Multi Structural Query Engine on a Large Database using Vector Space Approach

C.Subramanian[1], Dr. T.Bhuvaneswari[2] and Dr.S.P.Rajagopalan [3]

[1]Research scholar, Dr.MGR Educational & Research Institute University, Chennai.
[2,3]Professor and Head, Department of Computer Applications , Dr.MGR Educational & Research Institute University, Chennai

Summary
The Multi-structure query engine is a new data framework to support efficient analysis of large and complex data sets. This approach consists of a set of data objects, together with a schema that specifies segmentations of the set of data objects according to multiple distinct criteria. It develops a rich set of analytical operations and assists in the design of highly efficient algorithms for these operations and allows the user to analyze the underlying data in terms of the allowed segmentations. The contribution is a framework for expressing and computing the highlights of a large and complex data set. This includes a data model that is rich enough to represent the kinds of structures found in real applications. Finally, it provides efficient algorithms that give approximately optimal solutions for three important classes of objective functions.

Keywords:
DBMS, Large Database, Data abstraction, Query Vector.

1. Introduction

A software system is considered as a database system, (a selective view on the history of databases) and the success of relational database management systems in the business domain can be attributed to the different kind of databases. A database management system (DBMS) is a general purpose software system that facilitates the processes of defining, constructing and manipulating databases for various applications. The main objective of a DBMS is to treat data as a manageable corporate resource so as to increase data utilization and to integrate smoothly the data access and processing function with the rest of the organization. It should also enhance data security and provide data integrity. A DBMS provides to evolve by emphasizing data independence programs that access data maintained in the DBMS. To handling data using a DBMS provides an alternative for traditional file processing. In this approach, each user defines and implements the files needed for a specific application.
The following three characteristics distinguish the database approach from file processing:
(i) data abstraction
(ii) a database is self-contained
(iii) Program-data independence and program-operation independence.

Manuscript received June 5, 2011
Manuscript revised June 20, 2011

2. The Relational Data Model

A relational database management system (RDBMS) is a DBMS based on the relational data model. The relational data model helps to protect users of large data banks from having to know how the data is organized in the machine. Although the idea of RDBMS is perceived too theoretical by the majority of practitioners, prototype relational systems have proved that an implementation could be reasonably efficient. The nonrelational database systems are not provided much data independence. Application programs stop working after the representation changed, because they referenced nonexistent files. The formal model abstracts from ordering, indexing, and access paths. Since, data is only accessed through the model, changing the aspects cannot affect the correctness of applications any longer.

2.1 Database design with the relational model

The modeling of data with the relational data model can be explained by considering the disc example, which is shown in Figure 1.

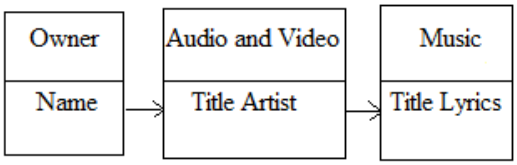


Figure 1 The Universe of Discourse (UoD) of a disc database

It defines the domains: titles (T), performing artists (A), music titles (M) and owner names (O). The collection of discs can be represented as a relation R(T,A,M,O) that relation is just a single value, one of all possible collections of compact discs that can be constructed in this UoD. In a database system, a relation variable 'C' of (relational) type R(T,A,M,O). A design based on one relation R(T,A,M,O) is not the only possible relational model of the UoD. It is important that neither of these

alternatives determines how the data is physically stored. Although the second alternative had less redundancy, this redundancy is at the conceptual level, and is not necessarily reflected at the physical level.

2.2 The Three Schema Architecture

The database systems proposed the three schema architecture as a framework for the design of DBMSs. This architecture of database management systems is shown in Figure 2.

