Embedded Database Query Processing Plan Generation Using Dynamic Programming

B. Selmi, H. Gharsellaoui, and S. Bouamama

***Abstract*—The efficiency of processing queries in an embedded database is critical for the system performance. The principal mechanism through which an embedded database maintains an optimal level of performance is the database query optimizer. It reduces the response time of a given query and the total time of processing queries. Nevertheless, because of the optimizer's importance to the robustness and flexibility of an embedded database, we outline the embedded database query optimization by generating an optimal query processing plan. In this paper, an approach is presented which is able to generate an optimal query processing plans for a given user query. The approach uses dynamic programming to determine optimal query plans for a given query.**

***Index Terms*—Embedded databases system, optimization, query plan, query cost.**

I. Introduction

In the last years the deployment of embedded real-time systems has increased dramatically. At the same time, the amount of data that needs to be managed by embedded real-time systems is increasing, thus requiring an efficient and structured data management. Hence, database functionality is needed to provide support for storage and manipulation of data in embedded real-time systems. However, a database that can be used in an embedded real-time system must fulfill requirements both from an embedded system and from a real-time system, i.e., at the same time the database needs to be an embedded and a real-time database [1]. The main objectives for an embedded database are low memory usage, i.e., small memory footprint, portability to different operating system platforms, efficient resource management, e.g., minimization of the CPU usage, ability to run for long periods of time without administration, and ability to be tailored for different applications. In addition, development costs must be kept as low as possible, with short time-to-market and reliable software.

An embedded database is an integral part of such applications or application infrastructures. Unlike traditional DBMSs, database functionality is delivered as part of the application or application infrastructure. These databases run with or as part of the applications in embedded systems [2]. Embedded databases provide an organized mechanism to access large volumes of data for applications. Instead of providing full features of traditional DBMSs, such as complex query optimization and handling mechanisms, embedded databases provide minimal functionality such as indexing, concurrency control, logging, and transactional guarantees. It is actually a broad technology category that includes: database systems with differing application programming interfaces (SQL as well as proprietary, native APIs); database architectures (client-server and Peer-to-Peer); storage modes (on-disk, in-memory, and combined); database models (relational-oriented, object-oriented), [2].

Nowadays, there are many embedded databases on the market, but, they vary widely from vendor to vendor. Existing commercial embedded database systems, e.g., Polyhedra, RDM, Velocis, Pervasive, SQL, Berkeley DB, and TimesTen, have different characteristics and are designed with specific applications in mind. They support different data models, e.g., relational vs object-oriented model, and operating system platforms. Moreover, they have different memory requirements and provide different types of interfaces for users to access data in the database. Application developers must carefully choose the embedded database that their application requires, and find the balance between the functionality an application requires and the functionality that an embedded database offers. Thus, finding the right embedded database, in addition of being a quiet time consuming, costly, concurrency control, transaction scheduling, and logging and recovery.

The rest of the paper is organized as follows. Section II presents some related works to the embedded databases. Section III presents the background of our approach within the heuristic used and the method which is dynamic programming. Section IV discusses the approach for query plan generation. Finally, the conclusion is provided in Section V.

II. Related Works

We present some works dealing with embedded and real-time systems. ODEA [3] is a platform for the embedded applications to achieve the QoS metrics which are the timeliness of transactions and the temporal freshness of data but not totally guaranteed. It adds an object-manager module that provides the organization of the main memory, the creation and deletion of RT objects and the concurrency control for these objects. Anti-Caching [4], the authors implement a prototype to overcome the restriction that all data fit in main memory where cold data is moved to disk and hot data stays in main memory, where the main memory is the primary storage device not the disk like the traditional DBMS. Data are stored even in the memory or in the disk not like the traditional DBMS, data can be in the memory and the

Manuscript received January 15, 2017; revised March 17, 2017.

B. Selmi is with the National Engineering School of Manouba, Manouba University, Tunisia (e-mail: boubaker.selmi@hotmail.fr).

H. Gharsellaoui was with National Engineering School of Carthage, Carthage University, Tunisia (e-mail: gharsellaoui.hamza@gmail.com).

S. Bouamama is with FCIT, University of Jeddah, KSA, National Engineering School of Manouba, Manouba University, Tunisia (e-mail: Sbouamama@uj.edu.sa).

