Multimedia Systems 9: 575–585 (2004)
Digital Object Identifier (DOI) 10.1007/s00530-003-0129-9

Multimedia Systems

VDBMS: A testbed facility for research in video database benchmarking*

Walid Aref[1], Ann Christine Catlin[1], Ahmed Elmagarmid[1], Jianping Fan[2], Moustafa Hammad[1], Ihab Ilyas[1], Mirette Marzouk[1], Sunil Prabhakar[1], Yi-Cheng Tu[1], Xingquan Zhu[3]

[1] Department of Computer Sciences, Purdue University, West Lafayette IN 47907, USA
[2] Department of Computer Science, University of North Carolina, Charlotte, NC 28223, USA
[3] Department of Computer Science, University of Vermont, Burlington, VT 05405, USA

Abstract. Real-world video-based applications require database technology that is capable of storing digital video in the form of video databases and providing content-based video search and retrieval. Methods for handling traditional data storage, query, search, retrieval, and presentation cannot be extended to provide this functionality. The VDBMS research initiative is motivated by the requirements of video-based applications to search and retrieve portions of video data based on content and by the need for testbed facilities to facilitate research in the area of video database management. In this paper we describe the VDBMS video database research platform, a system that supports comprehensive and efficient database management for digital video. Our fundamental concept is to provide a full range of functionality for video as a well-defined abstract database data type, with its own description, parameters, and applicable methods. Research problems that are addressed by VDBMS to support the handling of video data include MPEG7 standard multimedia content representation, algorithms for image-based shot detection, image processing techniques for extracting low-level visual features, a high-dimensional indexing technique to access the high-dimensional feature vectors extracted by image preprocessing, multimedia query processing and optimization, new query operators, real-time stream management, a search-based buffer management policy, and an access control model for selective, content-based access to streaming video. VDBMS also provides an environment for *testing* the correctness and scope of new video processing techniques, *measuring* the performance of algorithms in a standardized way, and *comparing* the performance of different implementations of an algorithm or component. We are currently developing video component wrappers with well-defined interfaces to facilitate the modification or replacement of video processing components. The ultimate goal of the VDBMS project is a flexible, extensible framework that can be used by the research community for developing, testing, and benchmarking video database technologies.

* This work was supported in part by the National Science Foundation under Grants IIS-0093116, EIA-9972883, IIS-9974255, and IIS-0209120 and by the NAVSEA/Naval Surface Warfare Center, Crane.

Correspondence to: A. Catlin (e-mail: acc@cs.purdue.edu)

1 Introduction

A significant and ever increasing portion of the information created today has audiovisual components, and most of it is now available in digital form. Real-world video-based applications require database technology that is capable of storing this information in the form of video databases and providing content-based video search and retrieval. Methods for handling traditional data storage and retrieval cannot be extended to provide this functionality for video. Two standard approaches have been developed for handling video data: storage in video servers and storage as binary large objects. Storing video data in video servers with few data management capabilities has been widely adopted by video-on-demand (VOD) systems. The Fellini VOD system [27] supports real-time and non-real-time storage and retrieval of continuous media data (video, audio.) It focuses primarily on media streaming and provides no support for content search and retrieval of the stored media. Many research VOD systems focus on efficient buffer management of raw video streams [8,22] and provide either very limited support or no support at all for content search and retrieval. It is not possible for such systems to address the integration of content search, retrieval, and streaming. Integration of the search and streaming of video data is, however, supported by some commercial systems. Virage[1] provides textual extraction, encoding, search, and streaming of continuous media. As a specialized video server, Virage offers a great deal of flexibility for applications that access video by textual content. On the other hand, it is difficult to integrate their video functionality with other types of data (e.g., relational data or high-dimensional data) or to express and execute complex declarative queries (e.g., SQL queries) on the stored media.

The second approach views video as a binary large object (BLOB) whose content is hidden from the system and for which no meaningful processing or optimization can be performed. BLOBs are supported by many industrial-strength database systems. The problem with the BLOB representation is that once it leaves the database, it is handled by a user application in an application-specific manner, resulting in data type mismatches between the database and the appli-

[1] `http://www.virage.com`

cation. Moreover, much important functionality, such as online and customized video views, query by content, similarity search queries, video editing functionalities, and data abstraction, cannot be provided as an integral part of a BLOB-based system. Some recent video research has focused on new applications of data management functionality to video data, such as mining videos [29,36] and monitoring/tracking objects in video streams using continuous queries [15]. Such features will benefit from the efficient storage, query, and indexing capabilities of a video database system that supports video data as a first-class database object. This functionality cannot be satisfied by handling video data as BLOBs. The requirements must be addressed by building upon existing database technologies and extending them as needed to efficiently support video database functionality.

VDBMS provides a full range of functionality for video as a well-defined data type, with its own description, parameters, and applicable methods. The development and integration of a video data type into the database management system achieves a clear separation between the video processing and database components. This allows video-based application design to focus on details of the application itself while relying on the underlying video framework components for storage, search, retrieval, analysis, and presentation of the video data. Video applications thus inherit all the powerful functionality generally provided by database management systems, including query processing, optimization, concurrency, and recovery. Furthermore, VDBMS is a general-purpose data management system and supports extensions to include new data processing functionalities such as video mining and continuous queries.

VDBMS system components include a video preprocessing toolkit, a high-dimensional index manager, a stream manager, and a search-based buffer management policy. These VDBMS system components are described in this paper, and details can be found in the literature [2,11,14,18]. We present VDBMS as a research platform because it provides an open and flexible environment for investigating new research areas related to video database management, including the implementation, integration, and evaluation of new and existing algorithms. Research problems that were addressed within the VDBMS environment to support the handling of video data include MPEG7 document compliance for importing and exporting video features [1], algorithms for image-based shot detection [20], image processing techniques for extracting low-level visual features [11], camera motion detection algorithms [35], hierarchical video summarization strategies for abstracting video content, a high-dimensional indexing technique to access the high-dimensional feature vectors extracted by image preprocessing, new multifeature rank-join query operators for image similarity matching [18], a new real-time stream manager to admit, schedule, monitor, and serve concurrent video stream requests periodically, an enhanced buffer management policy that integrates knowledge from the query processor to improve streaming performance [14], and an access control model that provides selective, content-based access to streaming video data [6].

The development of the VDBMS video database management research platform was motivated by the requirements of video-based applications to retrieve portions of video data based on content and by the need for testbed facilities to facilitate research in the area of video database management. While investigating, developing, and testing the fundamental components required supporting full video database functionality, we utilized VDBMS as a testbed for integrating and evaluating video processing technologies from other sources. As such, the system has provided us with an environment for *testing* the correctness and scope of algorithms, *measuring* the performance of algorithms in a standardized way, and *comparing* the performance of different implementations of a component. The next step in VDBMS system development is the construction of video component wrappers with well-defined interfaces that allow video components to be easily modified or replaced. We also plan to provide the corresponding semi-automatic mechanisms for integrating these components into VDBMS. The ultimate goal of the VDBMS project is a flexible, extensible framework that can be used by the research community for developing, testing, and benchmarking video database technologies.