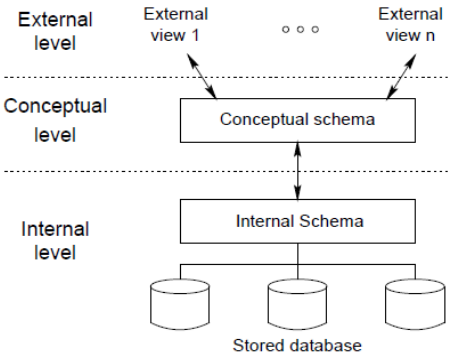


Figure 2 The three schema architecture

Its goal is to separate the user applications and the physical database by emphasizing data independence, which insulates a user from the adverse effects of the evolution of the database environment. The threeschema architecture is developed for database systems that operate in the business domain. Although the data independence can extend to emerging domains like digital libraries, whether DBMSs that operates in such emerging domains can be implemented according to the architecture is shown in Figure 2. The three schema architecture recognizes the following three levels in a database system:

(i) ***The internal level*** has an internal schema, which describes the physical storage structure of the database. It is oriented towards the most efficient use of the computing facility.

(ii) ***The conceptual level*** has a conceptual schema, which describes the structure of the database for its user community, but hides the storage details. The conceptual schema describes a model of the UoD, maintained for all applications of the enterprise.

(iii) ***The external level*** includes a number of external schemas or user views. The external schemas are simplified models of the UoD.

A DBMS based on the three schema architecture maintains several descriptions and mappings between the levels that are not known before hand and can change over time. Therefore, a DBMS provides a variety of languages for the specification of schemas and the manipulation of data at different levels of the architecture. Most notable are:

(i) ***DDL****(Data Definition Language),* which is used to specify the database schema, and

(ii) ***DML****(Data Manipulation Language),* used to manipulate the stored database. Typical manipulations include retrieval, insertion, deletion, and modification of the data.

(iii) ***DCL****(Data Control Language)* is used for managing transactions, access rights, and the creation and deletion of access structures.

(iv) ***SDL****(Storage Definition Language*) is used to specify the internal schema.

(v) ***VDL****(View Definition Language)* is to specify user views and their mappings to the conceptual schema.

2.3 Efficient Query Evaluation

The role of data abstraction in the database approach, data manipulation can described at the abstract level of the data model, where it makes no sense to talk about efficiency: database query languages are high-level declarative languages that can only express what data should be affected. Thus, the efficient evaluation of expressions in a query language is the responsibility of the database system as shown in Figure 3.

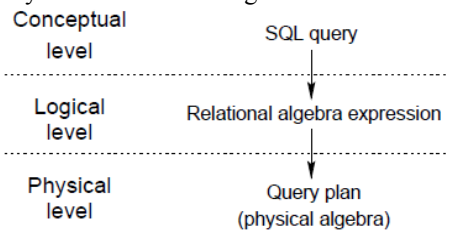


Figure 3 Query evaluation in databases

The techniques applied in the implementation of database systems that enable efficient query evaluation is shown in Figure 3. Database query languages are usually based on settheory and applied predicate logic. Most end-user languages, including SQL and relational calculus are based on the following structure, known as the setcomprehension expression is given in eqn. [1]

$$\{f(x) \mid x \varepsilon X \wedge p(x)\} \quad \ldots (1)$$

Query processing bridges the gap between the database query language and the file system. It transforms requests specified in the database query language into the query plan, a sequence of operations in the physical access language. Query optimization attempts to determine the optimal query plan. However, the search space consisting of all query plans that implement the user's original request is too large to be searched exhaustively. As a result, the selected query plan is often only suboptimal. In

any implementation of a database system, the task of the query optimizer is more to avoid very inefficient query plans, than to select the one very best option.

3. Large Database Architecture

The Large Database architecture is an alternative design for database systems in which the design of a Large Database has a layered architecture, with a central role for data abstraction. The unique, distinguishing aspect of the architecture is that the conceptual data model used by its endusers is mapped to a physical implementation using different data models at different levels of the database architecture. The implementation is separated in three layers:

(i) conceptual
(ii) logical and
(iii) physical layer

The Large database architecture is shown in Figure 4, takes up the gauntlet. It concerns the definition of interfaces between components and not a complete instruction for implementation.
In this work, instead of transforming complex object queries directly into operations in a physical algebra, as in most database architectures, query evaluation in a Large Database takes place in several phases. Both the logical and the physical layer can be extended with domain specific data types and operators. Each of these layers can be viewed as implementing complete three-schema architecture with its own data model and query languages as shown in Figure 5.

