

Multimedia Support for Databases

Banu Özden

Rajeev Rastogi

Avi Silberschatz

Bell Laboratories
700 Mountain Avenue
Murray Hill, NJ 07974

Abstract

Next generation database systems will need to provide support for both textual data and other types of multimedia data (e.g., images, video, audio). These two types of data differ in their characteristics, and hence require different techniques for their organization and management. For example, continuous media data (e.g., video, audio) requires a guaranteed transfer rate. In this paper, we provide an overview of 1) how database systems can be architected to support multimedia data, and 2) what are the main challenges in devising new algorithms to manage multimedia data. In order to provide rate guarantees for continuous media data, an *admission control* scheme must be employed that determines, for each client, whether there are sufficient resources available to service that client. To maximize the number of clients that can be admitted concurrently, the various system resources must be allocated and scheduled carefully. In terms of disks, we use algorithms for retrieving/storing data from/to disks that reduce seek latency time and eliminate rotational delay, thereby providing high throughput. In terms of main-memory, we use buffer management schemes that exploit the sequential access patterns for continuous media data, thereby resulting in efficient replacement of buffer pages from the cache. In addition to discussing resource scheduling, we also present schemes for the storage layout of data on disks and schemes that provide fault-tolerance by ensuring uninterrupted service in the presence of disk failures.

1 Introduction

New advances in computing, communication and storage technologies have stimulated the invention of new multimedia applications, which in turn are pushing these technologies to their limit. Examples of such application domains include digital libraries, archival and processing of images captured by remote-sensing satellites and air photos, training and education, en-

tertainment, medical databases containing X-rays and MRIs and special-purpose databases that contain face and fingerprint data. These applications deal with large volumes of multimedia data that need to be stored either in a database or file/storage system. Conventional database and file systems, however, do not provide the basic support needed for dealing with multimedia data. The difficulty stems from the fact that multimedia data, such as images, video and audio, differ from conventional data (e.g., text) in their characteristics, and hence require different techniques for their organization and management. The challenge is to support all these different types of data under one unified umbrella. In the following, we first discuss the need for multimedia support in database and file systems, and then the need for database functionality to support multimedia data effectively.

What are the characteristics of multimedia data that complicate their storage in conventional databases or file systems? First, multimedia data tends to be voluminous. For example, a 100 minute video compressed using the MPEG-I compression algorithm requires about 1.25 GB of storage space, and a 100 minute video compressed using the JPEG compression algorithm requires about 8.3 GB of storage space. Most databases and file systems do not provide support for such large objects. Second, *continuous media* data, such as video and audio have timing characteristics associated with them. For example, video clips, which are typically stored in units of frames, must be delivered to viewers at a certain rate (which is typically 30 frames/sec). For video compressed using the MPEG-I standard, this translates to a data rate of approximately 1.5 Mbps. Similarly, some of the multimedia data needs to be collected in real-time without losing portions of the data (e.g., images collected by sensors, stock quotes, etc.) and real-time storage impose timing constraints on the multimedia data. Conventional databases and file systems are not designed to meet the real-time characteristics of multimedia data. Moreover, continuous media data requires support for interactive control (e.g., VCR-like operations), synchronization between different media (e.g.,

Permission to make digital/hard copies of all or part of this material for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copyright is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires specific permission and/or fee

PODS '97 Tucson Arizona USA

Copyright 1997 ACM 0-89791-910-6/97/05 ...\$3.50

lipsynching) and authoring (e.g., to generate multimedia presentations); all of these are not supported by conventional systems.

Due to the high storage and bandwidth requirements of multimedia data, it is vital to manage critical resources such as memory, disks and tapes in a manner that provides reasonable throughput and response time so that cost-effective systems can be built. However, resource management and data organization methods in conventional databases and file systems are not amenable for achieving this.

Finally, in order to support a wide range of multimedia applications, it is essential that multimedia objects can be queried and retrieved based on their content. This, in turn, requires sophisticated feature extraction, feature representation and indexing schemes that are again not supported by conventional systems.

Currently, most of the multimedia data is stored in either multimedia or conventional file systems. This, however, raises a host of problems including difficulty in accessing data, redundancy, inconsistency, concurrent access anomalies, as well as integrity, atomicity and security problems. The impact of these problems become more pronounced with the proliferation of multimedia data and multimedia applications. Therefore, the need for database functionality, such as query processing, indexing, concurrency control, recovery, durability and atomicity, is emerging rapidly.

The following are two possible approaches to providing database functionality for multimedia data.

1. The multimedia data as well as the metadata for it is stored together in a single database system.
2. The multimedia data is stored in a separate file system while the corresponding metadata for it is stored in a database system.

Both approaches have their advantages and disadvantages. The first approach implies that databases need to be redesigned to support multimedia data along with conventional data. Since this requires modifications to existing databases, businesses may not be convinced to replace their databases with a new one to accommodate multimedia. Furthermore, this integrated approach might be a burden for users who do not need full-fledged multimedia support. The second approach allows businesses to capitalize on their existing base and purchase a multimedia storage system and the necessary glue to integrate their databases with the multimedia storage system. This approach, however, may complicate the implementation of some of the database functionality such as data consistency.

The remainder of this paper is organized as follows. In Section 2, we discuss the issues for supporting content-based queries and provide a brief survey of research done in this arena. In Sections 3-7, we address various issues

related to the storage and retrieval of continuous media data.

2 Support for Content-Based Queries

Numerous multimedia applications require access to data based on content. However, it is difficult to represent the content of most non-textual types of multimedia data (e.g., images, video and audio) with simple attributes. Also, multimedia objects that may widely differ in their uninterpreted binary representations can be perceptually *similar*, which makes similarity testing a difficult task. For example, two images that differ in pixels can be visually similar. As a result, string matching techniques fail to identify whether a multimedia object has a given pattern/feature. Thus, in order to extract features, the data is typically transformed into a space (e.g., Hilbert space) where the distance between perceptually similar objects becomes close. Once features are extracted, a multimedia object can be represented as a set of feature vectors. The feature vectors per multimedia object can be precomputed and stored in a database. Then, similarity queries can be performed against the database of stored features using distance functions that are aimed to match human perception in sorting the objects in the database in the order of similarity. For example, if a user wants to access images that are similar to an example image, its features are compared to the ones stored in the database and the multimedia objects that are the best match are returned to the user.