We describe VDBMS system components in Sects. 2 and 3. To demonstrate the usefulness of VDBMS as a testbed for video database benchmarking, Sect. 4 presents experimental studies and analysis for alternative techniques implemented within the VDBMS environment.

2 The query interface

A VDBMS query interface client supports end-user content-based query, search, retrieval, and real-time streaming for the VDBMS video database server. End users can query by image, camera motion type, or keywords or specify an SQL statement. In image-based queries, users present an example image and query the database for images or shots "most similar" to the example based on any number and combination of the listed visual features. The features of the user's query image are extracted online and sent to the server for execution. Results can be either frame level (video frame images with similar visual features) or shot level (video shots with similar aggregate visual features, where feature aggregation is computed across shot frames). Keyword queries are matched against video annotation data associated with logical video scenes. The query interface generates and displays the SQL equivalent of the user's query for all non-SQL queries. The VDBMS query processor returns a ranked list of results, where the user determines the number of top-ranked results to retrieve. Users can navigate an image skim of the results using any step size. When the user requests shot-level results, a key frame representing shot content is returned to the user, and the user can select the key frame to stream the shot directly from the database to the query interface media player.

Users access the VDBMS query interface using the Windows-based client shown in Fig. 1. The client connects to the VDBMS system that resides on a Sun Enterprise 450 machine with four UltraSparc II processors, running the Solaris 5.6 operating system. The system memory is 1 GB, with RAID disks of 170 GB of storage. VDBMS functionality has been tested against more than 500 h of medical videos obtained from the Indiana University School of Medicine. The medical videos are digitized, compressed into MPEG1 format, processed offline by the VDBMS preprocessing toolkit to generate image-based and content-based metadata, and then stored together with their metadata in the VDBMS database.

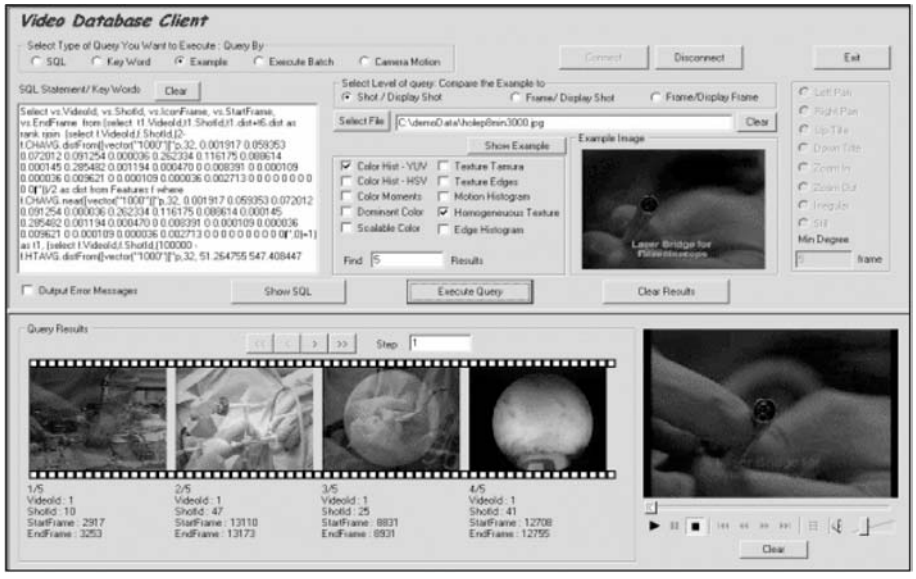

Fig. 1. VDBMS query interface

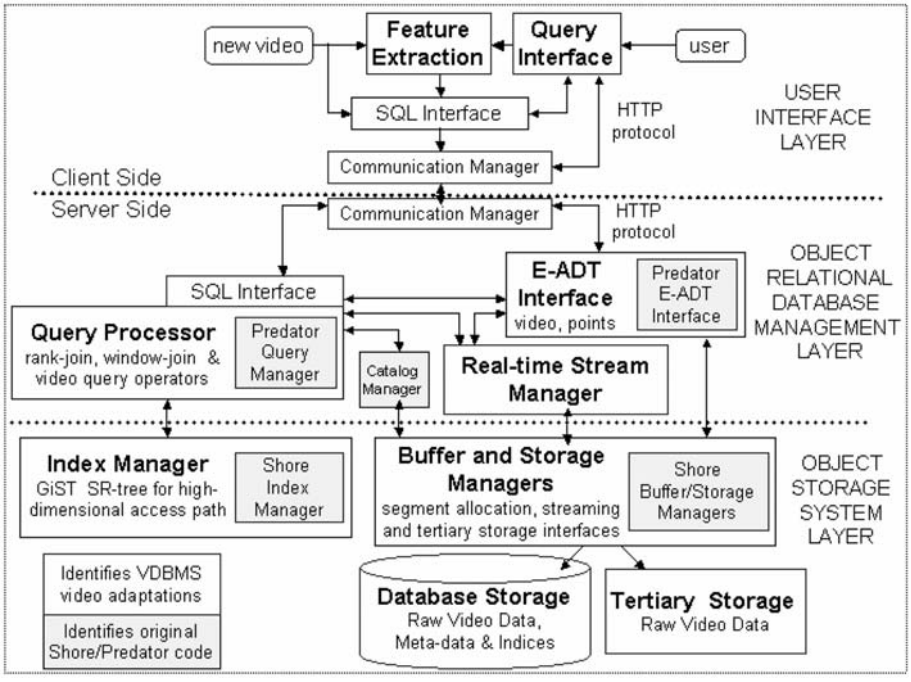

Fig. 2. VDBMS layered system architecture

3 The video database management system

The VDBMS database management system is built on top of an open source system consisting of Shore [33], the storage manager developed at the University of Wisconsin, and Predator [30], the object relational database manager from Cornell University. The VDBMS research group has developed the extensions and adaptations needed to support full database functionality for the video as a fundamental, abstract database data type. Key database extensions include high-dimensional indexing, video store and search operations, new video query types, real-time video streaming, search-based buffer management policies for continuous streaming, and support for extended storage hierarchies including tertiary storage. These extensions required major changes in many traditional database system components. Figure 2 illustrates our layered system architecture with its functional components and their interactions. The system consists of the object storage system layer at the bottom, the object relational database management layer in the middle, and the user interface layer at the top.