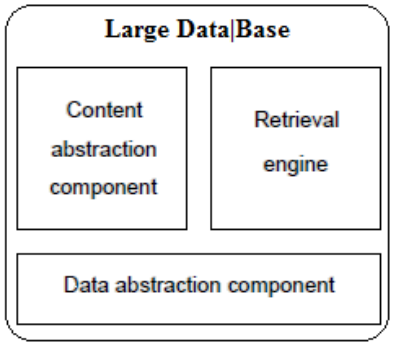


Figure 4 Large database architecture

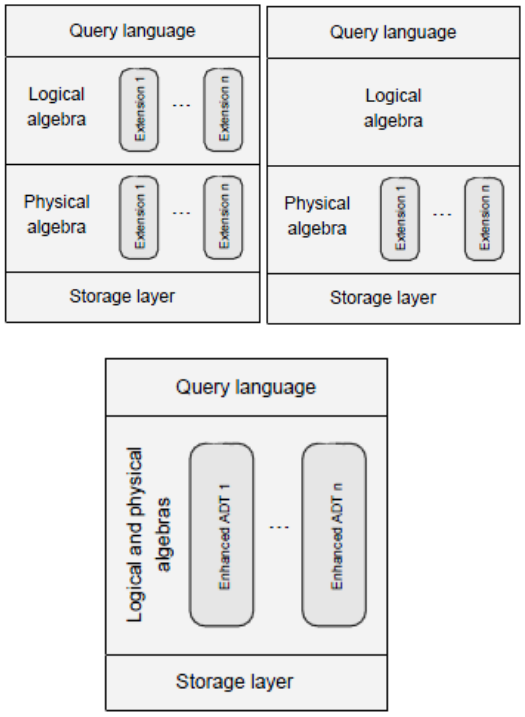


Figure 5 The Large Database architecture to the extended relational model

4. Multi-structural query engine

The query engine implements the basic (f, o) dynamic programming algorithm for hierarchical and numerical dimensions, and provides a framework for computing sequential and factored Pairwise-Disjoint Collection (PDCs) for arbitrary collections of dimensions. It also implements more efficient algorithms for min-monotone PDCs, and for sum-additive PDCs. A simple optimizer selects the appropriate algorithm at each step. Finally, the system includes implementations of the query types described in Section 4.1. The query engine should be viewed as a reference implementation to compute multi-structural queries. It has not been optimized for performance. A system can produce responses to complex multi-structural queries in times measured in seconds or tens of seconds, rather than hours. There are many future modifications that could provide further improvements and so the timing numbers should be viewed as upper bounds.

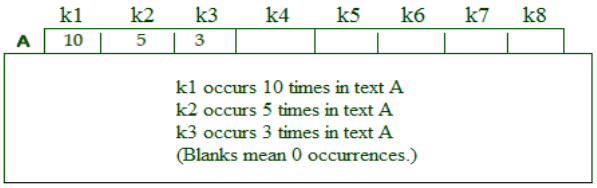
4.1 Algorithm:

1. Parse the query.

2. Convert words into wordIDs.
3. Seek to the start of the doclist in the short barrel for every word.
4. Scan through the doclists until there is a document that matches all the search terms.
5. Compute the rank of that document for the query.
6. If in the short barrels and at the end of any doclist, seek to the start of the doclist in the full barrel for every word and go to step 4.
7. If not at the end of any doclist go to step 4. Sort the documents that have matched by rank and return the top k.

5. Vector Space Model

A vector is like an array of floating point which has direction and magnitude. Each vector holds a place for every term in the collection, therefore, most of the vectors are sparse. Documents are represented as "group of words" represented as vectors when used computationally.



