multiple application classes with heterogeneous performance requirements rather than pure multimedia systems. One might add such support in a middleware layer at the cost of some run-time overhead and reduced application isolation [93]. Experiments in [92] also show that an integrated server in most cases performs better compared to using an integration layer on a partitioned server. In such a file system, the implementation process will be more complex, but such an integrated system will be easier to administrate and ease the integration of new service classes.

As the number of application classes increase, we believe that a storage system must be able to manage various different loads and heterogeneous requirements. To be able to fully support different service classes, all components must have appropriate designs and implementations. Thus, research in the area of storage systems should focus on systems that can automatically and without user intervention adapt to particular workloads by adapting the data layout, the scheduling strategy and buffer management approaches. Finally, a large amount of existing work propose different mechanisms and policies, and work remain to be performed in the integration of suitable mechanisms and see closer how they perform together.

References

[1] S. Akyurek and K. Salem. Adaptive block rearrangement. *ACM Transactions on Computer Systems (TOCS)*, 13(2):89–121, May 1995.

[2] D.C. Anderson, J.S. Chase, S. Gadde, A.J. Gallatin, K.G. Yocum, and M.J. Feeley. Cheating the i/o bottleneck: Network storage with trapeze/myrinet. In *USENIX Annual Technical Conference*, New Orleans, LA, USA, June 1998.

[3] R. Anderson, Y. Osawa, and R. Govindan. A file system for continuous media. *ACM Transactions on Computer Systems*, 10(4):311–337, 1992.

[4] Apple Computer Inc. *HFS Plus Volume Format*, February 2000. Technical Note TN1150.

[5] T.C. Bell, A. Moffat, I.H. Witten, and J. Zobel. The mg retrieval system: Compressing for space and speed. *Communications of the ACM*, 38(4):41–42, April 1995.

[6] Steven Berson, Shahram Ghandeharizadeh, Richard R. Muntz, and Xiangyu Ju. Staggered striping in multimedia information systems. In *ACM Int. Conf. on Management of Data (SIGMOD'94)*, pages 70–90, Minneapolis, MN, USA, May 1994.

[7] Yitzhak Birk. Track-pairing: A novel data layout for vod servers with multi-zone-recording disks. In *Proceedings of the International Conference on Multimedia Computing and Systems (ICMCS)*, pages 248–255, Washington, D.C., USA, May 1995.

[8] Yitzhak Birk. Random raids with selective exploitation of redundancy for high performance video servers. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 13–23, St. Louis, MO, USA, May 1997.

[9] Susanne Boll, Christian Heinlein, Wolfgang Klas, and Jochen Wandel. Mpeg-l/mrp: Adaptive streaming of mpeg videos for interactive internet. In *Proceedings of the International Workshop on Multimedia Information Systems (MIS)*, pages 104–113, Chicago, IL, USA, October 2000.

[10] Rajesh Bordawekar, Steven Landherr, Don Capps, and Mark Davis. Experimental evaluation of the hewlett-packard exemplar file system. *ACM Performance Evaluation Review*, 25(3):21–28, December 1997.

[11] P. Bosch and S.J. Mullender. Real-time disk scheduling in a mixed-media file system. In *6th IEEE Real Time Technology and Applications Symposium (RTAS 2000)*, pages 23–33, Washington D.C., USA, May 2000.

[12] Milind M. Buddhikot. *Project MARS: Scalable, High Performance, Web Based Multimedia-on-Demand (MOD) Services and Servers*. PhD thesis, Department of Computer Science, Washington University, St. Louis, MO, USA, August 1998.

[13] Milind M. Buddhikot, Xin Jane Chen, Dakang Wu, and Guru M. Parulkar. Enhancements to 4.4bsd unix for efficient networked multimedia in project mars. In *Proceedings of the International Conference on Multimedia Computing and Systems (ICMCS)*, pages 326–337, Austin, TX, USA, June 1998.

[14] Milind M. Buddhikot and Gurudatta M. Parulkar. Efficient data layout, scheduling amd playout control in MARS. *ACM/Springer Multimedia Systems*, 5(3):199–212, 1997.

[15] R. Card, T. Ts'o, and S. Tweedie. Design and implementation of the second extended filesystem. In *1st Dutch Int. Symp. on Linux*, 1994.