DOI: 10.7763/IJCTE.2017.V9.1161

disk. QeDB (Quality-aware real-time Embedded DataBase) [5] is a database for data-intensive real-time applications running on embedded devices where not all the data can be fit into main-memory. QeDB is an extension of Berkeley DB. It has been designed to improve and support the QoS in the embedded applications, which are the timeliness of transactions and the freshness of data by using a novel feedback control technique [6] and exploiting Multiple Input/Output (MIMO) operations. eXtremeDB Fusion [7] is a solution for military and aerospace, it is a small footprint embedded real time DataBase for MilAero (Military-Aerospace). Also used in other various fields such as: control flights, fusion of sensor data, radio, telecommunications and driver support… eXtremeDB Fusion offers a small code pattern (50K) with advanced features by providing: high availability of data; hybrid storage data from the memory and disk. SQLite [8] is an open-source embedded relational database to meet the needs of embedded systems which is small, fast, simple, reliable and easy-to-port. SQLite has such special advantages such as powerful, fast, simple interface as well as small footprint, easy to control without any external dependencies and no setup or administration needed so it's especially suitable for applications in embedded environment. SQLite transactions are Atomic, Consistent, Isolated, and Durable (ACID Properties).

In [9], the authors optimize SQLite COMMIT strategy by using the FileManager to optimize the response time of transactions. Also, to reduce the frequency of executing COMMIT by wrapping up necessary sequences in one transaction:

1- Wrap all sequences of SQL statements during mobile device connection initial.

2- Wrap all sequences of SQL statements for transferring each file. FileManager is an application program that manages file storing and provides file-searching service. It stores files and database files in the Flash Cards. The users transfer files to the Flash Cards. At this time, the mobile will update related information into the database which is stored in the flash memory.

LGeDBMS [10] is a small DBMS for mobile embedded system with flash memory. It has been designed to meet those features:

1- Optimized to flash memory with LFS design principle,

2- Compact size suitable for consumer electronics appliances,

3- Transaction process based on flash memory characteristics.

The LGeDBMS process makes only an atomic and durable update operation to reduce as possible the unnecessary management cost. To be resist for a system crash, LGeDBMS uses a PID mapping table visioning scheme for logging/recovery. However, LGeDBMS method reduces the number of I/Os by writing a final PID mapping table than writing a log for each data change. Embedded RFID Middleware [11] is software intermediate between the embedded system software (embedded operating system, embedded database) and application software. It uses the functions and services offered by embedded operating system in goal to provide the development environment for the application. Embedded RFID middleware is designed to deal the limits of embedded systems, which are: power consumption optimization of embedded system software; Real-Time support; and limited resources (it must have a complete control of resources, design optimization algorithm, and control the use of resources). In [12], the authors propose an algorithm to optimize the embedded query by using a practical swarm optimization (PSO). The previous version of PSO finds at each iteration 2 solutions, pbest (fitness achieved till current iteration) and gbest (global best solution). Based on this, the authors design a new version of PSO, which is named "dynamic PSO algorithm".

A. Embedded Database System

The database engine is the boss control module of the database system, its main function is to achieve global control and ensure the correctness and efficiency of the work. It monitors the database during all operations, control the allocation and management of resources. Data access module implements all the basic operations on the base table, which is the core of the operating system to achieve database module. Its functions:

- Find a record based on property values.
- Find records using the relative position.
- Add a record to a base table.
- Delete a record from the base table.
- Modify a record and write a result back to the base table.

Database maintenance module is used to backup and restore the database if it is necessary.

We will implement NoSQL embedded database. NoSQL [13]-[15] is a new solution embedded database introduced and began in 2009 and is growing rapidly, still till now evaluated and contributed new types and versions of embedded databases. It is designed to address some of the points: being non-relational, distributed, open-source and horizontal scalable. NoSQL means not only SQL. NoSQL characteristics: schema-free and less, easy replication support and distribution, simple API, Queries need to return answers quickly, mostly query and few updates, asynchronous inserts and updates, eventually consistent / BASE (ACID transaction properties are not needed), Large data volumes, CAP Theorem (Consistency, Availability, Partition tolerance). Consistency means all nodes see the same data at the same time and a set of operations has occurred all at once. Availability means that node failures do not prevent survivors from continuing to operate and every operation must terminate in an intended response. Partition tolerance means that the system continues to operate despite arbitrary message loss and the operations will complete even if individual components are unavailable. NoSQL Databases Types:

- Column Store: each storage block contains data from only one column,
- Document store: stores documents made up of tagged elements,
- Key-value store: is a hash table of keys.