Features of a multimedia object include color, texture (e.g., contrast, coarseness, directionality), shape, text (i.e., a set of keywords and other text, such as annotations, associated with a multimedia object) and motion. There can also be some features specific to a particular application domain and others based on knowledge (e.g., the features of center, left and right fields of a baseball field are stored, and a query such as "hits to left fielder" is performed based on searching "left field") [38, 18]. An active research area is the study of issues such as which features are best, how these features should be extracted, represented and compared, and what similarity measures are appropriate.

One of the main issues that needs to be addressed for supporting content-based queries is high dimensional data structures. The number of dimensions of feature vector spaces for multimedia objects can be high (e.g., color space can be 64-dimensional [14]). In some cases the number of dimensions can be above one hundred [37]. However, most multidimensional indexing methods, such as grid files [17] and R-trees [12] do not scale well to high dimensions. These data structures result in search times with exponential complexity in the number of dimensions or a linear search as dimensionality increases. One approach to dealing with

high dimensionality is to map the higher dimensional feature space to a lower dimensional one using a distance preserving transform [9]. Once the dimensionality of the feature space is reduced, then one of the existing multidimensional data structures, such as R*-trees [3], can be used for indexing feature vectors. Another approach is designing new data structures or modifying the existing ones to scale to higher dimensions [37, 13].

There are a number of projects and systems that are examining some of these problems, such as QBIC [10], Photobook [29], Virage [35], Chabot [18], Infromedia [36] and MARS [15]. For example, Query by Image Content (QBIC) supports queries based on example images, user-constructed sketches and drawings, color, texture, and keywords [10]. Queries based on color are performed as follows. The system stores a color histogram for each image in the database. The color histogram of the example image is compared against those stored in the database using a metric such as mean squared difference. The system returns a set of images sorted by this metric. Technical issues addressed by QBIC include a visual query language, graphical user interface, indexing techniques for high-dimensional features and similarity retrieval. Another example is Photobook [29] which is a tool for performing content-based queries on image databases. Features are compared using either one of the system-provided matching algorithms or a user-defined matching algorithm. Furthermore, an interactive learning agent selects models based on examples from the user to offer assistance in choosing an appropriate model for a given query.

While research on content-based retrieval [9, 37, 13, 10, 29, 18, 36, 15] is evolving, the following issues remain open: the scalability of these approaches to larger databases and/or higher dimensional multimedia objects, integration of higher dimensional data structures into databases, appropriate concurrency control and recovery schemes for these structures, integration of content-based techniques into query processing, and extensions to retrieval and data models and query languages to support content-based retrieval of multimedia objects.

3 Continuous Media Storage and Retrieval

Multimedia applications require support for the storage and retrieval of *multimedia* data, which typically consists of video, audio, text and images. As described earlier, this data can be categorized into *continuous media* (CM) data (e.g., video, audio) and *non-continuous media* data (e.g., text, images).

The timing characteristics and the large volumes of CM data make the design of a multimedia storage server a challenging task. Such a server should:

1. provide rate guarantees for the storage and retrieval

of CM data.

2. provide support for VCR operations (e.g., fast-forward, rewind, pause).
3. provide support for retrieval of non-CM data concurrently with CM data.
4. manage critical storage resources such as memory, disks and tapes so as to maximize throughput and reduce response times.
5. provide support for authoring and synchronization between different CM data.

A multimedia server will typically employ several secondary (e.g., disk) and tertiary (e.g., tapes, optical disks) storage devices to permanently store the data. A small amount of RAM is used to stage the data retrieved from disks and tapes before it is transmitted to clients. Cost, latency and transfer rates of these devices are as described in the table below.

Storage Device	Cost/MB	Latency	Data Rate
Magnetic Disks	20c	25 ms	5-8 MBps
Tapes (low end)	0.5c	3 min	1.5 MBps
Tapes (high end)	0.7c	2-3 min	10 MBps
Optical Disks	10c	35 ms	1.25 MBps

Architecting a multimedia server which is cost-wise optimal from the standpoint of minimizing storage cost is an important and challenging research problem. Since tapes have the least storage cost, the cost of a server can be reduced considerably by storing CM clips primarily on tapes. However, since tapes have very high latencies for data access, in order to retrieve data for a large number of CM clips concurrently, frequently accessed CM clips should be stored on magnetic disks (that have lower access latencies, but are more expensive than tapes). Thus, a multimedia server consists of a storage hierarchy with tapes at the bottom, disks in the middle and RAM at the top. The exact configuration of the server (e.g., the number of disks and tapes, the amount of RAM) would need to be determined based on the number of clips and the frequency with which they are accessed.

In subsequent sections, we show how a storage server can use *admission control* in order to provide rate guarantees for CM data. We address disk scheduling and buffer management as each of these issues presents new challenges to CM applications. First, disk scheduling is crucial to achieving high throughputs since disks have relatively low transfer rates (e.g., 32-60 Mbps) and a relatively high latency for data access (e.g., 20-30 ms). Second, most existing buffer page replacement algorithms like *least recently used* (LRU) and *most recently used* (MRU) are only approximation algorithms that ignore

inner track transfer rate	r_{disk}	45 Mbps
Settle time	t_{settle}	0.6 ms
Seek time (worst case)	t_{seek}	17 ms
Rotational latency (worst case)	t_{rot}	8.34 ms
Worst case latency	t_{lat}	25.5 ms
Cost	C_d	\$700
Capacity		2 GB

Figure 1: Disk Parameters for a Commercially Available Disk

the data access patterns of the higher level applications. We present a new and simple buffering scheme that exploits the sequential access patterns to CM data when determining the next page to be replaced from the buffer cache [24]. This reduces cache misses and hence disk I/O's with a consequent improvement in performance. We then show how *disk striping* can be used to distribute the load uniformly across several disks, and present an effective disk layout scheme that completely eliminates disk latency. Finally, we present schemes that enable uninterrupted retrieval of CM data even if a disk were to fail.