[16] M.J. Carey, R. Jauhari, and M. Livny. Priority in dbms resource scheduling. In *Proceedings of the Internetional Conference on Very Large Databases (VLDB)*, pages 397–410, Amsterdam, The Netherlands, 1989.

[17] Jesus Carretero, Weiyu Zhu, Xiaohui Shen, and Alok Choudhary. MiPFS: Multimedia integrated parallel file system. Technical Report CPDC-TR-9810-021, Center for Parallel and Distributed Computing, Northwestern University, 1998.

[18] Albert Chang, Mark F. Mergen, Robert K. Rader, Jeffrey A. Roberts, and Scott L. Porter. Evolution of storage facilities in AIX version 3 for RISC system/6000 processors. *IBM Journal of Research and Development*, 34(1):105–110, 1990.

[19] E. Chang and H. Garcia-Molina. Bubbleup: Low latency fast-scan for media servers. In *Proceedings of the ACM International Multimedia Conference (ACM MM)*, Seattle, WA, USA, November 1997.

[20] P.M. Chen, E.K. Lee, G.A. Gibson, Randy H. Katz, and «««< all.bib D.A. Patterson. Raid: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, June 1994.

[21] Intel Corporation. Microprocessor quick reference guide. http://www.intel.com/pressroom/kits/-quickrefyr.htm.

[22] S.J. Daigle and J.K. Strosnider. Disk scheduling for multimedia data streams. In *Conf. on High-Speed Networking and Multimedia Computing*, pages 212–223, San Jose, CA, USA, February 1994.

[23] Asit Dan, Martin Kienzle, and Dinkar Sitaram. A dynamic policy of segment replication for load-balancing in video-on-demand servers. *ACM/Springer Multimedia Systems*, 3(3):93–103, 1995.

[24] Asit Dan and Dinkar Sitaram. A generalized interval caching policy for mixed interactive and long video workloads. Technical Report RC 20206 (89404), IBM Research Division, September 1995.

[25] Asit Dan and Dinkar Sitaram. An online video placement policy based on the bandwidth to space ratio (bsr). In *ACM SIGMOD Int. Conf. on Management of Data*, pages 376–385, San Jose, CA, USA, May 1995.

[26] Ali E. Dashti, Seon Ho Kim, Cyrus Shahabi, and Roger Zimmermann. *Streaming Media Server Design*. Prentice Hall, 2003. ISBN 013067038-3.

[27] P.J. Denning. Effects of scheduling on file memory operations. In *AFIPS Spring Joint Computer Conference*, pages 9–21, April 1967.

[28] Wolfgang Effelsberg and T. Härder. Principles of database buffer management. *ACM Transactions on Database Systems*, 9(4):560–595, December 1984.

[29] Wu-Chi Feng, Jin Choi, Wu chang Feng, and Jonathan Walpole. Under the plastic: A quantitative look at dvd video encoding and its impact on video modeling.

[30] A. Garcia-Martinez, J. Fernadez-Conde, and Á. Viña. Efficient memory management in vod servers. *Computer Communications Journal*, 23(3):253–266, February 2000.

[31] R. Geist and S. Daniel. A continuum of disk scheduling algorithms. *ACM Transactions on Computer Systems*, 5(1):77–92, 1987.

[32] R. Geist, D. Suggs, R. Reynolds, S. Divatia, F. Harris, E. Foster, and P. Kolte. Disk performance enhancement through markov-based cylinder remapping. In *30th ACM Annual Southeast Regional Conf.*, pages 23–28, Raleigh, NC, USA, 1992.

[33] D. James Gemmell, Harrick M. Vin, Dilip D. Kandlur, P. Venkat Rangan, and Larry A. Rowe. Multimedia storage servers: A tutorial. *IEEE Computer*, 28(5):40–49, 1995.

[34] J. Gemmell and Stavros Christodoulakis. Principles of delay-sensitive multimedia data storage retrieval. *ACM Transactions on Information Systems*, 10(1):51–90, January 1992.

[35] Shahram Ghandeharizadeh and Seon Ho Kim. Striping in multi-disk video servers. In *SPIE High-Density Data Recording and Retrieval Technologies Conference*. October 1995.

[36] Shahram Ghandeharizadeh, Seon Ho Kim, and Cyrus Shahabi. On configuring a single disk continuous media server. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 37–46, Ottawa, Ontario, Canada, 1995.