5.1Document Vectors

In this vector, one location for each word is shown in Figure 6, where all keywords occur with different times in text A.

Figure 6 keywords occur with different times in text

The vector scheme detects inconsistencies when copies of a document are independently updated. This scheme allows copies of a document to be stored at multiple hosts. Although it uses the term document, the scheme can be applied to any other data items such as tuples of a relation. The vector scheme is initially designed to deal with failure in distributed file systems. The scheme gained importance because mobile computers often store copies of files that are also present in the server system, in effect constituting a distributed file system that is often disconnected. Another application of the scheme is in groupware system, where hosts are connected periodically, rather than continuously and must exchange updated documents. The vector scheme also has applications in replicated databases.
In Figure 7 shows, Plot the vectors that are "close" in space, which is similar.

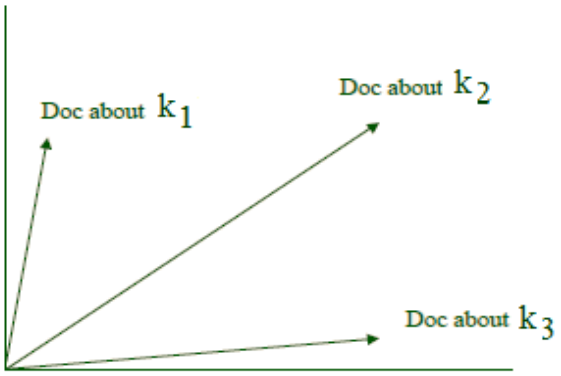


Figure 7 Plots of the vector

5.2 Vector Space Model

In vector space model, the Documents (D) are represented as vectors in term of space, these terms are usually stems. Documents represented by binary vectors of terms. The queries represented the same as documents. A vector distance measure between the query and documents is used to rank retrieved the documents. Query and Document similarity is based on length and direction of their vectors, in which vector operations is to capture boolean query.

5.2.1 Vector Space Documents and Queries

In the Figure 8, three different terms, in which t1 and t2 has common numbers are D5 and D6. Compare between t2 and t3 shows common numbers are D3, D5 and D10 and compare between t3 and t1 shows common numbers are D1 and D5 common number between all terms is D5.

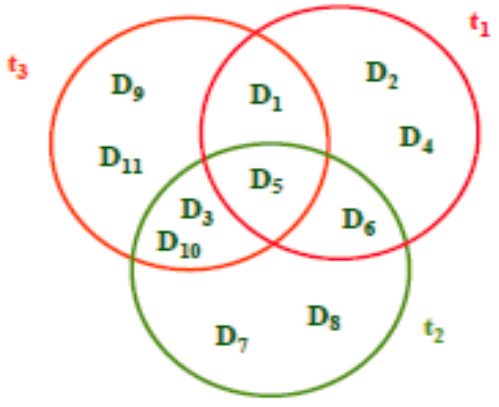


Figure 8 Boolean Term Combinations

docs	t1	t2	t3	RSV=Q.Di
D1	1	0	1	4
D2	1	0	0	1
D3	0	1	1	5
D4	1	0	0	1
D5	1	1	1	6
D6	1	1	0	3
D7	0	1	0	2
D8	0	1	0	2
D9	0	0	1	3
D10	0	1	1	5
D11	1	0	1	3
Q	1	2	3	
	q1	q2	q3	

Figure 9 Q is a query – also represented as a vector

$$w_{ik} = tf_{ik} * \log (N / n_k) \quad \ldots (2)$$

T_k = term k in document D_i
tf_{ik} = frequency of term T_k in document D_i
id_{ik} = inverse document frequency of term T_k in C
N = total number of documents in the collection C
n_k = the number of documents in C that contain T_k

