VDBMS: A Testbed Facility for Research in Video Database Benchmarking

Walid Aref, Ann Catlin, Ahmed Elmagarmid, Jianping Fan, Moustafa Hammad, Ihab Ilyas, Mirette Marzouk, Sunil Prabhakar, Yi-Cheng Tu, Xingquan Zhu
http://www.cs.purdue.edu/icds

VDBMS research[1] is motivated by the requirements of video-based applications to search by content, and by the need for testbed facilities for research in video database management. Our fundamental concept is to provide a full range of functionality for video as a well-defined abstract data type. The research issues addressed include: MPEG7 for multimedia content representation, techniques for image processing, high-dimensional indexing, multimedia query processing and optimization, new query operators, real-time stream management, access control models for streaming, and search-based buffer management. VDBMS also provides an environment for testing and comparing algorithms in a standardized way. We are currently developing component wrappers with well-defined interfaces to facilitate the modification or replacement of components. Our ultimate goal is a flexible, extensible framework that can be used for developing, testing and benchmarking video database technologies.

1.0 Introduction

A significant and ever increasing portion of the information created today has audio-visual 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, and current approaches for handling video (stored in video servers or as Binary Large OBjects) hide the video data from the database system so that meaningful processing and optimization is not possible. Important functionality such as online customized video views, content-based queries, video content control during streaming and data abstraction cannot easily be supported. The development of the VDBMS video database management research platform is 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. 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.

VDBMS system components include a video pre-processing 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 [1,2,8,10,13]. 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 [7,8], image processing techniques for extracting low-level visual features [8], hierarchical video summarization strategies for abstracting video content, a high-dimensional indexing technique to access the high dimensional feature vectors extracted by image pre-processing, new multi-feature rank-join query operators for image similarity matching [13], a real-time stream manager to admit, schedule, monitor and serve concurrent video stream requests, an enhanced buffer management policy that integrates knowledge from the query processor to improve streaming performance [10], and an access control model that provides selective, content-based access to streaming video data [5].

While investigating, developing, and testing the fundamental components required to support full video database functionality, we also 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

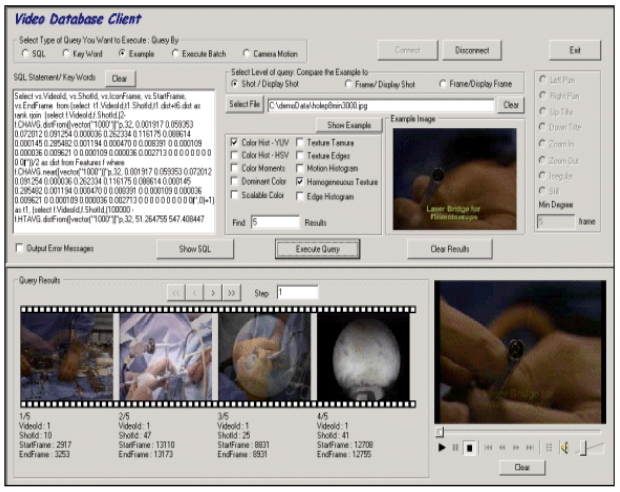
[1] This work was supported in part by the National Science Foundation under Grants IIS-0093116, EIA-9972883, IIS-9974255, and IIS-0209120

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 selected VDBMS system components in Sections 2 and 3. To demonstrate the usefulness of VDBMS as a testbed for video database benchmarking, Section 4 presents experimental studies for alternative techniques implemented within the VDBMS environment.