[37] Shahram Ghandeharizadeh, Roger Zimmermann, Weifeng Shi, Reza Rejaie, Doug Ierardi, and Ta-Wei Li. Mitra: A scalable continuous media server. *Kluwer Multimedia Tools and Applications*, 5(1):79–108, 1997.

[38] G.A. Gibson, J.S. Vitter, and John Wilkes. Strategic directions in storage i/o issues in large-scale computing. *ACM Computing Surveys (CSUR)*, 28(4):779–793, December 1996.

[39] K. Gopalan. Real-time disk scheduling using deadline sensitive scan. Technical Report TR-92, Experimental Computer Systems Labs, Dept. of Computer Science, State University of New York, Stony Brook, 2001.

[40] D.D. Grossman and Silverman H.F. Placement of records on a secondary storage device to minimize access time. *Journal of the ACM (JACM)*, 20(3):429–438, July 1973.

[41] The PC Technology Guide. Components - processors. http://www.pctechguide.com/02procs.htm.

[42] The PC Technology Guide. Storage - hard disks. http://www.pctechguide.com/04disks.htm.

[43] W.v. Hagen. *Linux Filesystems*. Sams publishing, 2002.

[44] Pål Halvorsen, Vera Goebel, and Thomas Plagemann. Q-l/mrp: A buffer management mechanism for qos support in a multimedia dbms. In *Proceedings of the IEEE International Workshop on Multimedia Database Management Systems (IW-MMDBMS)*, pages 162–171, Dayton, OH, USA, August 1998.

[45] Roger Haskin. The shark continuous media file server. In *Proceedings of the Computer Society International Conference (COMPCON)*, pages 12–17, San Francisco, CA, USA, February 1993. IEEE Press.

[46] Roger Haskin. Tiger Shark — A scalable file system for multimedia. *IBM Journal of Research and Development*, 42(2):185–197, 1998.

[47] Jay Huber, Christopher L. Elford, Daniel A. Reed, Andrew A. Chien, and David S. Blumenthal. PPFS: A high performance portable paralle file system. In *Proceedings of the ACM Conference on Supercomputing*, pages 385–394, Barcelona, Spain, July 1995.

[48] EBSnet Inc. ERTFS Pro 4.4: File system software for embedded applications. http://www.etcbin.com/-ertfs.htm, May 2003.

[49] D.M. Jacobson and John Wilkes. Disk scheduling algorithms based on rotational position. Technical Report HPL-CSP-91-7, Hewlett Packard Labs, CA, USA, February 1991.

[50] M. Kamath, K. Ramamritham, and Don Towsley. Continuous media sharing in multimedia database systems. In *4th Int. Conf. on Database Systems for Advanced Applications (DASFAA'95)*, pages 79–86, Singapore, April 1995.

[51] J. Kang and H.Y. Yeom. Placement of vbr video data on mzr disks. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Baskin Ridge, NJ, USA, June 1999.

[52] John F. Karpovich, Andrew S. Grimshaw, and James C. French. Extensible file systems (ELFS): An object-oriented approach to high performance I/O. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications*, pages 191–204, Portland, OR, USA, October 1994.

[53] Deepak R. Kenchammana-Hosekote and Jaideep Srivastava. I/o scheduling for digital continuous media. *ACM/Springer Multimedia Systems*, 5(4):213–237, 1997.

[54] Jan Korst. Random duplicated assignment: an alternative to striping in video servers. In *Proceedings of the ACM International Multimedia Conference (ACM MM)*, pages 219–226, Seattle, WA, USA, 1997.

[55] Buck Krasic and Jonathan Walpole. Priority-progress streaming for quality-adaptive multimedia. In *Proceedings of the ACM Multimedia Doctoral Symposium*, Ottawa, Canada, October 2001.

[56] Taeck-Geun Kwon, Yanghee Choi, and Sukho Lee. Disk placement for arbitrary-rate playback in an interactive video server. *ACM/Springer Multimedia Systems*, 5(4):271–281, 1997.

[57] P.W.K. Lie, J.C.S. Lui, and L. Golubchik. Threshold-based dynamic replication in large-scale video-on-demand systems. In *Int. Workshop on Research Issues in Data Engineering (RIDE)*, pages 52–59, Orlando, FL, USA, February 1998.