$$id_{fk} = \log \left(\frac{N}{nk}\right)$$

5.2.2 Vector Space with Term Weights and Cosine Matching

Figure 10 shows,

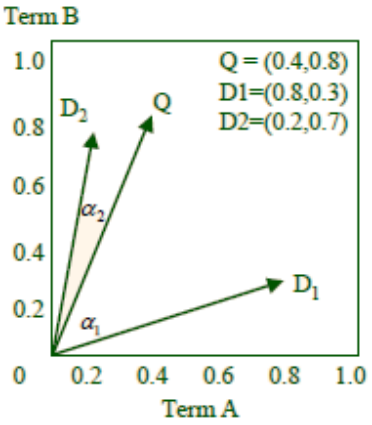


Figure 10 Different matching of Vector space

$$D_i=(d_{i1},w_{di1};d_{i2}, w_{di2};\ldots;d_{it}, w_{dit})$$
$$Q =(q_{i1},w_{qi1};q_{i2}, w_{qi2};\ldots;q_{it}, w_{qit})$$

$$sim(Q, D_i) = \frac{\sum_{j=1}^{t} w_{q_j} w_{d_{ij}}}{\sqrt{\sum_{j=1}^{t} (w_{q_j})^2 \sum_{j=1}^{t} (w_{d_{ij}})^2}}$$

$$sim(Q, D2) = \frac{(0.4 \cdot 0.2) + (0.8 \cdot 0.7)}{\sqrt{[(0.4)^2 + (0.8)^2] \cdot [(0.2)^2 + (0.7)^2]}}$$

$$= \frac{0.64}{\sqrt{0.42}} = 0.98$$

$$sim(Q, D_1) = \frac{.56}{\sqrt{0.58}} = 0.74$$

5.2.3 Problems with Vector Space

(i) There is no real theoretical basis for the assumption of a term space, it is more for visualization than having any real basis.
(ii) Terms are not really orthogonal dimensions and not independent of all other terms.

The main idea to this work that modify the existing query based on relevance decision, then extract terms from relevant documents and add them to the query. The AND/OR re-weight the terms already in the Query. There are many variations to exist the query approach.

(i) Usually positive weights for terms from relevant docs,
(ii) Sometimes negative weights for terms from non-relevant docs.

$$Q_1 = \alpha\ Q_0 + \frac{\beta}{n_1} \sum_{i=1}^{n_1} R_i - \frac{\gamma}{n_2} \sum_{i=1}^{n_2} S_i \quad \ldots (4)$$

where,
Q_0 = the vector for the initial query
R_i = the vector for the relevant document i
S_i = the vector for the non-relevant document i
n_1 = the number of relevant documents chosen
n_2 = the number of non-relevant documents chosen
α, β and γ tune the importance of relevant and nonrelevant terms

5.2.4 Vector Illustration

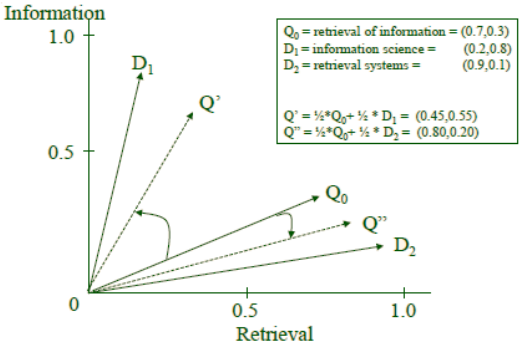


Figure 11 Vector Illustration

6. Experimental Results

Each query requires heavy processing and is run either once or twice for each sample of three sample sizes: 100, 1,000, and 5,000 documents. The index timings are higher than in the warm cache case, but the overall end-to-end latencies are not significantly larger. Figure 12(a) shows the histogram of the number of seconds spent in the backend generating all necessary data. Figure 12(b) shows, the same results for end-to-end processing time.