3.1 A video preprocessing toolkit

The VDBMS video preprocessing toolkit applies image and semantic processing to partition raw video streams into shots, then associates the shots with extracted visual and semantic descriptors that represent and index the video content for searching. Preprocessing algorithms detect the video scene boundaries that partition the video into meaningful shots using a process that computes color histogram differences and incorporates a mechanism for dynamic threshold determination.

Video shots are then processed to extract MPEG7-compatible low-level visual feature descriptors, camera motion classification, spatial and temporal segmentation, representative key frames, and the semantic annotations of domain experts. The video and its features and indices are stored in the VDBMS database, along with physical metadata such as resolution for quality-of-service presentation. Our system follows the recent trend of representing the video content description in an XML-like format according to MPEG7 [19] multimedia content descriptors. MPEG7 is the worldwide standard for video content description and has been incorporated as an integral part of VDBMS feature representation. VDBMS video preprocessing extracts nearly all low-level features defined by MPEG7, including color histogram in both HSV and YUV formats, texture tamura, texture edges, color moment and layout, motion and edge histograms, dominant and scalable color, and homogeneous texture.

We are currently developing a wrapper that abstracts the extraction, representation, and query of features. This plug-in component allows users to define a new feature, supply its extraction (image processing) algorithm, and query against the feature for image similarity matching. Our wrapper and integration mechanisms incorporate the feature into the query interface, create the schema for database representation, and apply the user-provided algorithm during video preprocessing and image-based queries. We provide users with a graphical interface for defining and integrating video segmentation algorithms, feature extraction algorithms, camera motion classification techniques, and other video processing techniques that can be used for content representation and content-based retrieval. This will allow researchers to compare and evaluate alternative methods, improve existing algorithms, or develop new ones.

3.2 High-dimensional video indexing

Since high-dimensional feature data are collected for each video frame and aggregated for each video shot, the metadata that represent and index video content occupy more disk space than the video itself. The magnitude of these metadata and their storage in the database as high-dimensional vectors present serious indexing and searching difficulties in the execution and optimization of feature-based queries. The VDBMS research group extended the indexing capability of Shore by incorporating the GiST v2.0 implementation [16,17,34] of the SR-tree as the high-dimensional index [4,5,21] and modified the query-processing layer of Predator to access the Shore/GiST index. VDBMS added the *vector* ADT to be used by all feature fields and implemented

CREATE GSR INDEX <table>_<fieldname> <table>

to create an instance of the GiST SR-tree for each field to be used as the access path in feature matching queries. The multi-dimensional indexing structure handles the high-dimensional feature vectors that are produced by visual feature extraction and used in image similarity searches.

We are currently building an interface to support a plug-in component for indexing techniques so that alternative indexing mechanisms can be implemented, tested, and compared within the VDBMS system.

3.3 The query processor and video query operators

The query processor was modified extensively to handle the new high-dimensional indexing scheme as well as to support new video query operators and their integration into the query execution plan. VDBMS query processing must take into account the video methods and operators in generating, optimizing, and executing query plans. Image similarity search is performed by issuing nearest neighbor queries to the high-dimensional access path.

In multifeature image similarity queries, users generally present a sample image and query the database for images "most similar" to the example based on some collection of visual features. Results should be determined according to a combined similarity order [13,24]. We have developed a practical, binary, pipelined query operator, NRA-RJ, that determines an output global ranking from the input ranked video streams based on a score function [18]. Our algorithm extends Fagin's optimal aggregate ranking algorithm [10] by assuming no random access is available on the input streams. The output of NRA-RJ thus serves as valid input to other operators in the query pipeline, supporting a hierarchy of join operations and integrating easily into the query processing engine of any database system.

We created a new VDBMS query operator that encapsulates the rank-join algorithm in its *GetNext* operation. Each call to *GetNext* returns the next top element from the ranked inputs. The internal state information needed by the operator consists of a priority queue of objects encountered thus far, sorted by worst score in descending order. *GetNext* is binary (although this restriction is merely practical), and the algorithm holds for more than two inputs. Our modifications to the original NRA algorithm are the following:

- The right input list is a source stream that provides the operator with the ranked objects and their exact scores. The left input may not be a source list since it can be the output of another NRA-RJ operator. In this case, the score is expressed as a range, from worst to best. This means that *GetNext* must be able to handle a score range rather than an exact score from the left iterator.
- The parameter k, the number of requested output objects, is not known in advance; rather it increases for each call to *GetNext*. The modified algorithm first checks if another object can be reported from the priority queue without violating the stopping condition and, if not, moves deeper into the input streams to retrieve more objects.
- In each call to *GetNext*, the current depth of the caller is passed to the operator. This extra information assures synchronization among the pipeline of NRA-RJ operators.

The incremental and pipelining properties of our aggregation algorithm are essential for practical use in real-world database engines, and our new operator will help in implementing this type of join in ordinary query plans.

A modular interface for the integration of query operators into the VDBMS query processor is currently underway. The interface will support the integration of user-developed operators into the query execution plan and will also support the performance evaluation and comparison of alternative algorithms for implementing query operators by allowing developers to identify performance metrics and test point locations

for collecting measurements and statistics. In Sect. 4.2, we demonstrate this concept in the context of performance analysis for different algorithms that implement the multifeature ranking query operator. The experimental study investigates scalability as well as time and space complexity and discusses performance trade-off issues

3.4 The stream manager

The VDBMS stream manager is responsible for handling the special needs of video streaming. Each request for video data needs to be streamed at a predetermined rate (MPEG1 needs, on average, a 1.5-Mbps display rate). Violating the rate of streaming by either increasing or decreasing the display rate may result in overflow at the client buffer or hiccups at the client side. To hide the latency associated with access to disk storage, the stream manager streams part of the data (a *segment*) while prefetching the next segment into the memory buffers.

Since many stream requests are serviced simultaneously by the manager, resources such as memory buffers and disk bandwidth must be divided among the streams. This is achieved by serving each stream request *periodically* and serving additional concurrent streaming requests within that period. Due to limited memory and disk bandwidth, the manager can only serve a specific number of requests within a single period. To serve requests in real time, the segment referenced next should be retrieved into the buffer before the end of the current period. We have implemented a real-time stream manager above the buffer manager layer in VDBMS [1], and its functionality is as follows:

1. Admit a new stream request if the maximum number of concurrent streams has not been reached; otherwise delay the request and retry when one of the current requests finishes.
2. Schedule segment prefetching by sending requests to the buffer manager. Each page of the allocated segment is fixed in the buffer pool until the page is streamed.
3. Send the segment to the client according to a predetermined streaming rate. The segments are processed pagewise, and each page is unfixed and returned to the buffer manager after streaming the content.
4. Communicate with the query manager to keep track of search results. The stream manager is implemented as multithreaded modules and has well-defined interfaces with the query engine, the buffer manager, and the Extensible Abstract Data Type (E-ADT) interface. The stream manager operates by issuing requests to the buffer manager, guiding the underlying buffer management policies, communicating with the query processor, and sending streams to clients at a specific rate.