4 Disk Storage Issues

A CM server is a computer system consisting one or more processors, RAM, and disks to store data. For the development of disk scheduling schemes, it is important to understand the characteristics of disks, which we first describe. Data on disks is stored in a series of concentric circles, or *tracks*, and accessed using a disk head. Disks rotate on a central spindle and the speed of rotation determines the transfer rate of disks. Data on a particular track is accessed by positioning the head on (also referred to as *seeking* to) the track containing the data, and then waiting until the disk rotates enough so that the head is positioned directly above the data. Seeks typically consist of a coast during which the head moves at a constant speed and a settle, when the head position is adjusted to the desired track. Thus, the latency for accessing data on disk is the sum of seek and rotational latency. In the table of Figure 1, we present the notation and values employed in the paper for the various disk characteristics (the values are for a commercially available disk [1]).

CM clips are stored on disks in compressed form; each CM clip C_i , depending on the compression algorithm (e.g., MPEG-1, MPEG-2), needs to be displayed at a certain rate, r_i . The server is connected to clients via a high speed network over which clip C_i is transmitted to clients at the rate r_i . Data for CM clips is retrieved from disks in *rounds* of duration T , where in each round, $T \cdot r_i$ bits are retrieved for clip C_i .

A buffer of size $2 \cdot T \cdot r_i$ is allocated for a clip C_i before data retrieval for it is initiated. In addition, at the end of the round in which the first $T \cdot r_i$ bits for the clip are retrieved, transmission of data to clients that requested the clip is begun. Thus, during each round, $T \cdot r_i$ bits for a clip are retrieved into its buffer, and concurrently, $T \cdot r_i$ bits for the clip are transmitted to clients.

During a round, data for the clips being serviced is retrieved from the disk using the CSCAN disk scheduling algorithm [32]. CSCAN scheduling ensures that the disk head moves in a single direction when servicing clips during a round. As a result, random seeks to arbitrary locations are eliminated. Note that, in order to ensure that data for clips is retrieved at the required rate, we require the time to service clips during a round to never exceed T , the duration of the round. Since, during a round, the disk head travels across the disk at most twice, and retrieving data for each clip, in the worst case, incurs a settle and a worst-case rotational latency overhead, we require the following equation to hold (C_1, C_2, \dots, C_q are the clips being serviced):

$$2 \cdot t_{seek} + q \cdot (t_{rot} + t_{settle}) + \sum_{i=1}^q \frac{T \cdot r_i}{r_{disk}} \leq T \quad (1)$$

Thus, a new client request for a clip is admitted by the server into the system, if on adding it to the list of clips being serviced, Equation 1 above holds and there is at least $2 \cdot T \cdot r_i$ bits of free buffer space available for the clip. If r_{max} is the maximum rate among all the clips, then the minimum number of clips that can be serviced during a round, q_{min} , can be obtained by solving the following equation.

$$2 \cdot t_{seek} + q_{min} \cdot (t_{rot} + t_{settle} + \frac{T \cdot r_{max}}{r_{disk}}) \leq T \quad (2)$$

A number of additional problems arise when retrieving data from disks. In modern disks, outer tracks have higher transfer rates than inner tracks. By taking into account the disk transfer rate of the tracks where CM clips are stored, one could substantially reduce buffer requirements. Also, to service both CM as well as non-CM data, schemes would either have to reserve a portion of a round or use the *slack* time during a round to service non-CM requests (see [21]). Furthermore, if clips are frequently inserted and deleted, then they must be stored non-contiguously on disks (in fixed size blocks) to reduce fragmentation [21]. Unlike conventional file systems that use a small block size (4KB-8KB), the block size for clip C_i must be chosen to be as close to $T \cdot r_i$ (the data retrieved during a round) as possible, in order to reduce the latency overhead. Index structures for accessing random blocks for a clip also need to be designed.

A number of other schemes for handling the storage and retrieval of continuous media data from disk have been proposed in the literature [2, 30, 19, 31, 6]. Among the existing work, [2, 31] address the issue of servicing non-CM data requests concurrently with CM requests. In [21], algorithms for completely eliminating the rotational latency of disks are proposed, while the schemes presented in [2, 30] do not attempt to reduce disk latency. In order to reduce buffer requirements, the *grouped sweeping scheduling* (GSS) scheme for servicing CM clips was proposed in [6]. Requests are organized into groups; requests in each group are serviced using elevator scheduling, while individual groups are serviced in a given fixed-order.

5 Buffer Management Issues

By having requests share a global pool of buffer pages, the number of I/O requests can be considerably reduced, thereby enabling a larger number of requests to be serviced [26]. For example, if two requests for a clip arrive at an interval of 5 or 10 seconds, then by caching the pages accessed by the first request, disk accesses for the second requests can be totally eliminated.

An important research issue is that of buffer page replacement policy. Existing page replacement policies may be unsuitable since they exploit neither the knowledge of outstanding requests nor the fact that CM clip data is accessed predominantly sequentially. For example, consider a server with 100 buffer pages that follows the *least recently used* (LRU) policy (this is followed in most conventional storage servers). Let data retrieval for clips C_1 and C_2 (with the same rate) be initiated at some round k . Thus, at round $k + 51$, the first page of C_1 and C_2 are replaced to make room for the 51st page. If a request for C_1 were to arrive in round $k + 52$, then with LRU, pages for the second request for C_1 would need to be accessed separately from the disks (since the page to be retrieved for the request during a round would have been paged out during the previous round). However, if, after the second request for C_1 arrives, instead of replacing C_1 's pages, we were to replace only pages accessed by C_2 , then only the first page for C_1 would need to be accessed from the disk and subsequent pages would already be present in the buffer when they are needed, thus eliminating the need for subsequent disk accesses for the second request for C_1 . Thus, a buffer page replacement policy that takes into account requests being serviced would result in better performance.