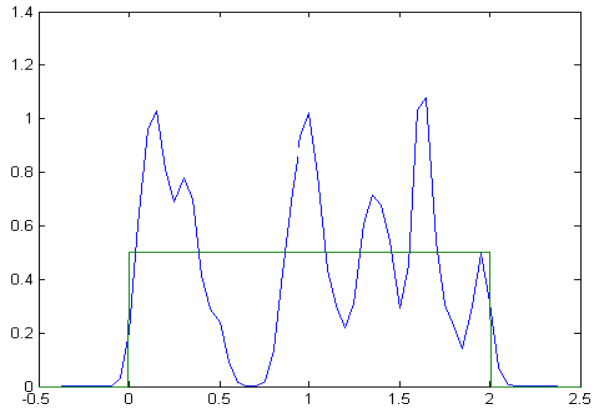


Figure 12(a) Histogram of the processing with necessary data

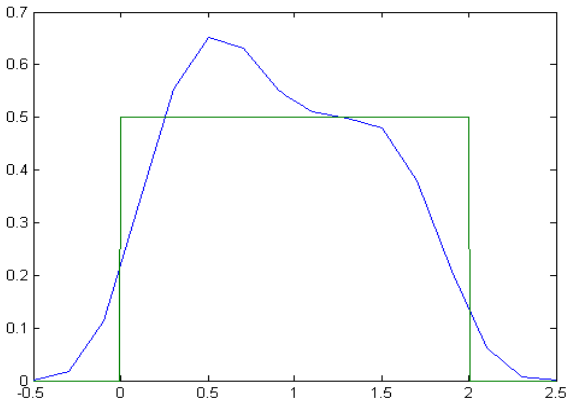


Figure 12(b) Histogram of the end-to-end processing

The bytes of data and metadata exploration are being returned by these queries. Figure 13 shows a histogram of the cached size for sample sizes of 1,000 documents and 10,000 documents. Several queries at the 10,000 document size return in excess of 100M of data, which is marshaled, transferred over the network, unmarshaled and written to disk in the client cache. The left frame Figure 13(a) shows the results for 1,000 document samples and the right frame Figure 13(b) shows the results for 10,000 document samples.

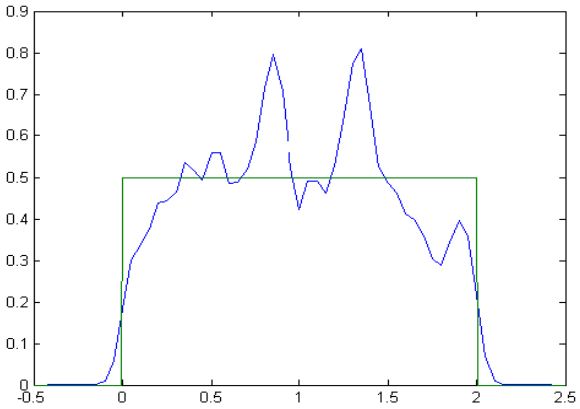


Figure 13(a) Sizes in megabytes for data returned per query for left frame

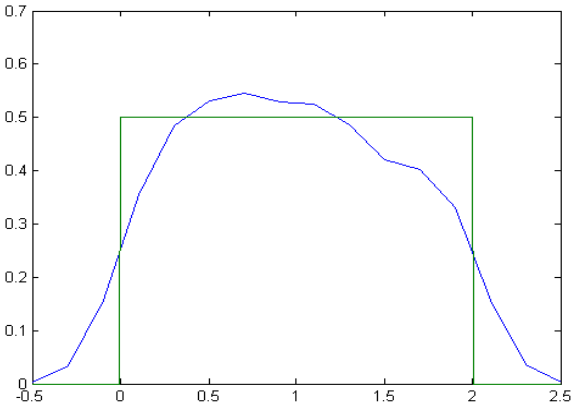


Figure 13(b) Sizes in megabytes for data returned per query for right frame

6.1 Multi-structural query engine performance

In the query engine, the average time to compute a multi-structural query is, broken by sample size and size ‘k’ of the resulting PDC. The size of the resulting PDC is a much less significant contributor to the overall time than the sample size. Certain query types are quadratic in the sample size and hence, show significant growth with sample size. An average time in seconds to solve multi-structural query, after all data and metadata has been loaded from the backend, over 28 benchmark queries for various sample sizes and PDC sizes ‘k’, is analyzed. Figure 14 shows the results for PDCs of various different sample sizes. From Figure 14(a) and Figure 14(b), it is clear that certain query types have significantly higher processing times than others. These queries have more compute-intensive ‘f’ functions, which dominate the runtime of the queries that take more than one minute. Average query engine timing over all queries within a

query family for sample sizes of 100, 1,000 and 5,000 documents, after all backend activity has completed. Results for k = 5 are shown in Figure 14(a), and for k = 10 Figure 14(b).