[58] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in hard-real-time environment. *Journal of the ACM*, 20(1):47–61, 1973.

[59] M. Livny, S. Khoshafian, and H. Boran. Multi-disk management algorithms. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 69–77, Banff, Alberta, Canada, May 1997.

[60] Ketil Lund. *Adaptive Disk Scheduling for Multimedia Database Systems*. PhD thesis, University of Oslo, Norway, February 2003. (ongoing).

[61] Dwight J. Makaroff, Gerald W. Neufeld, and Norman C. Hutchinson. Design and implementation of a VBR continuous media file server. *Software Engineering*, 27(1):13–28, 2001.

[62] Cliff Martin, P.S. Narayanan, Banu Özden, Rajeev Rastogi, and Avi Silberschatz. *Multimedia Information Storage and Management, S.M. Chung (Ed.)*, chapter The Fellini Multimedia Storage Server. Kluwer Academic Publishers, 1996.

[63] Marshall K. McKusick, Keith Bostic, Michael J. Karels, and John S. Quarterman. *The Design and Implementation of the 4.4 BSD Operating System*. Addison Wesley, 1996.

[64] Marshall K. McKusick, William N. Joy, Samuel J. Leffler, and Robert S. Fabry. A fast file system for unix. *Computer Systems*, 2(3):181–197, 1984.

[65] A.G. Merten. Some quantitative techniques for file organization. Technical Report 15, University of Wisconsin Computing Center, Wisconsin, USA, 1970.

[66] Frank Moser, Achim Kraiss, and Wolfgang Klas. L/mrp: A buffer management strategy for interactive continuous data flows in a multimedia dbms. In *Proceedings of the Internetional Conference on Very Large Databases (VLDB)*, pages 275–286, Zurich, Switzerland, 1995.

[67] MPEG.org. Dvd technical notes - bitstream breakdown. http://mpeg.org/MPEG/DVD, March 2003.

[68] Klara Nahrstedt and Ralf Steinmetz. Resource management in multimedia networked systems. *IEEE Computer*, 28(5):52–63, 1995.

[69] Gerald W. Neufeld, Dwight Makaroff, and Norman C. Hutchinson. Design of a variable bit rate continuous media file server for an atm network. In *Proceedings of IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN)*, pages 370–380, San Jose, CA, USA, January 1996.

[70] Banu Özden, Rajeev Rastogi, and Avi Silberschatz. Buffer replacement algorithms for multimedia storage systems. In *Proceedings of the International Conference on Multimedia Computing and Systems (ICMCS)*, pages 172–180, Hiroshima, Japan, June 1996.

[71] Banu Özden, Rajeev Rastogi, and Avi Silberschatz. Disk striping in video server environments. In *Proceedings of the International Conference on Multimedia Computing and Systems (ICMCS)*, pages 580–589, Hiroshima, Japan, June 1996.

[72] Huanxu Pan, Lek Heng Ngoh, and Aurel A. Lazar. A buffer-inventory-based dynamic scheduling algorithm for multimedia-on-demand servers. *ACM/Springer Multimedia Systems*, 6(2):125–136, 1998.

[73] Seung-Ho Park, Si-Yong Park, Gwang Moon Kim, and Ki Dong Chung. Design and implementation of the parallel multimedia file system based on message distribution. In *Proceedings of the ACM International Multimedia Conference (ACM MM)*, pages 422–425, Marina del Rey, CA, USA, 2000.

[74] D.A. Patterson, G. Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). In *ACM Conf. on Management of Data (SIGMOD)*, pages 109–116, Chicago, IL, USA, June 1998.

[75] Kenneth W. Preslan, Andrew P. Barry, Jonathan Brassow, Russell Cattelan, Adam Manthei, Erling Nygaard, Seth Van Oort, David Teigland, Mike Tilstra, Matthew O'Keefe, Grant Erickson, and Manish Agarwal. Implementing journaling in a linux shared disk file system. In *IEEE/NASA Symposium on Mass Storage Systems*, pages 351–378, March 2000.

[76] P. Venkat Rangan and Harrick M. Vin. Designing file systems for digital video and audio. In *Proceedings of the ACM Symposium of Operating Systems Principles (SOSP)*, pages 81–94, Pacific Grove, CA, USA, 1991.