III. Background

As the data required to process the user query existing in various relations, there is a need to process a query plan that

has an optimal cost. The query processing cost will be cheaper if the numbers of query's relations are low and concentrations of data queried on the relations are higher. This happens because cost of transmission of data from different relations will be higher if numbers of relations are bigger. When the numbers of relations used are less, then sending of data will not be required as much. There are many issues in query plan generation which use the iterative improvement and simulated annealing algorithms to produce query plans and generate an optimal one. In [16], [17], the authors used an optimal hybrid genetic based approach to resolve the real-time scheduling of embedded systems with optimization. Also, in [18] authors take the genetic algorithm as a method to solve this problem of finding an optimal query plan in their work. Our new original proposed work uses the two heuristics defined in [18] with a proposition of a new one.

A. The Heuristic

The first heuristic works on number of relations required to answer the user query. Lesser the number of relations involved in query processing, lesser will be the communication between the relations. As a result, query processing will be efficient. Second heuristic works on the concentration of data queried on the relations. There will be many valid query plans having low number of relations but will be preferred which are having higher number of data queried on the relations. Based on these two heuristics and the Query Proximity Cost (QPC) given in [18], we will propose a new version of this QPC as the following expression demonstrates:

$$\sum_{i=1}^{M} \frac{Ri}{N}\left(1-\frac{Ri}{N}\right)$$

where M is the number of relations accessed by the query plan, Ri is the number of times the ith relation is used in the Query plan and N is the number of data queried by the query. The QPC varies from 0 to (N-1)/N, where Zero specifies that all the data queried by the queries, resides in the same relation and therefore will be the closest. On the other hand, (N-1)/N specifies that each of the data queried by the query, resides in different relations and therefore are the least closest. The query plans having less QPC are generated using dynamic programming, which is discussed next.

B. Dynamic Programming

This algorithm is developed in IBM's System R project [19] and it is used in almost all commercial database products [20]. It works in bottom-up way by building more complex sub-plans from simpler sub-plans until the complete plan is constructed. In the first phase, the algorithm builds access plan for every relation in the query. Typically, there are several different access plans for a relation. In the second phase, the algorithm finds all two-way join plans using the access plans as building blocks. Again, the algorithm would find alternative join plans for all relations. The algorithm executes in this way until it has enumerated all n-way join plans, and those plans are passed by the "finalizePlans" function to become full plans for the query. The beauty of the dynamic programming is that inferior plans are pruned. This is carried out by the "prunePlan" function.

IV. The Proposed Approach

The dynamic programming algorithm is adapted to query plan generation problem. The dynamic programming algorithm for query plan generation is shown below. This algorithm takes relation-data matrix and relation participating in the FROM clause of the SQL query as input and produces optimal query plan as output.

INPUT: Relation-Data matrix and relations participating in the FROM clause of the query

OUTPUT: Optimal plan with minimum cost.

Method:

Step 1: Find all possible access paths of the relations of q;

Step 2: Evaluate QPC of each query plan based on closeness;

$$\sum_{i=1}^{M} \frac{Ri}{N}\left(1-\frac{Ri}{N}\right)$$

Step 3: compare their cost and keep the least expensive and pruned the others;

Step 4: Add the resulting plans into set D;

Step 5: For i=1 to number of joins in q do;

Step 6: consider joining the relevant access paths found in previous iterations using all possible join methods;

Step 7: compare the cost of the resulting plans and keep the least expensive then pruned the others;

Step 8: Add the resulting plans into set D;

Step 9: end for;

Step 10: Return Optimal Plan;

Where q is the query

A. Proof

As an example, we will consider the following query:

Select D1, D2, D3, D4
From R1, R2, R3, R4
where R1.D1=R2.D2 AND R3.D3=R4.D4

R1, R2, R3, R4 are the relations crossed by the query. To answer the query, it requires four data D1, D2, D3, and D4. Therefore, the length of query plan is four. The query plans are constructed by the relations containing the data in the SELECT clause. We assume the relation-data matrix given below: (see Table I).

TABLE I: Relation-Data Matrix Description

Relation-Data Martix	
R1	D1 D3 D7 D9
R2	D1 D6 D4 D2
R3	D1 D5 D8 D4
R4	D1 D3 D4 D2

This matrix gives an overview of the relations and the data it contains. Relation 1 (R1) contains Data 1, Data 3, Data 7 and Data 9. Relation 2 (R2) contains Data 1, Data 6, Data 4 and Data 2. Relation 3 (R3) contains Data 1, Data 5, Data 8 and Data 4. Relation 4 (R4) contains Data 1, Data 3, Data 4 and Data 2.

We will present some of possible query plans.