2.0 The Query Interface

A VDBMS query interface client supports content-based query, search, retrieval and real-time streaming for the VDBMS video database server. End-users can query by image, camera motion, or keywords. 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 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 frames with similar features) or shot level (video shots with similar aggregate features, where aggregation is computed across shot frames.) The VDBMS query processor returns a ranked list of results, and users can navigate an image skim of the results. 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.



```
Select vs.VideoId, vs.ShotId, vs.IconFrame, vs.StartFrame,
vs.EndFrame from (select t1.VideoId,t1.ShotId,t1.dist+t6.dist as
rank rjoin (select f.VideoId,f.ShotId,(2-
f.CHAVG.distFrom([vector("1000")["p.32, 0.001917 0.059353
0.072012 0.091254 0.000036 0.262334 0.116175 0.088614
0.000145 0.285482 0.001194 0.000470 0 0.008391 0 0.000109
0.000036 0.009621 0 0.000109 0.000036 0.002713 0 0 0 0 0 0 0 0
0 0]")]/2 as dist from Features f where
f.CHAVG.near([vector("1000")["p.32, 0.001917 0.059353 0.072012
0.091254 0.000036 0.262334 0.116175 0.088614 0.000145
0.285482 0.001194 0.000470 0 0.008391 0 0.000109 0.000036
0.009621 0 0.000109 0.000036 0.002713 0 0 0 0 0 0 0 0 0 0]",0)=1)
as t1, (select f.VideoId,f.ShotId,(100000 -
f.HTAVG.distFrom([vector("1000")["p.32, 51.264755 547.408447
```

```
1/5                 2/5                 3/5                 4/5
VideoId : 1         VideoId : 1         VideoId : 1         VideoId : 1
ShotId : 10         ShotId : 47         ShotId : 25         ShotId : 41
StartFrame : 2917   StartFrame : 13110  StartFrame : 8831   StartFrame : 12708
EndFrame : 3253     EndFrame : 13173    EndFrame : 8931     EndFrame : 12755
```

Figure 1. VDBMS query interface.

Users access the VDBMS query interface using the Windows-based client shown in Figure 1. The client connects to the VDBMS system which resides on a Sun Enterprise 450 machine with 4 UltraSparc II processors. VDBMS functionality has been tested against more than 500 hours of medical videos obtained from the Indiana University School of Medicine. The medical videos are digitized, compressed into MPEG1 format, processed off-line by the VDBMS pre-processing toolkit to generate image and content-based meta-data, and then stored together with their meta-data in the VDBMS database.

3.0 The Video Database Management System

The VDBMS database management system is built on top of an open source system consisting of Shore [23], the storage manager developed at the University of Wisconsin, and Predator [20], 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.

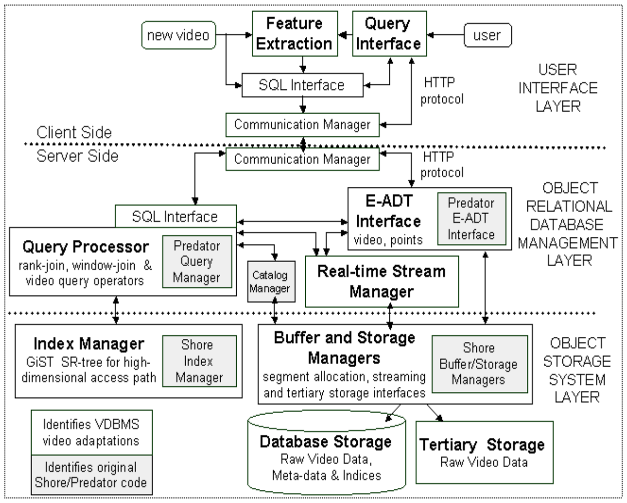


Figure 2. VDBMS layered system architecture.

3.1 A Video Pre-processing 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 [7]. Video shots are then processed to extract MPEG7 compatible low-level visual feature descriptors [2,7,8], spatial and temporal segmentation, representative key frames, and the semantic annotations of domain experts. The video, its features and indices are stored in the VDBMS database. Our system follows the recent trend of representing content description in an XML-like format according to MPEG7 [14] multimedia content descriptors. MPEG7 is the worldwide standard for video content description, and VDBMS video pre-processing extracts nearly all low-level features defined by MPEG7, including color histogram (HSV,YUV), 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 pre-processing and image-based queries. This will allow researchers to compare and evaluate alternate methods, improve exiting algorithms or develop new ones.

3.2 High-dimensional Video Indexing

Since high-dimensional feature data is collected for each video frame and aggregated for each video shot, the meta-data that represents and indexes video content occupies more disk space than the video itself. The magnitude of this meta-data and its storage as high-dimensional vectors present serious indexing and searching difficulties in the execution and optimization of feature-based queries. VDBMS extended the indexing capability of Shore by incorporating the GiST v2.0 implementation [11,24] of the SR-tree as the high-dimensional index [4,12,15] 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 creates an instance of the GiST SR-tree for each field to be used as the access path in feature matching queries.

3.3 The Query Processor

The query processor handles the new high-dimensional indexing scheme, supports new video query operators, and takes 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 multi-feature image similarity queries, users 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 [9,17]. We have developed a practical, binary, pipelined query operator, NRA-RJ, which determines an output global ranking from the input ranked video streams based on a score function [13]. Our algorithm extends Fagin's optimal aggregate ranking algorithm [6] by assuming no random access is available on the input streams. A new VDBMS query operator encapsulates the rank-join algorithm in its *GetNext* operation. Each call to *GetNext* returns the next top element from the ranked inputs. 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.