3.5 A search-based buffer management policy

Continuous-media servers that support content-based search and retrieval use a main memory buffer to store the requested media streams before sending them on to the user. Buffering policies for media streaming have been investigated in several studies. Chang and Garcia-Molina [8] introduce a memory-efficient prefetching schedule based on fixing the time displacement between prefetching requests. A recent study [22] proposes dynamic buffer allocation for media streaming that minimizes the memory requirement for concurrent media streams. The work in [12] presents the basic functionalities of buffer management for delay-sensitive multimedia data, and in [28] Ozden et al. describe changes needed by database management systems to support multimedia data. Replacement policies for media streams have been studied for target applications such as VOD [9,23,26], which are designed for streaming purposes only. Brown et al. [7] propose a goal-oriented buffer allocation for different database workloads, where a target goal (average execution time, for instance) is designated for each workload. In the VDBMS project, we have investigated a new buffer management policy that addresses the relationship between the searching and streaming processes of video data.

Caching parts of media streams that may be referenced in the near future enhances streaming performance in two ways: it reduces the number of references to disk storage and it minimizes delay associated with the start of streaming. However, precise caching decisions are often difficult to make. Optimal prefetch and replacement policies would prefetch the data before their first reference and replace the data block that will not be referenced for the longest time [28]. An obvious difficulty is the policy's dependence on knowledge about expected streams, which is generally not available. In the case of video streaming, however, there is an inherent connection between query processing and streaming. Choices for streaming are usually based on query results, and this relationship can be used by the buffer manager to prefetch and cache pages expected for reference.

We have developed an efficient buffer management policy that uses feedback from the search engine to make more accurate replacement and prefetching decisions. Top-ranked query results from the query processor are used to predict future video streaming requests, and a weight function [3] determines candidates for caching. By integrating knowledge from the query and streaming components, VDBMS can achieve better caching of media streams, thus minimizing initial latency and reducing disk I/O. Many factors must be considered when basing prefetching or replacement decisions on search results. Streaming based on the search context is probabilistic: new streaming requests can be based on any of the search results or even on none of them. Also, since the caching space is now shared by pages for current as well as expected streams, there will be increased overhead in the replacement policy associated with balancing the space assigned to each.

In our search-based replacement policy, pages in the buffer pool that are referenced by either current or expected streams are considered for caching. To maximize the number of caching pages, we replace the page that will be referenced by current or expected streams after a long period of time has passed. Also, we prefer caching pages that will be referenced by current streams to those that will be referenced by expected streams by assigning higher *keep* weight values [31] to the current streams. Lookup tables contain pointers to expected streams, which are collected from the search results and checked by the stream manager for matches when determining pages to replace. With knowledge collected in lookup tables for expected streams, we predict with high probability

that one of the expected streams will be requested. The stream manager tracks the utilization of the streaming period and utilizes any fraction of the streaming period unused by current streams to prefetch the first segment of the top-ranked expected streams into the memory buffer. If an expected stream becomes an actual request, most of the pages in the first segment would already be cached in the buffer pool, and as a result the number of references to lower-level storage would be significantly reduced. The prefetching policy does not introduce much overhead since it operates only during idle period time, utilizing unused and reserved streaming resources.

The performance of the search-based policy was evaluated by investigating the effects of buffer management on the number of I/Os when referencing the first segment of a requested stream. Experimental results are presented in Sect. 4.1. They show that initial latency of the search-based policy is reduced on average by 20% when compared with traditional policies.

3.6 Extended storage hierarchies

Video database storage and buffer managers handle huge volumes of data with real-time constraints [23,26]. In VDBMS, the buffer pools are divided between the database buffer area and the streaming area where requests for streams are serviced. Extended buffer management handles multiple page requests with segment allocation (instead of the traditional page-based approach) for the large streaming requests from the stream manager. An interface between the buffer manager and the stream manager is used to exchange information that guides buffer caching for stream requests. The storage manager was extended to perform necessary video operations and process both real-time and non-real-time requests. VDBMS methods for handling extended storage hierarchies support transparent, real-time access to buffer, disk, and tertiary storage. Different caching levels on buffer and disk storage enhance access for frequently referenced data, and a tertiary storage server manages access to tertiary resident data, making them directly accessible to the VDBMS system.

The tertiary storage and cache disk managers are implemented underneath the storage volume manager. A dedicated disk partition is used for caching hot items in tertiary storage, and the cache disk manager maintains and reports these items. The tertiary storage manager communicates with a PowerFile manager (remote NT process), which locates the requested page in its changer, loads the CD/DVD, and sends the requested data through the local network via TCP. Due to the overhead associated with reading from the device, the basic transfer unit is a block instead of a page. The block is copied to the cache disk, and the first page is sent to the buffer pool. DVD jukeboxes can be daisy-chained, giving VDBMS access to terabytes of data.

4 A testbed for video database benchmarking

While investigating and implementing components to support full video database management, we have utilized VDBMS to investigate, integrate, validate, compare, and evaluate alternate video processing techniques and technologies. To illustrate the effectiveness of the current VDBMS system for new component integration, validation, and performance evaluation, we briefly describe three recent research projects carried out within the VDBMS environment. The contribution of these and other experimental studies to the understanding of video processing within the database environment is the motivation for our effort to create a complete testbed facility for video database benchmarking.

4.1 Validation of a buffer management policy

To validate the search-based buffer management policy in a heavy workload environment, we execute 32 simultaneous clients. Each client submits an image-based query to VDBMS and receives a collection of key frames representing the results of a shot-based image similarity search. The client delays for a random period (uniformly distributed between 10 and 20 s) after retrieving the results and then submits a streaming request for one of them. We assume the client plays a shot selected from the four top-ranked results 80% of the time. The VDBMS stream manager admits the streaming request if possible; otherwise the request is delayed until one of the current streams has finished. The client immediately submits a new search request following the streaming of the selected shot, so that a heavy load situation is maintained. The search results are synthesized by random selection of ten candidate shots from the database. The random selection provides an upper bound for the performance of our policy. Our *keep* weight is set to 3 if a page is referenced by an expected stream from the top-ranked results and 4 if the page is referenced by a current stream. Higher values for the keep parameter are not recommended since they lead to excessive looping over buffer pages to find replacement candidates. The experimental data consist of eight 1-h videos, compressed in MPEG1 format with a total size of 5 GB. Each video has been preprocessed into shots with lengths between 5 and 10 min. We set the page size to 8 KB, the segment size to 30 pages and the maximum number of concurrent streams to 16. Each experimental run lasts for 30 min, and the total number of buffer references is approximately 500,000.