In [22], a new buffer replacement algorithm, BASIC, was presented. When a buffer page is to be allocated, BASIC selects for replacement, the buffer page that would not be accessed for the longest period of time by the existing clients (if each client consumed data at the specified rate). Furthermore, if there are buffer pages

that would not be accessed by the existing clients, the page with the highest *offset-rate ratio* is selected as the victim (e.g., the tenth page of a clip with a rate of 1.5 Mbps in a system where the page size is 32 KB will have an offset of 288 KB and an offset-rate ratio of $288 \text{ KB}/1.5 \text{ Mbps} = 1.536$ seconds). In [26], a demand-based buffer allocation algorithm was presented, which optimizes the buffer requirement of a video-on-demand system; however, in that paper, the issue of buffer page replacement was not considered.

6 Disk Striping Issues

Since CM clips require large amounts of storage space, a CM server typically, employs several disks. As a result, schemes for laying out the clips on multiple disks are crucial to distributing the load uniformly across the various disks thereby utilizing the disk bandwidth effectively. For example, certain clips may be more popular than others, and a naive approach that stores every clip on an arbitrarily chosen disk could result in certain disks being over-burdened with more requests than they can support, while other disks remain underutilized. Also, unless a clip is replicated on several disks, the number of clients that are concurrently accessing portions of the clip is bounded above by the bandwidth of the disk storing the clip.

The above problems can be alleviated by using *disk striping*, a popular technique in which consecutive logical data units (referred to as *stripe units*) are distributed among the disks in a round-robin fashion [28, 11, 7, 4, 23]. Disk striping, in addition to distributing the workload uniformly across disks, also enables multiple concurrent streams of a clip to be supported without having to replicate the clip.

Striping can be either *fine-grained* or *coarse-grained*. In fine-grained striping, the striping unit is typically a bit, a byte or a sector [11]. For clip C_i , each retrieval unit of size $T \cdot r_i$ is distributed among all the disks (see Figure 2(a)). As a consequence, if there are m disks, then every retrieval involves all the m disk heads, and the m disks behave like a single disk with bandwidth $m \cdot r_{disk}$. This striping technique is followed in the RAID-3 data distribution scheme [28].

In coarse-grained striping, the size of stripe units for clip C_i is much larger; it is $T \cdot r_i$, the amount of data typically retrieved during a single disk access [11, 4]. Thus, in contrast to fine-grained striping in which all disk arms are involved in data retrieval during each access, in coarse-grained striping, usually, only a single disk is involved. Consecutive stripe units of size $T \cdot r_i$ belonging to clip C_i are stored on the m disks in a round-robin fashion (see Figure 2(b)). For large requests and for sequential accesses to data, coarse-grained striping distributes the workload evenly among the various disks. This striping technique is followed in the RAID-5 data

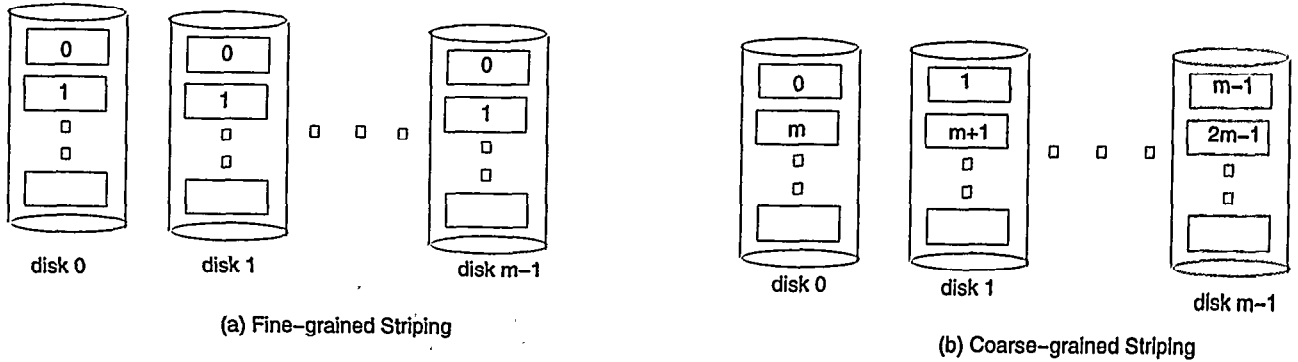


Figure 2: Striping Schemes

distribution scheme [28].

In environments like Movies-on-Demand where there are a large number of requests for a few movies, retrieving data for the movies periodically and having multiple client requests share the retrieved data could result in decreased response times. To illustrate, consider a server that stores 50 movies, each of duration 100 minutes, and is configured to support 1000 concurrent retrievals for movies. Suppose requests for each movie arrive every minute. Thus, every minute, a new retrieval for each movie is initiated. As a result, after the first 20 minutes, the server has to support 20 retrievals for each movie and thus, no further new retrievals can be initiated for the next 80 minutes (after 80 minutes, a new retrieval for each movie can be initiated). Consequently, the average response time for each of the 80 following requests for each movie is 40 minutes, and over all requests, the average response time is 32 minutes. Furthermore, the worst-case response time for a request is 80 minutes.

Now, suppose that the server were configured such that a new retrieval for each movie is initiated at fixed intervals of 5 minutes. Thus, at any point in time, the number of concurrent retrievals for each movie is 20; the retrievals being distributed uniformly across the duration of the movie. In this case, the average response time for a request would be 2.5 minutes and the worst-case response time would be 5 minutes. Thus, by retrieving movies periodically from disk, better response times can be provided to client requests. Furthermore, by retrieving frequently requested movies more frequently, and movies that are seldom requested, less frequently, better over-all average response times can be provided. Thus, the periods for the various movies can be determined based on factors like their popularity. Finally, letting clients know about the periods and the exact times at which movies are retrieved is a desirable feature for clients.

6.1 CM Clips with Different Periods

Each CM clip is associated with a period. The period P_i for a clip C_i is the distance (in rounds) between rounds in which successive retrievals for C_i are initiated. Thus, starting with a certain round s_i , data retrieval for a clip C_i must be initiated during rounds s_i , $s_i + P_i$, $s_i + 2 \cdot P_i$ and so on.