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 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.
- Parameter *k*, the number of requested output objects, is not known in advance, rather it increases for each call to *GetNext*.

The incremental and pipelining properties of our aggregation algorithm are essential for practical use in real-world database engines. Our new operator will help implement 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. It 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 Section 4.2, we demonstrate this concept in the context of performance analysis for different algorithms that implement the multi-feature ranking query operator.

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 with a predetermined 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 while pre-fetching 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 [2] above the buffer manager layer in VDBMS as multi-threaded modules. It has well defined interfaces with the query engine, the buffer manager, and the Extensible Abstract Data Type (E-ADT) interface.

3.5 Search-based Buffer Management

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. 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. Optimal pre-fetch and replacement policies would pre-fetch the data before its first reference and replace the data block that will not be referenced for the longest time [21]. 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 a 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 pre-fetch and cache pages expected for reference.

The VDBMS buffer management policy uses feedback from the query engine to make more accurate replacement and pre-fetching decisions [10]. Top-ranked query results 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.

In our search-based replacement policy, pages in the buffer pool that are referenced by either current or expected streams are considered for caching. We prefer caching pages that will be reference by current streams to those that will be referenced by expected streams, assigning higher *keep* weight values 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. The stream manager tracks the utilization of the streaming period, and utilizes any fraction of the streaming period unused by current streams to pre-fetch the first segment of the top ranked expected streams into the memory buffer. The pre-fetching 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 Section 4.1.They show that initial latency of the search-based policy is reduced on the average by 20% when compared with traditional policies.

4.0 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 two 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 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 [10], we execute 32 simultaneous clients. Each client submits an image-based query to VDBMS and receives a collection of key frame representing the results of a shot-based image similarity search. The client delays for a random period (uniformly distributed between 10 to 20 seconds) 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. Search results are synthesized by randomly selecting 10 candidate shots from the database. The random selection provides an upper bound for the performance of our policy. Our *keep* weight is set to three for pages referenced by an expected stream, and four for pages referenced by a current stream. Higher values for the keep parameter lead to excessive looping over buffer pages to find replacement candidates. The experimental data consists of eight one-hour videos, compressed in MPEG-1 format with a total size of five Gbytes. Each video has been pre-processed into shots with lengths between 5 and 10 minutes. We set the page size to 8Kbytes, the segment size to 30 pages, and the maximum number of concurrent streams to 16. Each experimental run lasts for 30 minutes, and the total number of buffer references is approximately 500,000.

We studied the performance of the following policies:

- Search-based replacement (SrchBR): pages cached if referenced by current or expected-stream requests
- Search-based pre-fetching and replacement (SrchBPR): first segment of expected-stream pre-fetched; pages cached if referenced by current or expected requests
- Stream-based replacement (StrmBR): pages cached only if referenced by concurrent stream request
- Use&Toss: pages are candidates for replacement immediately after use

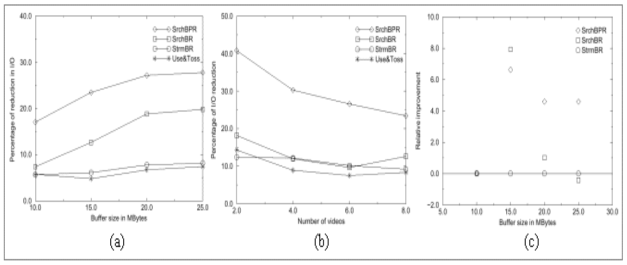


Figure 3. Reduction in I/O with change in (a) buffer size (b) number of videos. (c) Improvement in buffer hit ratio as buffer size changes.

Figure 3(a) 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 Mbytes. The figure shows that SrchBPR caches about 25% of the total pages of new streams based on the search results (initial latency reduced by 25%.) Although SrchBR achieves better results than StrmBR and Use&Toss, it caches only those pages either used by current streams or referenced by expected streams, therefore the improvement is smaller than that of the pre-fetch policy. StrmBR has no knowledge of expected streams and performs about the same as Use&Toss. In Figure 3(b), the buffer size is fixed at 25 Mbytes, 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 pre-fetching and replacement strategies, since more common data now exists between current and expected-streams. As the number of videos increases, the chance for interaction decreases and the improvement is dominated by the positive effects of pre-fetching. 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 both have no 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 set 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 3(c) 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 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 multi-feature queries. Our experimental study compares the VDBMS NRA-RJ operator, the J* operator introduced by Natsev et al.[18], and (for a baseline comparison) the non-pipelined version of the NRA algorithm as a multi-way rank-join operator, MW-RJ [6]. 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 multi-feature 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.