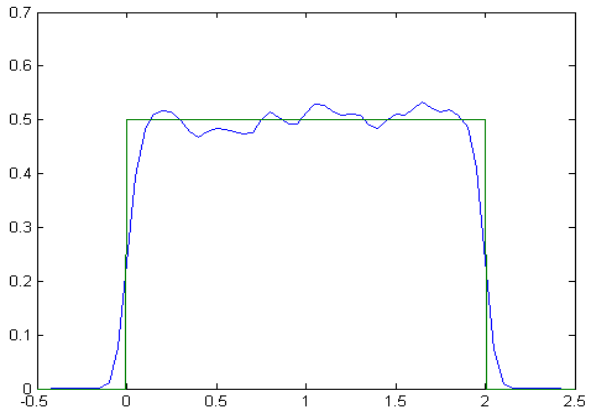


Figure 14(a) Average query engine timing over all queries

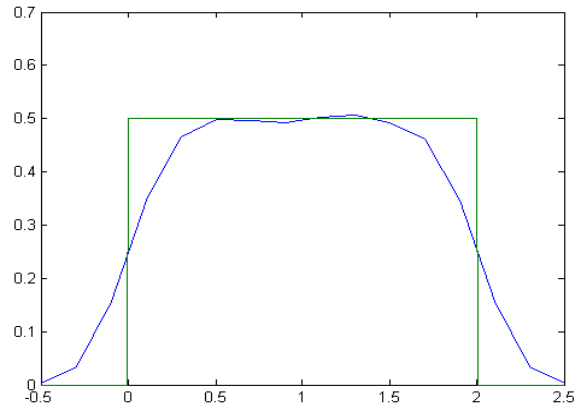


Figure 14(b) Aggregate end-to-end timings of overall system timings

7. Conclusion

In this paper, a database management system is proposed (general-purpose software system) that facilitates the processes of defining, constructing, and manipulating databases for various applications. The main characteristic of the 'database approach' is that it increases the value of data by its emphasis on data independence. The main problem of multimedia data management is providing access to the stored objects. The content structure of administrative data is easily represented in alphanumeric values. Thus, database technology has primarily focused on handling the objects' logical structure. In the multimedia data, representation of content is far from trivial though, and not supported by current database management systems.

Retrieval systems based on these ideas are typically standalone systems that have been developed for very specific applications. There is not much consensus on the integration of these techniques in general-purpose DBMSs. This leaves to the user the burdens of both query formulation and the combination of results for each single representation into a final judgment. Also, this leads to inefficient query processing for queries involving several content representations. Like any DBMS, a MMDBMS is a general-purpose software system that supports various applications; but, the support is targeted to applications in the specific domain of digital libraries. Four new requirements have been identified for this domain: (i) multimedia objects can be active objects, (ii) querying is an interaction process, (iii) query processing uses multiple representations, and (iv) query formulation provides content independence. Recognizing the strong relationship with IR query processing, the network retrieval model is adapted for multimedia retrieval.

References

[1] A.P. de Vries, "Mirror: Multimedia query processing in extensible databases", In Proceedings of the fourteenth Twente workshop on language technology (TWLT14): Language Technology in Multimedia Information Retrieval, pages 37–48, Enschede, The Netherlands, December 1998.

[2] A.P. de Vries and H.M. Blanken. Database technology and the management of multimedia data in Mirror. In Multimedia Storage and Archiving Systems III, volume 3527 of Proceedings of SPIE, pages 443–455, Boston MA, November 1998.