Depending on the data layout and the periods for clips C_1, \dots, C_n , it may not always be possible to retrieve data for the clips periodically. The reason for this is that data for a limited number of clips can be retrieved during a round. In this section, for the coarse-grained striping scheme (see Figure 2(b)), we show that the problem of retrieving clips periodically is the same as scheduling periodic tasks on a multiprocessor [33]. The scheduling problem is NP-hard and heuristics to tackle the problem are presented in [25].

In the coarse-grained striping scheme, the first stripe unit of each clip C_i is stored on disk 0. Successive consecutive stripe units of size $T \cdot r_i$ belonging to each clip C_i are stored on the disks, beginning with disk 1, in a round-robin fashion. Thus, we only need to consider the problem of retrieving data for the clips periodically from disk 0. For every other disk j , the data retrieved during a round is the $T \cdot r_i$ bits immediately following the $T \cdot r_i$ bits retrieved for clip C_i in the previous round from disk $j - 1$.

For every clip C_i , the first stripe unit for C_i must be retrieved from disk 0 starting at some round s_i and subsequently, at intervals of P_i rounds. Furthermore, if w_i stripe units belonging to C_i are stored on disk 0, then each of the $w_i - 1$ remaining stripe units need to be retrieved from disk 0 at intervals of m rounds (m is the number of disks). Thus, if the first stripe unit for a clip C_i is retrieved from disk 0 during round u , then the $w_i - 1$ subsequent stripe units belonging to C_i need to be retrieved from disk 0 during rounds $u + m, u + 2 \cdot m, \dots, u + (w_i - 1) \cdot m$. If q_{min} is the minimum number of clips for which data can be retrieved from a disk during a round (due to Equation 2), then the problem of retrieving data for the clips periodically

is the same as that of scheduling tasks T_1, \dots, T_n on q_{min} processors. Each task T_i consists of w_i subtasks, each subtask has computation time 1 and the interval between any two consecutive subtasks of a task is m . Furthermore, the first subtask of a task T_i must be scheduled at intervals of P_i .

6.2 CM Clips with Same Period

We now present a scheme that completely eliminates disk latency and retrieves each clip with the same period. If C is the capacity of a disk, then the period with which each clip is retrieved is $\frac{C}{r_{disk}}$ (that is, the time to read an entire disk sequentially).

The scheme views the m disks as a single disk with bandwidth $m \cdot r_{disk}$ (as in the fine-grained striping scheme in Figure 2(a)). Let T be an arbitrarily small amount such that $T \cdot r_i$ is a multiple of the unit of retrieval from disk. The disk is viewed as a sequence of blocks of size $T \cdot m \cdot r_{disk}$. For each clip C_i , consecutive portions of size $T \cdot r_i$ are stored in consecutive disk blocks (of size $T \cdot m \cdot r_{disk}$), wrapping around to the first disk block when the last disk block is reached (see Figure 3). Also, the next clip is stored beginning with the disk block following the block in which the last portion of the previous clip was stored. The algorithm for storing the clips terminates once all clips are exhausted or if there is no room remaining in a disk block to store the next portion of a clip. As a result, the sum of all the portions assigned to a block does not exceed its size $T \cdot m \cdot r_{disk}$.

It can be shown that by sequentially reading, from start to end, the blocks of size $T \cdot m \cdot r_{disk}$ from disk, data for clip C_i can be retrieved at a rate r_i . The reason for this is that the time to retrieve an entire block from disk is T (since the size of the block is $T \cdot m \cdot r_{disk}$). Also, the time to consume a portion of size $T \cdot r_i$ at rate r_i is T . Thus, in the time that it takes to consume a portion of size $T \cdot r_i$, the next block containing the next portion can be retrieved from disk. Once the disk head reaches the end, it is repositioned to the start and the sequential reading of disk blocks is resumed. Thus, retrieval of each clip's data is initiated at intervals of $\frac{C}{r_{disk}}$ - the time to read an entire disk sequentially.

Repositioning the disk head to the start after it has reached the end can be handled as described in [19, 24] by storing an appropriate number of blocks at the end in RAM (to mask the time it takes the head to seek from the end to the start).

7 Fault-Tolerance Issues

For a single disk, the *mean time to failure* (MTTF) is about 300,000 hours. Thus, a server, with, say, 200 disks has an MTTF of 1500 hours or about 60 days. Since data on a failed disk is inaccessible until the disk has been repaired, a single disk failure could result in the interruption of service, which in

many application domains is unacceptable. In order to provide continuous, reliable service to clients, it is imperative that it be possible to reconstruct data residing on a failed disk in a timely fashion. Our goal is to develop schemes that make it possible to continue transmitting CM data at the required rate even if a disk failure takes place.

A number of schemes for ensuring the availability of data on failed disks have been proposed in the literature [8, 27]. A majority of the schemes employ data redundancy in order to cope with disk failures. Typically, for a group of data blocks residing on different disks, a parity block containing parity information for the blocks is stored on a separate disk (the data blocks along with the parity block form a *parity group*). In case a disk containing a data block were to fail, the remaining data blocks in its parity group and the parity block are retrieved, and the data block on the failed disk is reconstructed.

A similar approach can be employed in a multimedia storage server to ensure high data availability in the presence of disk failures. However, since a multimedia server must also provide rate guarantees for CM data, it must retrieve in a timely fashion (from the surviving disks) the additional blocks needed to reconstruct the required data blocks. This may not be possible unless for every additional block either 1) it has been pre-fetched and is already contained in the server's buffer, or 2) the bandwidth required for retrieving it has been reserved *a-priori* on the disk containing it. In this section, we propose schemes that exploit the sequential playback of CM data in order to pre-fetch all the data blocks belonging to a parity group before the first data block of the parity group is accessed. As a result, if a disk were to fail, for every data block to be retrieved from the failed disk, since the remaining data blocks in its parity group have been pre-fetched, only the parity block for its parity group is retrieved and the data block is reconstructed before it is accessed¹. Thus, all the data blocks to be retrieved from the failed disk during a round are available in the buffer at the end of the round. Also, since only a single parity block per data block on the failed disk needs to be retrieved, the additional load generated and the bandwidth that needs to be reserved on each disk is reduced. In sections 7.1 and 7.2, we present two schemes for parity data placement (a) separate parity disks are used to store parity blocks; (b) parity blocks are distributed among all the disks.