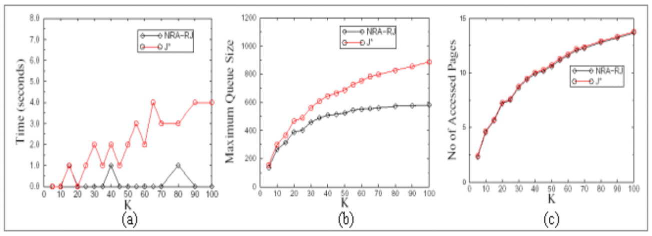


Figure 4. Comparison of NRA-RJ and J* for m=2.

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 4(a) 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 Figure 4(c). NRA-RJ has a smaller maximum queue size than that of J*, as shown in Figure 4(b), 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 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.

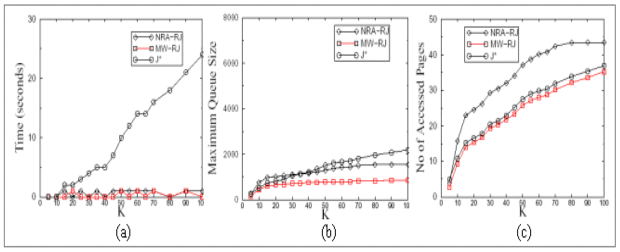


Figure 5. Comparison of NRA-RJ, J* and MW-RJ for m=3.

Figure 5 compares the NRA-RJ, J* and MW-RJ operators for *m=3*. Figure 5(a) 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 Figure 5(b) and the number of database accesses given in Figure5(c), we make the following observations:

- NRA-RJ has a larger maximum queue size and more database accesses than MW-RJ. This results from the tendency of NRA-RJ in the early pipeline stages to retrieve more database objects in order to deliver as many ranked tuples as required by the next NRA-RJ operator. We refer to this as NRA-RJ's *local ranking* problem.
- The J* operator has less database access cost than NRA-RJ, and close to the cost of MW-RJ, despite NRA-RJ's *local ranking* problem. In contrast to NRA-RJ, the J*'s algorithm does not retrieve equal numbers of objects from its left and right children.
- For the same reason that J* has less 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 Figure 5(c).

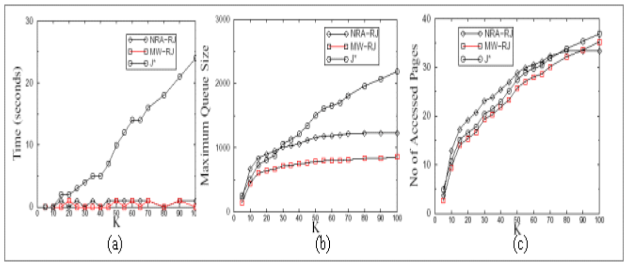


Figure 6. The optimized NRA-RJ operator.

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 retrieve 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 the performance. Figure 6 shows the comparison between the modified NRA-RJ, the J* and the MW-RJ operator. 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 less space requirements, and has a comparable number of disk accesses [13].

5.0 Conclusion

In this paper, we present a video database research initiative that resulted in the successful development of a video database management system which 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. 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 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.

References

[1] Aref, W., Catlin, A. C., Elmagarmid, A., Fan, J., Hammad, M., Ilyas, I., Marzouk, M., and Zhu, X. A video database management system for advancing video database research., *MIS2002. Intl. Workshop on Multimedia Info. Sys*. Tempe, Arizona. Nov. 2002.

[2] Aref, W., Catlin, A. C., Elmagarmid, A., Fan, J., Guo, J., Hammad, M., Ilyas, I., Marzouk, M., Prabhakar, S., Rezgui, A., Teoh, S., Terzi, E., Tu, Y., Vakali, A. and Zhu, X. A distributed server for continuous media. In *Proc. 18th Conf. on Data Eng*. San Jose, CA. Feb. 2002.

[3] Aref, W., Kamel, I. and Ghandeharizadeh, S. Disk scheduling in video editing systems. *IEEE Trans. on Knowledge & Data Eng.* 13(6). pp. 933-950. Nov. 2001.