We compare the performance of the following policies:

- Search-based replacement (SrchBR): pages are cached if referenced by current or expected stream requests.
- Search-based prefetching and replacement (SrchBPR): first segment of expected stream is prefetched; pages are cached if referenced by current or expected stream requests.
- Stream-based replacement (StrmBR): pages are cached only if referenced by a concurrent stream request.
- Use&Toss: pages are candidates for replacement immediately after use [32].

Figure 3a shows the effect of the buffer policies on reducing the number of I/Os when referencing the first segment of the stream. For each first segment, we measure the percentage of pages found in the buffer as we increase the buffer size from 10 to 25 MB. The figure shows that SrchBPR caches about 25% of the total pages of new streams based on the search results; that is, the initial latency is reduced by 25%. Although SrchBR achieves better results than StrmBR and Use&Toss, it

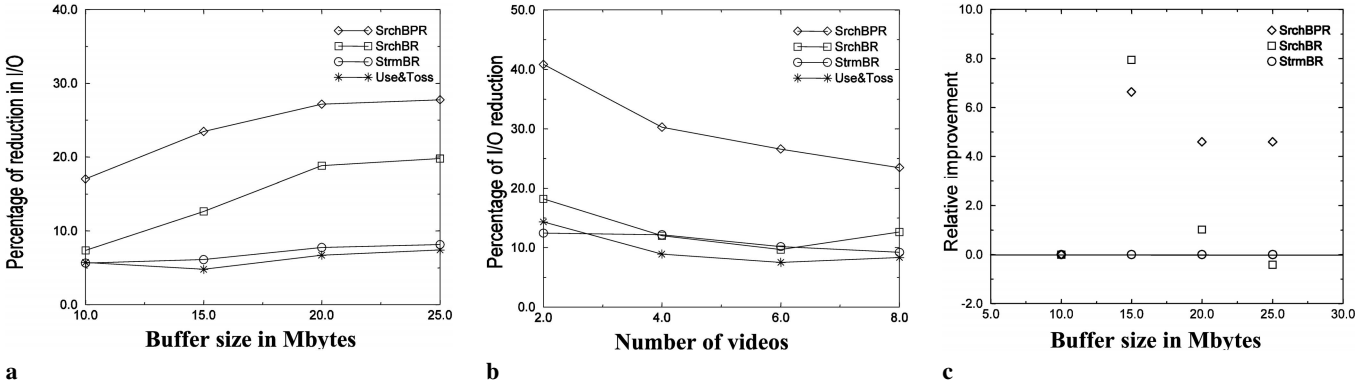

Fig. 3a–c. Performance evaluation of buffer management policies. **a** Reduction in I/O as buffer size changes. **b** Reduction in I/O as number of videos changes. **c** Relative improvement in buffer hit ratio as buffer size changes

caches only those pages either used by current streams or referenced by expected streams, and therefore the improvement is smaller than that of the prefetch policy. StrmBR has no knowledge of expected streams and performs about the same as Use&Toss. In Fig. 3b, the buffer size is fixed at 25 MB, and we measure the reduction in I/O when referencing the first segment of the stream as the number of stored videos is increased from two to eight. SrchBPR achieves the best performance, as high as 40% reduction in the number of I/Os. This improvement results from both prefetching and replacement strategies since more common data now exist between current and expected streams. As the number of videos increases, the chance for interaction decreases. Thus the improvement is dominated by the positive effects of prefetching. The effect of the replacement policy is obvious in SrchBR and StrmBR, as both reduce the I/Os with small data sets. With larger data sets, StrmBR and Use&Toss contribute similarly to the reduction of I/O since neither has any knowledge about expected streams. The short duration of streamed segments represents an obstacle for replacement algorithms that depend only on current streams for two reasons: (1) in large data sets with uniform access patterns, common pages are infrequent, and (2) common pages generally exist within a short interval of each other (intervals are bounded, on average, by half the length of a shot). Replacement policies based on caching common pages between current streams will thus have a small number of pages to recommend for caching.

Figure 3c shows the relative improvement in the buffer hit ratio for policies based on current streams. As the buffer size increases, more space is available to cache the data and the chance of replacement is decreased. With small buffer sizes, pages are replaced more frequently and the improvement achieved with search-based policies such as SrchBPR and StrmBR becomes significant.

4.2 Performance evaluation of rank-join query operators

We implemented three state-of-the-art rank-join algorithms as query operators in VDBMS for an extensive empirical study to evaluate operator performance and trade-off issues in executing multifeature queries. Our experimental study compares the NRA-RJ operator developed by the VDBMS research group [18], the J* operator introduced by Natsev et al. [25], and (for a baseline comparison) the nonpipelined version of the NRA algorithm as a multiway rank-join operator, MW-RJ [10]. Although most query optimizers are restricted to binary operators, MW-RJ provides a reference line for the best possible performance. We investigated scalability as well as time and space complexity between the algorithms for executing a join of multiple ranked inputs (any number and combination of features) on the stored video objects.

The following multifeature query for the k top-ranked results was issued against the VDBMS features:

Retrieve the top k video shots "most similar" to a given image based on m visual features.

The query evaluation plan has m nearest neighbor (NN) operators on m different visual features, and $m-1$ rank-join binary operators are used, where the results of one operator are pipelined to the next operator in the pipeline. The number of features m in our study varies from 2 to 6, and the number of top-ranked results k varies from 5 to 100. To evaluate the operators, we used the following performance metrics: (1) query running time for retrieving the top matching k output results, (2) size of the buffer maintained by the operator, and (3) number of database accesses in disk pages. While the number of database accesses should give a good indication of the time complexity of the operator, the experiments show a significant CPU time complexity difference between the two operators that affects the total running time, especially for small numbers of inputs.