A vast majority of the experimental, analytical, and simulation studies for RAID-based disk arrays assumes a conventional workload [8, 27]. Reads and writes access small amounts of data, are independent of each other, are aperiodic, and do not impose any real-

¹We assume that the cost of reconstructing the data block by xor'ing the blocks in its parity group is negligible in comparison to the cost of retrieving the blocks from disk.

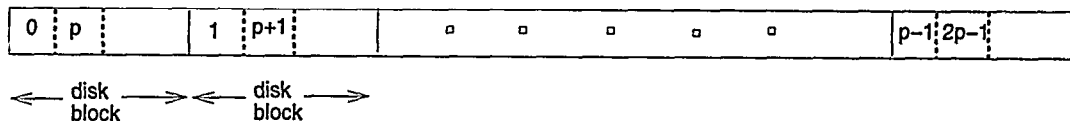


Figure 3: Data Layout

time requirements. In contrast, access to CM data is sequential, periodic, and imposes real-time constraints. Schemes that exploit the inherent characteristics of CM data for data placement and recovery, and that enable CM data to be retrieved at a guaranteed rate despite disk failures were proposed in [5, 16, 34].

In [16], the authors presented the *doubly-striped* mirroring scheme which distributes mirror blocks for data blocks on a disk among all other disks. The scheme ensures that in case of a disk failure, the mirror blocks to be retrieved are uniformly distributed across the remaining disks. However, the scheme has a high (100%) storage overhead since every data block is replicated.

In [34], the authors presented the *streaming RAID* approach which uses parity encoding techniques to group disks into fixed size clusters of p disks each with one parity disk and $p-1$ data disks. A set of $p-1$ data blocks, one per data disk in a cluster, and its parity block (stored on the parity disk in the cluster) form a *parity group*. The granularity of a read request is an entire parity group; as a result, the parity block is always available to reconstruct lost data in case a data disk fails. The streaming RAID scheme has high buffer requirements since it retrieves an entire parity group in every access.

To reduce the buffer space overhead, for environments in which a lower level of fault tolerance is acceptable, a non-clustered scheme was proposed in [5], where disks are organized in clusters, each cluster containing a single parity disk. In the event of a disk failure, entire parity groups are read, but only for parity groups containing the faulty disk. The non-clustered scheme, however, has the following drawback. During the transition from retrieving individual data blocks to retrieving entire parity groups for a failed cluster, blocks for certain clips may be lost and thus, clients may encounter discontinuities in playback.

7.1 With Parity Disks

In the first parity block placement policy, the m disks are grouped into clusters of size p with a single dedicated disk within each cluster to store parity blocks (referred to as the *parity disk*). For a CM clip C_i with rate r_i , data blocks of size $T \cdot r_i$ are stored on consecutive data disks (these exclude parity disks) using a round-robin placement policy. The first data block of each CM clip is stored on the first data disk within a cluster.

Furthermore, $p-1$ consecutive data blocks in a single cluster along with the parity block for them (stored on the parity disk for the cluster) form a parity group.

As described in Section 4, an *admission control* algorithm is used to determine if a client request for a clip can be serviced. The determination is based on whether there is sufficient disk bandwidth to service the requested CM clip; if this is the case, then buffer space is allocated for the clip and data retrieval for the CM clip is initiated. Due to the periodic nature of playback of audio and video clips, the server retrieves data for clips in *rounds* of duration T . A service list is maintained for every cluster, and it contains the CM clips for which data is being retrieved from the cluster during a round. During each round, for every service list, $p-1$ consecutive blocks from the $p-1$ disks in the cluster are retrieved concurrently for each CM clip in the list (a single block is retrieved from each disk in the cluster) using the C-SCAN disk scheduling algorithm [32]. In order to maintain continuity of playback, the duration of a round must not exceed T . This can be ensured by restricting the number of CM clips in each service list so that the time required by the server to retrieve blocks for CM clips in the service list does not exceed T . Since, during a round, disk heads travel across the disk at most twice (due to C-SCAN scheduling), and retrieving data for each CM clip, in the worst case, incurs a settle and a worst-case rotational latency overhead, we require the following equation to hold for clips C_1, \dots, C_q in the service list of a cluster [6, 21]:

$$\sum_{i=1}^q \frac{T \cdot r_i}{r_{disk}} + q \cdot (t_{rot} + t_{settle}) + 2 \cdot t_{seek} \leq T \quad (9)$$

Since the number of data disks in a cluster is $p-1$, the data retrieved for a clip C_i in a round is $(p-1) \cdot T \cdot r_i$. Thus, for the next $p-2$ rounds, no data is retrieved for C_i and during the $p-1$ th round after the current round, the next $p-1$ blocks for C_i are retrieved from the next cluster. Consequently, the service list for a cluster becomes the service list for the next cluster after $p-1$ rounds. Also, for a clip C_i , transmission of data to the client is begun at the end of the round during which the first $p-1$ blocks for the request are retrieved. As a result, at the start of a subsequent round in which data is retrieved for clip C_i from a cluster, one block of data is contained in the buffer. At the end of the round, $p-1$ data blocks for C_i are contained in the buffer, and

0	1	2	3	4	5	6	7	8
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8
D_9	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}	D_{16}	D_{17}
D_{18}	D_{19}	D_{20}	D_{21}	D_{22}	D_{23}	D_{24}	D_{25}	D_{26}
D_{27}	D_{28}	D_{29}	D_{30}	D_{31}	D_{32}	D_{33}	D_{34}	D_{35}
D_{36}	D_{37}	D_{38}	D_{39}	D_{40}	D_{41}	D_{42}	D_{43}	D_{44}
D_{45}	D_{46}	D_{47}	D_{48}	D_{49}	D_{50}	D_{51}	D_{52}	D_{53}
P_{10}	P_{13}	P_{16}	P_0	P_3	P_6	P_9	P_{12}	P_{15}
P_2	P_5	P_8	P_{11}	P_{14}	P_{17}	P_1	P_4	P_7

Figure 4: Uniform, flat parity block placement

during the next $p - 2$ rounds, one block is transmitted to clients per round. Thus, the average buffer space required for clip C_i is $\frac{p \cdot T \cdot r_i}{2}$.