[4] Berchtold, S., Böhm, C., Jagadish, H., Kriegel, H-P. and Sander, J. Independent quantization: An index compression technique for high-dimensional data spaces. In *Proc. 16th Intl. Conference on Data Engineering*. San Diego, CA. pp. 577-588. February 2000.

[5] Bertino, E., Hammad, H., Aref, W. and Elmagarmid, A. An access control model for video database systems. In *Proc. 9th Intl. Conf. on Info. & Knowledge Mgmt.* pp. 336-343. November 2000.

[6] Fagin, R., Lotem, A. and Naor, M. Optimal aggregation algorithms for middleware. In *PODS'01* Santa Barbara, CA. May 2001.

[7] Fan, J., Aref, W., Elmagarmid, A., Hacid, M-S., Marzouk, M. and Zhu, X. Multiview: Multi-level video content representation and retrieval. *Journal of Electrical Imaging,* Vol. 10, No. 4, pp. 895-908, Oct 2001.

[8] Fan, J., Hacid, M-S. and Elmagarmid, A. Model-based video classification for hierarchical video access. *Multimedia Tools and Appls.*. Vol. 15. Oct 2001.

[9] Guntzer, U., Balke, W-T. and Kiessling, W. Optimizing multi-feature queries for image databases. In *Proc. 26th Conf. on Very Large Databases*. Cairo, Egypt. p. 419-428. Sept 2000.

[10] Hammad, M., Aref, W., and Elmagarmid, A. Search-based buffer management policies for streaming in continuous media. In *Proc. IEEE Intl. Conf. on Multimedia & Expo.* Lausanne, Switzerland. Aug. 2002.

[11] Hellerstein, J., Naughton, J. and Pfeffer, A. Generalized search trees for database systems. In *Proc. of 21st Conf. on Very Large Data Bases*. Zurich, Switzerland. Sept 1995.

[12] Ilyas, I. and Aref, W. SP-GiST: An extensible database index for supporting space partitioning trees. *J. of Intelligent Sys. (JIIS)*. 17(2-3). pp. 215-235. 2001.

[13] Ilyas, I, Aref, W, and Elmagarmid, A. Joining ranked inputs in practice. In *Proc. 28th Conf. on Very Large Databases.* Hong Kong, China. 2002.

[14] ISO/IEC/JTC1/SC29/WG11: ISO/IEC 15938-3 Multimedia Content Description Interface *Final Committee Draft. Document No. N4062*. March 2001

[15] Katayama, N. and Satoh, S. The SR-tree: An index structure for high dimensional nearest neighbor queries. ACM *SIGMOD Record,* Vol. 26(2). 1997.

[16] Moser, F., Kraiss, A. and Klas, W. L. A buffer management strategy for interactive continuous data flows in a multimedia dbms. In *Proc. 21st Conf. on Very Large Databases*. Zurich, Switzerland. pp. 275-286. Sept. 1995.

[17] Nepal, S., Ramakrishna, M. Query processing issues in image (multimedia) databases. In *Proc. 15th Intl. Conf. on Data Eng*. March 23-26. Sydney, Australia. p. 22-29. IEEE Computer Society, 1999.

[18] Natsev, A., Chang, Y-C., Smith, J., Li, C-S. and Vitter, J. Supporting incremental join queries on ranked inputs. In *Proc. of 27th Conf. on Very Large Data Bases*. Rome, Italy. 2001.

[19] Ozden, B., Rastogi, R. and Silberschatz, A. Buffer replacement algorithms for multimedia storage systems. In *Proc. of IEEE Conf. on Multimedia Computing and Systems*. pp. 172-180. 1996.

[20] Seshadri, P. Predator: A resource for database research. *SIGMOD Record.* Vol. 27(1). pp. 16-20. 1998.

[21] Smith, J. Sequentiality and prefetching in database systems. *ACM Trans. on Database Systems*. 3(3). pp. 223-247. September 1978.

[22] Stonebraker, M. Operating system support for database management. *CACM*. 24(7). pp. 412-418. 1981

[23] Storage Manager Architecture. *Shore Documentation, Computer Sciences Department*. UW-Madison. 1999.

[24] Thomas, M., Carson, C., and Hellerstein, J. Creating a Customized Access Method for Blobworld, In *Proc 16th Conf. on Data Eng.*, San Diego, CA. Mar. 2000.