[77] P. Venkat Rangan and Harrick M. Vin. Efficient storage techniques for digital continuous multimedia. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):564–573, August 1993.

[78] A.L. Narasimha Reddy and James C. Wyllie. Disk scheduling in a multimedia i/o system. In *Proceedings of the ACM International Multimedia Conference (ACM MM)*, pages 225–233, Anaheim, CA, USA, August 1993.

[79] A.L. Narasimha Reddy and James C. Wyllie. I/o issues in a multimedia system. *IEEE Computer*, 27(3):69–74, 1994.

[80] Y. Rompogiannakis, Guido Nerjes, Peter Muth, Michael Paterakis, Peter Triantafillou, and Gerhard Weikum. Disk scheduling for mixed-media workloads in a multimedia server. In *Proceedings of the ACM International Multimedia Conference (ACM MM)*, pages 297–302, Bristol, UK, September 1998.

[81] Mendel Rosenblum and John K. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, 10(1):26–52, 1992.

[82] C. Ruemmler and John Wilkes. Disk shuffling. Technical Report HPL-91-156, HP Labs, October 1991.

[83] Jose Renato Santos and Richard R. Muntz. Performance analysis of the rio multimedia storage system with heterogeneous disk configurations. In *Proceedings of the ACM International Multimedia Conference (ACM MM)*, pages 303–308, Bristol, UK, September 1998.

[84] Jose Renato Santos, Richard R. Muntz, and A. Ribeiro-Neto Berthier. Comparing random data allocation and data striping in multimedia servers. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 44–55, Santa Clara, CA, USA, 2000.

[85] Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere. CODA: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):441–459, April 1990.

[86] Frank Schmuck and Roger Haskin. Gpfs: A shared-disk file system for large computing clusters. In *Proceedings of the First Conference on File and Storage Technologies (FAST)*, January 2002.

[87] Henning Schulzrinne. Operating system issues for continuous media. *ACM/Springer Multimedia Systems*, 4(5):269–280, 1996.

[88] Seagate.com. Product listing - disc drives - mainstream servers. http://www.seagate.com/cda/products/-discsales/servermain/.

[89] Margo Seltzer, Keith Bostic, Marshall K. McKusick, and Carl Staelin. An implementation of a log-structured file system for unix. In *USENIX Technical Conf.*, pages 307–326, San Diego, CA, USA, January 1993.

[90] Prashant J. Shenoy. The case for reexamining integrated file system design. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 51–54, Chapel Hill, NC, USA, June 2000.

[91] Prashant J. Shenoy, Pawan Goyal, Sriram S. Rao, and Harrick M. Vin. Symphony: An integrated multimedia file system. In *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking (MMCN)*, pages 124–138, San Jose, CA, USA, January 1998.

[92] Prashant J. Shenoy, Pawan Goyal, and Harrick M. Vin. Architectural considerations for next generation file systems. In *Proceedings of the ACM International Multimedia Conference (ACM MM)*, pages 457–467, Orlando, FL, USA, October 1999.

[93] Prashant J. Shenoy, Saif Hasan, Purushottam Kulkarni, and Krithi Ramamritham. Middleware versus native os support: Architectural considerations for supporting multimedia applications. In *Proceedings of the IEEE Real-time Technology and Applications Symposium (RTAS)*, pages 23–34, San Jose, CA, USA, September 2002.

[94] Prashant J. Shenoy and Harrick M. Vin. Efficient striping techniques for multimedia file servers. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 25–36, St. Louis, MO, USA, May 1997.

[95] Prashant J. Shenoy and Harrick M. Vin. Cello: A disk scheduling framework for next-generation operating systems. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, June 1998.

[96] Prashant J. Shenoy and Harrick M. Vin. Cello: A disk scheduling framework for next generation operating systems. *Real-Time Systems Journal: Special Issue on Flexible Scheduling of Real-Time Systems*, 22(1):9–47, 2002.

[97] Avi Silberschatz, P.B. Galvin, and G. Gagne. *Operating System Concepts*. John Wiley & Sons, 6 edition, 2003.

[98] Silicon Graphics, Inc., 1600 Amphitheatre Pkwy., Mountain View, CA 94043. *SGI CXS Clustered File System, DataSheet*.