Figures 4 and 5 give performance comparisons for NRA-RJ, J*, and MW-RJ, for $m=2$ and $m=3$, respectively, where m is the number of input sources that give a pipeline of length $m-1$. For $m=2$, NRA-RJ is identical to MW-RJ since there is no pipeline. Figure 4a compares the total running time of the NRA-RJ and J* operators. The J* algorithm has a significant CPU overhead due to the execution of its underlying A* graph search algorithm, which considers more join combinations. Thus, NRA-RJ shows a faster execution time. Both operators are nearly equal in the database access count depicted in Fig. 4c. NRA-RJ has a smaller maximum queue size than that of J*, as shown in Fig. 4b, and the difference increases as k increases (i.e., as more results are requested). The difference in the maximum queue size and in the execution time can be explained by the fact that the J* algorithm has to consider more join combinations than NRA-RJ since

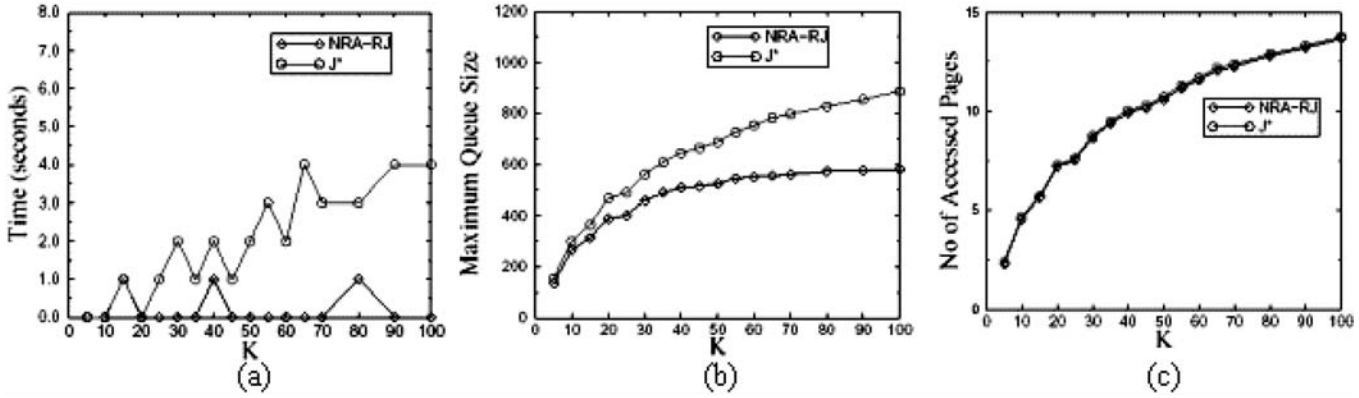

Fig. 4. Performance comparison for NRA-RJ and J* when $m = 2$

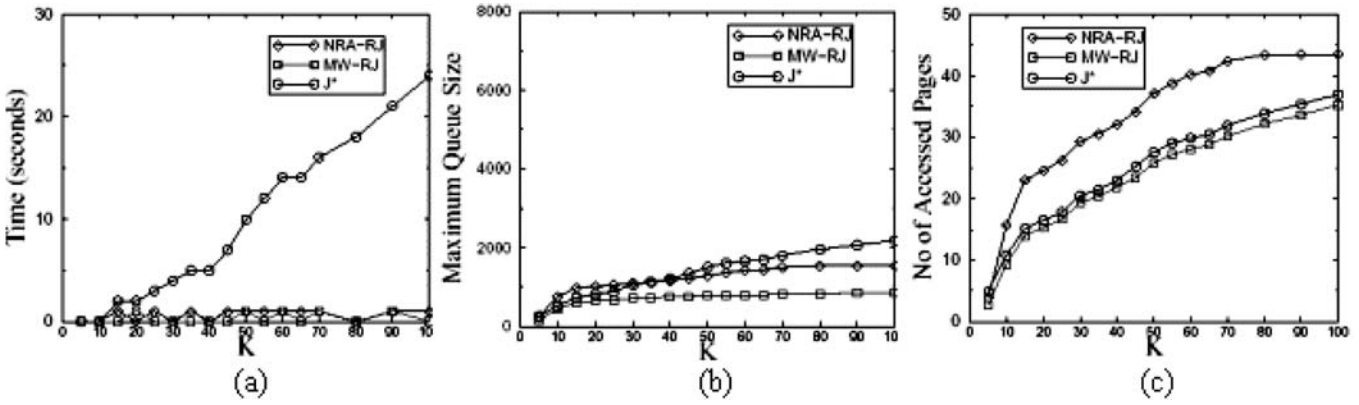

Fig. 5. Performance comparison for NRA-RJ, J*, and MW-RJ for $m = 3$

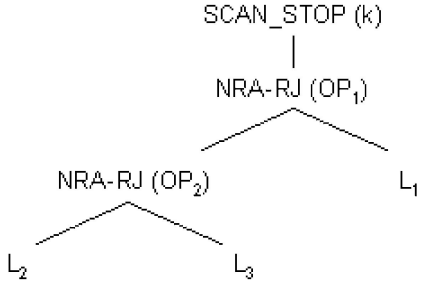

Fig. 6. A query pipeline with $m = 3$

it was developed for a general join condition. When used in self-join problem settings, the generality of the J* algorithm causes expensive unnecessary computations that increase both the queue size and the running time.

Figure 5 compares the NRA-RJ, J*, and MW-RJ operators for $m = 3$. Figure 5a shows that NRA-RJ still outperforms J* in total running time, and the pipeline does not affect the speed of the NRA-RJ operator when compared with MW-RJ. For the maximum queue size given in Fig. 5b and the number of database accesses given in Fig. 5c, we make the following observations:

- NRA-RJ has a larger maximum queue size and more database accesses than MW-RJ. To understand this difference, we clarify how NRA-RJ operates in a pipeline of three inputs. Figure 6 shows NRA-RJ with three input streams, L_1, L_2, and L_3. When the top NRA-RJ operator, OP_1, is called to produce the next top-ranked object, several *GetNext* calls for the left and right children are invoked. According to NRA-RJ's *GetNext* algorithm, OP_1 gets the next tuple from its left and right children at each step. Hence, OP_2 will be required to deliver as many top-ranked objects for L_2 and L_3 as for L_1. These calls to the ranking algorithm in OP_2 force L_2 and L_3 to retrieve unnecessary objects, which results in larger queue sizes with more database accesses. We refer to this as the *local ranking* problem, that is, the NRA-RJ operator in the early pipeline stages tends to retrieve more database objects in order to deliver as many ranked tuples as required by the next NRA-RJ operator.
- The J* operator has less database access cost than NRA-RJ and is close to the cost of MW-RJ, despite NRA-RJ's *local ranking* problem. In contrast to NRA-RJ, the J* algorithm does not retrieve equal numbers of objects from its left and right children.
- For the same reason that J* has fewer disk accesses than NRA-RJ, J* starts with smaller maximum queue size than NRA-RJ. However, as in the case for $m = 2$, J* begins to save many candidate join combinations in the queue, causing its maximum queue size to become larger than that of NRA-RJ as k increases. This also explains the fact that J* has a larger queue size than MW-RJ, even though both are retrieving almost the same number of database objects, as shown in Fig. 5c.