[3] A.P. de Vries and H.M. Blanken. The relationship between IR and multimedia databases. In The 20th IRSG colloquium: discovering new worlds of IR, Grenoble, France, March 1998.

[4] A. Hampapur and R. Jain. Multimedia data management. Using metadata to integrate and apply digital media, chapter Video data management systems: metadata and architecture, pages 245–286. In Sheth and Klas [SK98], 1998.

[5] E. Remias, G. Sheikholeslami, and A. Zhang. Blockoriented image decomposition and retrieval in image database systems. In The 1996 International Workshop on Multimedia Database Management Systems, Blue Mountain Lake, New York, August 1996.

[6] I. Mani, D. House, M. Maybury, and M. Green. Intelligent multimedia information retrieval, chapter Towards contentbased browsing of broadcast news video, pages 241–258. AAAI Press/MIT Press, 1997.

[7] Nita Goyal, Charles Hoch, Ravi Krishnamurthy, Brian Meckler, and Michael Suckow. Is gui programming a database research problem? In H. V. Jagadish and Inderpal Singh Mumick, editors, Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 46, 1996, pages 517–528. ACM Press, 1996

[8] P.A. Boncz, S. Manegold, and M.L. Kersten. Database architecture optimized for the new bottleneck: Memory access. In Proceedings of 25th International Conference on Very Large Databases (VLDB '99), Edinburgh, Scotland, UK, September 1999.
[9] S. Boll,W. Klas, and A. Sheth. Multimedia data management. Using metadata to integrate and apply digital media, chapter Overview on using metadata to manage multimedia data, pages 1–24. In Sheth and Klas [SK98], 1998.
[10] Thomas V. Papathomas, Tiffany E. Conway, Ingemar J. Cox, Joumana Ghosn, Matt L. Miller, Thomas P. Minka, , and Peter N. Yianilos. Psychophysical studies of the performance of an image database retrieval system. In Proc. SPIE, 1998.
[11] W. Greiff, W.B. Croft, and H. Turtle. PIC matrices: A computationally tractable class of probabilistic query operators. Technical Report IR132, The Center for Intelligent Information Retrieval, 1998. submitted to ACM TOIS.
[12] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, P. Yanker, and C. Faloutsos. The QBIC project: querying images by content using color, texture and shape. Technical Report RJ 9203, IBM Research Division, 1993.
[13] Haizhou Li, Bin Ma, and Chin-Hui Lee, "A Vector Space Modeling Approach to Spoken Language Identification", in IEEE Transactions on Audio, speech, and Language processing vol. 15, No. 1, January 2007.
[14] M. Basavaraju and Dr. R. Prabhakar, "A Novel Method of Spam Mail Detection using Text Based Clustering Approach", in International Journal of Computer Applications (0975 – 8887), Vol.5– No.4, August 2010.
[15] Christian Bockermann, Martin Apel, and Michael Meier, "Learning SQL for Database Intrusion Detection Using Context-Sensitive Modelling (Extended Abstract)", in U. Flegel and D. Bruschi (Eds.): DIMVA, LNCS 5587, pp. 196–205, 2009.
[16] N. Fuhr and K. Großjohann, "XIRQL: A Query Language for Information Retrieval in XML Documents", in proceedings of the 24th Annual ACM SIGIR Conference on Research and Development in Information Retrieval, pages 172–180. ACM Press, 2001.
[17] Ronald Fagin R. Guha Ravi Kumar Jasmine Novak D. Sivakumar Andrew Tomkins, "MultiStructural Databases", in PODS June 13–15, Baltimore, MD, 2005.
[18] R. Fagin, Ph. Kolaitis, R. Kumar, J. Novak, D. Sivakumar, A. Tomkins, "Efficient Implementation of Large-Scale Multi-Structural Databases", Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005.
[19] R. Fagin, R. Guha, R. Kumar, J. Novak, D. Sivakumar, and A. Tomkins, "Multi-structural databases. In Proceeding 24th ACM Symposium on Principles of Database Systems, 2005.