[99] IBM Open Source JFS Project Web Site. Journaled file system technology for linux. http://www-124.ibm.com/developerworks/opensource/jfs, March 2003.

[100] David A. Solomon and Mark E. Russinovich. *Inside Microsoft Windows 2000*. Microsoft Press, 3rd edition, 2000. ISBN 0-7356-1021-5.

[101] Ralf Steinmetz. Analyzing the multimedia operating system. *IEEE Multimedia*, 2(1):68–84, 1995.

[102] Ralf Steinmetz and Klara Nahrstedt. *Multimedia: Computing, Communications & Applications*. Prentice Hall, 1995.

[103] Adam Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto, and Geoff Peck. Scalability in the XFS file system. In *Proceedings of the USENIX Technical Conference*, pages 1–14, San Diego, CA, USA, 1996.

[104] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 2001.

[105] Renu Tewari, Richard P. King, Dilip D. Kandlur, and John Wilkes. Placement of multimedia blocks on zoned disks. In *Proceedings of IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN)*, San Jose, CA, USA, January 1996.

[106] H. Tezuka and T. Nakajima. Simple continuous media storage server on real-time mach. In *USENIX Annual Technical Conf.*, San Diego, CA, USA, January 1996.

[107] Chandramohan A. Thekkath, Timothy Mann, and Edward K. Lee. Frangipani: A scalable distributed file system. In *Proceedings of the Symposium on Operating Systems Principles*, pages 224–237, 1997.

[108] Tsun-Ping J. To and Babak Hamidzadeh. Dynamic real-time scheduling strategies for interactive continuous media servers. *ACM/Springer Multimedia Systems*, 7(2):91–106, 1999.

[109] Fourad A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID: A disk array management system for video files. In *Proceedings of the ACM International Multimedia Conference (ACM MM)*, pages 393–400, Anaheim, CA, USA, 1993.

[110] Harrick M. Vin, Sriram S. Rao, and Pawan Goyal. Optimizing the placement of multimedia objects on disk arrays. In *Proceedings of the International Conference on Multimedia Computing and Systems (ICMCS)*, pages 158–165, Washington, D.C., USA, May 1995.

[111] P. Vongsathorn and S.D. Carson. A system for adaptive disk rearrangement. *Software—Practice and Experience*, 20(3):225–242, March 1990.

[112] Chuanbao Wang, Denise J. Ecklund, Earl F. Ecklund, Vera Goebel, and Thomas Plagemann. Design and implementation of a lod system for multimedia supported learning for medical students. In *World Conference on Educational Multimedia, Hypermedia & Telecommunications (ED-MEDIA)*, Tampere, Finland, June 2001.

[113] Chuanbao Wang, Vera Goebel, and Thomas Plagemann. Techniques to increase disk access locality in the minorca multimedia file system. In *Proceedings of the ACM International Multimedia Conference (ACM MM)*, Orlando, FL, USA, October 1999.

[114] Joel L. Wolf, Philip S. Yu, and Hadas Shachnai. DASD dancing: a disk load balancing optimization scheme for video-on-demand computer systems. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 157–166, Ottawa, Ontario, Canada, 1995.

[115] Joel L. Wolf, Philip S. Yu, and Hadas Shachnai. Disk load balancing for video-on-demand systems. *ACM/Springer Multimedia Systems*, 5(6):358–370, 1997.

[116] B.L. Worthington, G.R. Ganger, and Y.N. Patt. Scheduling algorithms for modern disk drives. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 241–251, Nashville, TN, USA, May 1994.

[117] Chiung-Shien Wu, Gin-Kou Ma, and Mei-Chian Liu. A scalable storage supporting multistream real-time data retrieval. *ACM/Springer Multimedia Systems*, 7(6):458–466, 1999.

[118] SGI Developer Central Open Source Linux XFS. XFS: A high-performance journaling filesystem. http://oss.sgi.com/projects/xfs/, March 2003.

[119] Philip S. Yu, Ming-Syan Chen, and Dilip D. Kandlur. Design and analysis of a grouped sweeping scheme for multimedia storage management. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 44–55, La Jolla, CA, USA, November 1992.

[120] A. Zhang and S. Gollapudi. Qos management in educational digital library environments. Technical Report CS-TR-95-53, State University of New York at Buffalo, New York, NY, USA, 1995.