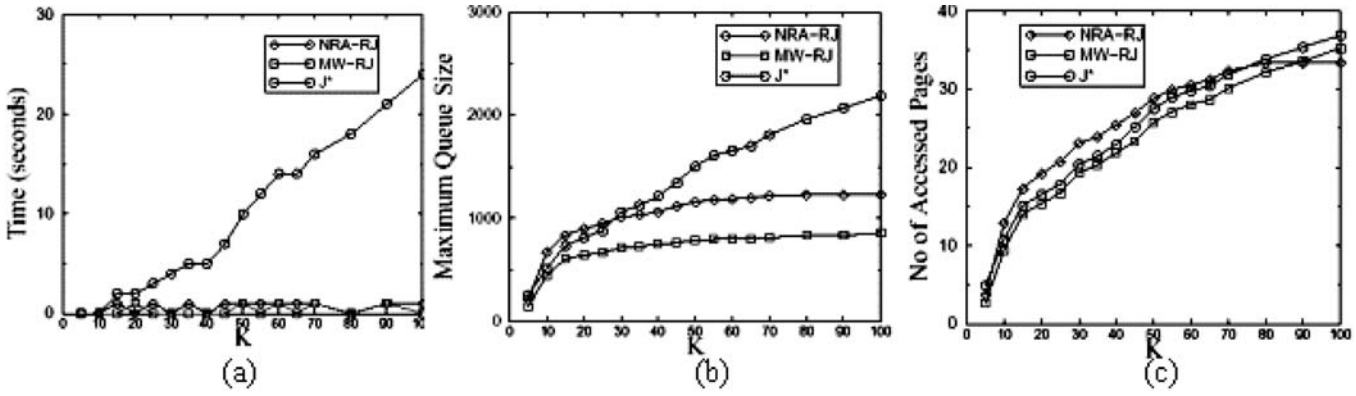

Fig. 7. Optimized NRA-RJ operator

We now evaluate the scalability of the two pipelined operators with respect to the length of the query pipeline m. By fixing $k = 20$, the operators NRA-RJ and J* are again compared with respect to our three chosen performance metrics. As m increases from 2 to 6, NRA-RJ has a larger queue size because of the increased local ranking overhead in the pipeline. As NRA-RJ encounters greater database access, I/O cost begins to dominate total running time. The overhead finally affects the running time enough to make NRA-RJ's performance worse than J*'s, demonstrating clearly that J* is scalable in terms of increased ranked inputs while NRA-RJ is not.

Our evaluation of the performance of NRA-RJ led to an important insight: we must minimize the excessive local ranking calls in earlier stages of the pipeline. Our solution was to unbalance the depth step in the operator children. We changed the NRA-RJ *GetNext* algorithm to reduce the local ranking overhead by changing the way it retrieves tuples from its children, that is, to require less expensive *GetNext* calls to the left child, which is also an NRA-RJ operator. Using different depths in the input streams had a major effect on performance. Figure 7 shows the comparison between the modified NRA-RJ, the J*, and the MW-RJ operators. In the optimized version of the NRA-RJ operator, one tuple is retrieved from the left NRA-RJ child for each p tuple retrieved from the right input child (in the figure, $p = 2$.) The optimized NRA-RJ operator showed significant performance improvements in both the maximum queue size and in the number of database accesses due to the reduction of local ranking overhead in the inner pipeline stages. With this improvement, the optimized NRA-RJ operator is superior to the J* operator, even for large m. The optimized NRA-RJ operator is an order of magnitude faster, has fewer space requirements, and has a comparable number of disk accesses.

4.3 A component for MPEG7 document compliance

The existing implementation for VDBMS feature extraction and database representation applies MPEG7 to descriptors and description schemes, using an XML-like format to define the semantic and image-based information that identifies video content. The video preprocessing toolkit extracts nearly all low-level features defined by MPEG7 as standard, and the VDBMS query interface allows users to retrieve video shots based on any combination of these features. Queries combining multiple low-level features can be used to approximate high-level content-based searches.

To accommodate the representation and query of MPEG7 features not currently extracted by VDBMS video preprocessing, we have developed an XML wrapper that can import any MPEG7 document specified with Data Definition Language (DDL) and map its descriptors to the VDBMS object relational database schema. The wrapper also supports the export of VDBMS extracted features and other metadata from the database as an MPEG7 document. The XML wrapper enables the VDBMS system to make use of any available preextracted metadata formatted as MPEG7 documents without preprocessing the video itself. In addition, features that VDBMS video preprocessing does not extract (such as event-based and other semantic features) can be integrated, represented, and queried as VDBMS metadata via this mechanism. A document import function takes as input a user-supplied MPEG7 document that is generated using multimedia description schemes (MMDS) and contains the high-level and low-level feature descriptors. The document is passed through the VDBMS MPEG7 wrapper to extract, parse, and map the descriptors to the VDBMS database feature schema. The video and its documented MPEG7 features are then stored inside the database, where they can be used for image-based and content-based queries. An export function extracts existing feature descriptors from the database and sends them through the wrapper, where they are mapped to the MPEG7 descriptors. The generated document can be used by other video processing tools or databases.

5 Conclusion

In this paper, we present a video database research initiative that resulted in the successful development of a video database management system that provides comprehensive and efficient capabilities for indexing, storing, querying, searching, and streaming video data. Our fundamental concept was to support a full range of functionality for video as a fundamental, well-defined abstract database data type. Research problems that were addressed by VDBMS to support the handling of video data include MPEG7 standard multimedia content representation, algorithms for image-based shot detection, image processing techniques for extracting low-level visual features, a high-dimensional indexing technique to access feature vec-

tors extracted by image preprocessing, multimedia query processing and optimization, new query operators, a real-time stream manager, a search-based buffer management policy, and an access control model for selective, content-based access to streaming video data. We have also used VDBMS as a testbed for integrating and evaluating video processing techniques and components. As such, the system has provided us with an environment for testing the correctness and scope of algorithms, measuring the performance of algorithms in a standardized way, and comparing the performance of different implementations of components. The use of VDBMS as a testbed facility was illustrated by performance studies to investigate and analyze alternative implementations of video database processing methods.

We are currently constructing video component wrappers with well-defined interfaces to facilitate the modification or replacement of video processing components. We are also developing semiautomatic mechanisms for integrating these components into VDBMS. The ultimate goal of the VDBMS project is a flexible, extensible framework that can be used by the research community for developing, testing, and benchmarking video database technologies.