In case a disk were to fail, for every data block to be retrieved from the failed disk during a round, the parity block in its parity group is retrieved instead. Thus, for a parity group containing a data block on the failed disk, at the start of the round in which the first data block in the group is transmitted, the $p - 2$ data blocks in the group and the parity block are contained in the buffer (these were retrieved in the previous round). These are used to reconstruct the missing data block on the failed disk, and as a result, a data block on the failed disk is always available in the buffer when it needs to be transmitted. Furthermore, since a separate parity disk is used to store parity blocks for parity groups in a cluster, it is unnecessary to reserve bandwidth on data disks. The admission control scheme only needs to ensure that Equation 3 holds for the clips in the service list of every cluster.

The above scheme is similar to the *staggered group* scheme presented in [5]. It has much lower buffer overhead per request than the streaming RAID scheme in which an entire parity group for a clip is retrieved during every round.

7.2 Without Parity Disks

Maintaining a separate parity disk per cluster can lead to an ineffective utilization of disk bandwidth since most of the parity disks remain idle. To alleviate this drawback, a uniform, flat parity placement policy can be employed at the server [20]. In such a policy, the m disks are grouped into clusters of $p - 1$ disks, and successive CM data blocks are stored on consecutive disks using a round-robin placement policy. Furthermore, $p - 1$ consecutive data blocks within a single cluster along with its parity block form a parity group. Parity blocks for successive parity groups within a cluster are uniformly distributed across the remaining disks. More precisely, the parity block for the i^{th} data block on a disk is stored on the $(i \bmod (m - (p - 1)))^{\text{th}}$ disk following the last disk of the cluster. Figure 4 depicts the uniform, flat

placement policy on a disk array with 9 disks, a cluster size of 3 and a parity group size of 4. D_0, D_1, \dots denote consecutive data blocks and P_i is the parity block for data blocks D_{3i}, D_{3i+1} and D_{3i+2} . Note that the above parity placement scheme is different from the improved bandwidth scheme of [5] in which parity blocks for a cluster are stored only in the adjacent cluster.

Data retrieval for CM clips, both in the presence and absence of failures, is performed as described in the previous subsection. Basically, $p - 1$ consecutive data blocks for a clip are retrieved concurrently from a cluster during a round, and then for the next $p - 2$ rounds, no data is retrieved for the clip. However, since, unlike the scheme presented in the previous subsection, parity blocks are stored on disks containing data blocks and not separate parity disks, *contingency* bandwidth for retrieving the additional parity blocks in case of a disk failure must be reserved on each disk. Let us assume that this bandwidth is $\sigma \cdot T$ ($0 \leq \sigma \leq 1$). Thus, the admission control procedure must ensure that for clips C_1, \dots, C_q in the service list for a cluster,

$$\sum_{i=1}^q \frac{T \cdot r_i}{r_{\text{disk}}} + q \cdot (t_{\text{rot}} + t_{\text{settle}}) + 2 \cdot t_{\text{seek}} \leq (1 - \sigma) \cdot T \quad (4)$$

Notice that parity blocks for the i^{th} data block and the $(i + j \cdot (m - (p - 1)))^{\text{th}}$ data block on a disk, $j \geq 0$, are stored on the same disk. Thus, if during a round, C_1, \dots, C_l are clips in the service list of a failed cluster, with parity blocks on the same disk i , then during the round, parity blocks for C_1, \dots, C_l must be retrieved from disk i during the same round using the reserved contingency bandwidth $\sigma \cdot T$. Thus, the admission control scheme must also ensure that for every service list containing clips C_1, \dots, C_l with parity blocks on the same disk, the following equation holds:

$$\sum_{i=1}^l \frac{T \cdot r_i}{r_{\text{disk}}} + l \cdot (t_{\text{rot}} + t_{\text{settle}}) \leq \sigma \cdot T \quad (5)$$

We do not include $2 \cdot t_{\text{seek}}$ in the left hand side of Equation 5 since it has already been included in Equation 4.

8 Concluding Remarks

In this paper, we provided an overview of how next-generation database systems – systems that need to provide support for both textual data and other types of multimedia data (e.g., images, video, audio), can be architected. We described the main challenges in devising new algorithms to manage multimedia data. One of the main differences between traditional databases and next-generation databases is that the latter need to support the storage and retrieval of continuous media data, and such data has rate guarantees associated with it.

Consequently, algorithms that provide rate guarantees for data retrieved from secondary storage devices (e.g., disks) and tertiary storage devices (e.g., tapes) need to be developed. Schemes for retrieving data from disks must take into account seek and rotational latencies, varying track transfer rates, bad sector remapping and recalibration of tables due to thermal expansion, while schemes for retrieving data from tapes must attempt to reduce the latency for data access. Furthermore, assuming that CM clips are striped across multiple disks, schemes for implementing VCR operations and retrieving data for CM clips periodically (to be shared by multiple client requests) must be devised. Also, since most of the previously proposed schemes for dealing with disk failures do not address the issue of continuously retrieving data at specific rates, algorithms for reconstructing data on a failed disk in a timely fashion need to be developed. Finally, since accesses to CM data is typically sequential in nature, improved buffer page replacement policies that exploit the sequentiality of access need to be devised. In this paper, we have presented various schemes for dealing with these issues. These schemes have been incorporated into the *Fellini* multimedia storage manager implemented at Bell Labs.

References

- [1] *Seagate Product Overview*. October 1993.
- [2] D. P. Anderson, Y. Osawa, and R. Govindan. A file system for continuous media. *ACM Transactions on Computer Systems*, 10(4):311-337, November 1992.
- [3] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The r^* -tree: An efficient and robust access method for points and rectangles. In *Proceedings of ACM-SIGMOD 1990 International Conference on Management of Data, Atlantic City, New Jersey*, pages 322-331, 1990.
- [4] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered striping in multimedia information systems. In *Proceedings of ACM-SIGMOD 1994 International Conference on Management of Data, Minneapolis, Minnesota*, pages 79-90, May 1994.
- [5] S Berson, L Golubchik, and R R. Muntz. Fault tolerant design of multimedia servers. In *Proceedings of SIGMOD Conference*, pages 364-375, 1995.
- [6] M. S. Chen, D. D. Kandlur, and P. S. Yu. Optimization of the grouped sweeping scheduling (gss) with heterogeneous multimedia streams. In *Proceedings of ACM Multimedia, Anaheim, CA*, pages 235-242, August 1993.
- [7] P. Chen, E. Lee, G. Gibson, R. Katz, and D. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):393-400, June 1994.
- [8] P M. Chen, E K. Lee, G A. Gibson, R H. Katz, and D A. Patterson. Raid: High-performance, reliable secondary storage. *Submitted to ACM Computing Surveys*, 1994.
- [9] C. Faloutsos and K. Lin. A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of ACM-SIGMOD 1995 International Conference on Management of Data, San Jose, California*, pages 163-174, June 1995.
- [10] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, and J. Hafner. Query by image and video content: The qbic system. *IEEE Computer*, 28(9):23-48, September 1995.
- [11] G. R. Ganger, B. L. Worthington, R. Y. Hou, and Y. N. Patt. Disk arrays: High-performance, high-reliability storage subsystems. *IEEE Computer*, 27(3):30-36, March 1994.
- [12] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of ACM-SIGMOD 1984 International Conference on Management of Data, Boston, Massachusetts*, pages 47-57, June 1984.
- [13] A. Henrich. Improving the performance of multi-dimensional access structures based on k-d-trees. In *Proceedings of 1996 International Conference on Data Engineering, New Orleans, Louisiana*, pages 68-75, February 1996.
- [14] T. Huang, S. Mehrotra, and K. Ramchandran. Digital image access and retrieval in libraries. In *Proceedings of the Data Processing Clinic*, 1996.
- [15] S. Mehrotra, Y. Rui, M. Ortega, and T. Huang. Content-based query in MARS. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Ottawa, Ontario*, June 1997.
- [16] Antoine Mourad. Reliable disk striping in video-on-demand servers. In *Proceedings of the International Conference on Distributed Multimedia Systems and Applications, Stanford, CA*, August 1995.
- [17] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38-71, March 1984.

- [18] V. E. Ogle and M. Stonebraker. Chabot: Retrieval from a relational database of images. *IEEE Computer*, 28(9):40-56, September 1995.
- [19] B. Özden, A. Biliris, R. Rastogi, and A. Silberschatz. A low-cost storage server for movie on demand databases. In *Proceedings of the Twentieth International Conference on Very Large Databases, Santiago*, September 1994.
- [20] B. Özden, R. Rastogi, P. Shenoy, and A. Silberschatz. Fault-tolerant architectures for continuous media servers. In *Proceedings of SIGMOD Conference*, 1996.
- [21] B. Özden, R. Rastogi, and A. Silberschatz. A framework for the storage and retrieval of continuous media data. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Washington D.C.*, May 1995.
- [22] B. Özden, R. Rastogi, and A. Silberschatz. Buffer replacement algorithms for multimedia databases. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Hiroshima, Japan*, June 1996.
- [23] B. Özden, R. Rastogi, and A. Silberschatz. Disk striping in video server environments. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Hiroshima, Japan*, June 1996.
- [24] B. Özden, R. Rastogi, and A. Silberschatz. On the design of a low-cost video-on-demand storage system. *Multimedia Systems*, pages 40-54, February 1996.
- [25] B. Özden, R. Rastogi, and A. Silberschatz. Periodic retrieval of videos from disk arrays. In *Proceedings International Conference on Data Engineering, Birmingham*, April 1997.
- [26] B. Özden, R. Rastogi, A. Silberschatz, and C. Martin. Demand paging for movie-on-demand servers. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Washington D.C.*, May 1995.
- [27] D. Patterson, G. Gibson, and R. Katz. A case for redundant array of inexpensive disks (raid). In *Proceedings of ACM SIGMOD'88*, pages 109-116, June 1988.
- [28] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks(RAID). In *Proceedings of ACM-SIGMOD 1988 International Conference on Management of Data, Chicago, Illinois*, 1988.
- [29] A. Pentland, R. Picard, and S. Sclaroff. Photobook: Tools for content-base manipulation of image databases. In *Proceedings of SPIE Conference on Storage and Retrieval of Image and Video Databases II, San Jose, California*, February 1994.
- [30] P. V. Rangan and H. M. Vin. Designing file systems for digital video and audio. In *Proceedings of the Thirteenth Symposium on Operating System Principles, New York*, pages 81-94, October 1991.
- [31] A. L. N. Reddy and J. C. Wyllie. I/O issues in a multimedia system. *IEEE Computer*, 27(3):69-74, March 1994.
- [32] A. Silberschatz and P. Galvin. *Operating System Concepts*. Addison-Wesley, 1994.
- [33] J. A. Stankovic and K. Ramamritham. *Hard Real-Time Systems*. IEEE Computer Society Press, Los Alamitos, California, 1988.
- [34] F. A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID: A disk storage system for video and audio files. In *Proceedings of ACM Multimedia, Anaheim, CA*, pages 393-400, August 1993.
- [35] Virage Inc. Virage Products, <http://www.virage.com/>.
- [36] H. D. Wactlar, T. Kanade, M.A. Smith, and S.M. Stevens. Intelligent access to digital video: Informedia project. *IEEE Computer*, 28(9):46-52, September 1995.
- [37] D. A. White and R. Jain. Similarity indexing with the ss-tree. In *Proceedings of 1996 International Conference on Data Engineering, New Orleans, Louisiana*, pages 516-523, February 1996.
- [38] H.J. Zhang, C.Y. Low, S.W. Smoliar, and J.H. Wu. Video parsing, retrieval and browsing: an integrated and content-based solution. In *Proceedings of ACM Multimedia, San Francisco, CA*, pages 15-24, November 1995.