References

1. Aref W, Catlin AC, Elmagarmid A, Fan J, Hammad M, Ilyas I, Marzouk M, Zhu X (2002) A video database management system for advancing video database research. In: Proceedings of the international workshop on management information systems, Tempe, AZ, November 2002, pp 8–17
2. Aref W, Catlin AC, Elmagarmid A, Fan J, Guo J, Hammad M, Ilyas I, Marzouk M, Prabhakar S, Rezgui A, Teoh S, Terzi E, Tu Y, Vakali A, Zhu X (2002) A distributed database server for continuous media. In: Proceedings of the 18th international conference on data engineering, San Jose, CA, 26 February–1 March 2002, pp 490–491
3. Aref W, Kamel I, Ghandeharizadeh S (2001) Disk scheduling in video editing systems. IEEE Trans Knowl Data Eng 13(6):933–950
4. Beckmann N, Kriegel H, Schneider R, Seeger B (1990) The R*-tree: an efficient robust access method for points and rectangles. SIGMOD Rec ACM Special Interest Group Manage Data 19(2):322–331
5. Berchtold S, Böhm C, Jagadish H, Kriegel H-P, Sander J (2000) Independent quantization: an index compression technique for high-dimensional data spaces. In: Proceedings of the 16th international conference on data engineering, San Diego, February 2000, pp 577–588
6. Bertino E, Hammad H, Aref W, Elmagarmid A (2000) An access control model for video database systems. In: Proceedings of the 9th international conference on information and knowledge management, McLean, VA, November 2000, pp 336–343
7. Brown K, Carey M, Livny M (1996) Goal-oriented buffer management revisited. In: Proceedings of the 1996 ACM SIGMOD international conference on management of data, Montreal, 4–6 June 1996, pp 353–364
8. Chang E, Garcia-Molina H (1997) Effective memory use in a media server. In: Proceedings of the 23rd international conference on very large data bases, Athens, Greece, 25–29 August 1997, pp 496–505
9. Dan A, Sitaram D (1996) A generalized interval caching policy for mixed interactive and long video environments. In: Proceedings of the IS&T SPIE multimedia computing and networking conference, San Jose, CA, January 1996
10. Fagin R, Lotem A, Naor M (2001) Optimal aggregation algorithms for middleware. In: Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, Santa Barbara, CA, May 2001, pp 102–113
11. Fan J, Aref W, Elmagarmid A, Hacid M-S, Marzouk M, Zhu X (2001) Multiview: multi-level video content representation and retrieval. J Electric Imag 10(4):895–908
12. Gemmell J, Christodoulakis S (1992) Principles of delay sensitive multimedia data storage and retrieval. In: ACM Trans Inf Sys 1(1):51–90
13. Guntzer U, Balke W-T, Kiessling W (2000) Optimizing multifeature queries for image databases. In: Proceedings of the 26th international conference on very large databases. Cairo, Egypt, 10–14 September 2000, pp 419–428
14. Hammad M, Aref W, Elmagarmid A (2002) Search-based buffer management policies for streaming in continuous media. In: Proceedings of the IEEE international conference on multimedia and expo, Lausanne, Switzerland, 26–29 August 2002
15. Hammad M, Aref W, Elmagarmid A (2003) Stream window join: tracking moving objects in sensor network databases. In: Proceedings of the 15th international conference on scientific and statistical database management, Boston, June 2003, pp 75–84
16. Hellerstein J, Naughton J, Pfeffer A (1995) Generalized search trees for database systems. In: Proceedings of the 21st international conference on very large data bases, Zurich, Switzerland, 11–15 September 1995, pp 562–573
17. Ilyas I, Aref W (2001) SP-GiST: an extensible database index for supporting space partitioning trees. J Intell Sys 17(2–3):215–235
18. Ilyas I, Aref W, Elmagarmid A (2002) Joining ranked inputs in practice. In: Proceedings of the 28th international conference on very large data bases, Hong Kong, China, 20–23 August 2002, pp 950–961
19. ISO/IEC/JTC1/SC29/WG11 (2001) Text of ISO/IEC 15938-3 Multimedia Content Description Interface, Part 3: Visual final committee draft document no N4062. Singapore, March 2001
20. Jiang H, Helal A, Elmagarmid A, Joshi A (1998) Scene change detection for video database systems. J Multimedia Sys 6(2):186–195
21. Katayama N, Satoh S (1997) The SR-tree: an index structure for high dimensional nearest neighbor queries. SIGMOD Rec ACM Special Interest Group Manage Data 26(2):69–380
22. Lee S-L, Whang K-Y, Moon Y-S, Song I-Y (2001) Dynamic buffer allocation in video-on-demand systems. In: Proceedings of the ACM SIGMOD international conference on management of data, Santa Barbara, CA, 21–24 May 2001
23. Moser F, Kraiss A, Klas WL (1995) A buffer management strategy for interactive continuous data flows in a multimedia dbms. In: Proceedings of the 21st international conference on very large data bases, Zurich, Switzerland, 11–15 September 1995, pp 275–286
24. Nepal S, Ramakrishna M (1999) Query processing issues in image (multimedia) databases. In: Proceedings of the 15th international conference on data engineering, Sydney, Australia, 23–26 March 1999, pp 22–29
25. Natsev A, Chang Y-C, Smith J, Li C-S, Vitter J (2001) Supporting incremental join queries on ranked inputs. In: Proceedings of the 27th international conference on very large data bases, Rome, August 2001, pp 281–290
26. Ozden B, Rastogi R, Silberschatz A (1996) Buffer replacement algorithms for multimedia storage systems. In: Proceedings of

the IEEE international conference on multimedia computing and systems, Hiroshima, Japan, June 1996, pp 172–180
27. Ozden B, Biliris A, Rastogi R, Silberschatz A (1994) Fellini: a low-cost storage server for movie on demand databases. In: Proceedings of the 20th international conference on very large data bases, Santiago, Chile, September 1994
28. Ozden B, Rastogi R, Silberschatz A (1997) Multimedia support for databases. In: Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, Tucson, AZ, 12–14 May 1997, pp 1–11
29. Pan J-Y, Faloutsos C (2002) GeoPlot: spatial data mining on video libraries. In: Proceedings of the international conference on information and knowledge management, McLean, VA, 4–9 November 2002
30. Seshadri P (1998) Predator: a resource for database research. SIGMOD Rec 27(1):16–20
31. Smith J (1978) Sequentiality and prefetching in database systems. ACM Trans Database Sys 3(3):223–247
32. Stonebraker M (1981) Operating system support for database management. Commun ACM 24(7):412–418
33. Storage Manager Architecture (1999) Shore Documentation, Computer Sciences Department, University of Wisconsin-Madison, June 1999
34. Thomas M, Carson C, Hellerstein J (2000) Creating a customized access method for blobworld. In: Proceedings of the 16th international conference on data engineering, San Diego, March 2000, p 82
35. Zhu X, Elmagarmid A, Xiangyang X, Catlin A (2003) InsightVideo: toward hierarchical content organization for efficient video browsing summarization and retrieval. IEEE Trans Multimedia J (in press)
36. Zhu X, Fan J, Aref W, Catlin AC, Elmagarmid A (2003) Medical video mining for efficient database indexing management and access. In: Proceedings of the 19th international conference on data engineering, Bangalore, India, 5–8 March 2003, pp 569–580