TABLE II: RELATIONS DESCRIPTION

Query Plans	
1 3 4 2	D1 at R1, D2 at R3 , D3 at R4, D4 at R2
3 3 4 2	D1 at R3, D2 at R3 , D3 at R4, D4 at R2
1 1 1 3	D1 at R1, D2 at R1 , D3 at R1, D4 at R3
1 1 4 4	D1 at R1, D2 at R1 , D3 at R4, D4 at R4

The number of possible query plans is NR1*NR2*NR3*NR4 where NRi is the number of relations containing the ith data. In this example; it is (4*4*4*4) valid query plans. For the query plans given in Table II, the first one involves four (4) relations, the second one involves three (3) relations, whereas the third and fourth involve two (2) relations. The third query plan has three data residing in relation 1 and a data residing in relation 3, this implies that this query plan has a higher concentration of data at the relation 1. So, it would be considered more optimal than the other query plans.

For the query plan given previous, we will introduce their QPC below:

TABLE III: QUERY PROXIMITY COST VALUES

Query Plans	Query Proximity Cost	QPC Value
[1,3,4,2]	1/4(1-1/4)+1/4(1-1/4)+1/4(1-1/4)	9/16
[3,3,4,2]	2/4(1-2/4)+1/4(1-1/4)+1/4(1-1/4)	5/8
[1,1,1,3]	3/4(1-3/4)+1/4(1-1/4)	3/8
[1,1,4,4]	2/4(1-2/4)+ 2/4(1-2/4)	1/2

As we can see in Table III, query plan 3 is the most optimal followed by query plan 4, then query plan 2, and the query plan 1 is the most expensive because it involves more number of relations participating in the query plan. So, less number of relations in the query plan gives less QPC and more optimal query plan.

This approach presents several advantages over other technologies in the sense that is easier to structure the data to specific real-time needs. The design is closer to the implementation and development is very simple and it is easier to modify processing and evolve data structures. Nonetheless, our proposed approach suffers from some drawbacks, including difficulty expressing integrity of the real-time constraints in embedded databases and difficulty in installation. We will work hardly in the next issues in order to handle these difficulties.

V. CONCLUSION

This paper focuses on the problem of embedded database query optimization, which is of great importance in embedded database system design. In this paper, an algorithm is proposed based on dynamic programming to generate query processing plans with the required data. This is done by formulating the query processing plan generation problem as a single-objective dynamic programming which the objective is to find an optimal query plan with minimum QPC. This query plan generated involves minimum number of relations to answer the user query. However, we are planning to develop a better solution during our future works. Moreover, we want to apply the proposed technique to big data centers.

REFERENCES

[1] A. Tesanovic, D. Nystr öm, J. Hansson, and C. Norstr öm, "Embedded databases for embedded real-time systems: A component-based approach," *Report-MRTC*, Sweden, pp. 1-77, 2002.

[2] A. Nori, "Mobile and embedded databases," in *Proc. the ACM SIGMOD International Conference on Management of Data*, 2007, pp. 1175-1177.

[3] Z. Ellouze, N. Louati, and R. Bouaziz, "A next generation object-oriented environment for real-time database application development (ODEA)," in *Proc. 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, 2013, pp. 1224-1232.

[4] J. DeBrabant, A. Pavlo, and S. Tu, "Anti-caching: A new approach to database management system architecture," in *Proc. the VLDB Endowment*, vol. 6, no. 14, 2013.

[5] W. Kang, S. H. Son, and J. A. Stankovic, "Design, implementation, and evaluation of a Qos-aware real-time embedded database," in *Proc. IEEE Transactions on Computers*, January 2012, pp. 45-59.

[6] K. D. Kang, J. Oh, and S. H. Son, "Chronos: Feedback control of a real database system performance," in *Proc. 28th IEEE International Real-Time Systems Symposium*, 2007, pp. 267–276.

[7] McObject LLC, *eXtremeDB Fusion for Military and Aerospace Applications*, 2009.

[8] L. Junyan, X. Shiguo, and L. Yijie, "Application Research of Embedded Database SQLite," in *Proc. International Forum on Information Technology and Applications*, 2009, pp. 539–543.

[9] W. Song and T. Tao, "Performance optimization for flash memory database in mobile embedded system," in *Proc. Second International Workshop on Education Technology and Computer Scienc*e, 2010, pp. 35–39.

[10] G. Jeong, K. S.-C. Baek, H.-S. Lee, H.-D. Lee, and M. Jeung Joe, "LGeDBMS: A small DBMS for embedded system with flash memory," in *Proc. of 32th International Conference on Very Large Data Bases (VLDB)*, pp. 1255-1258.

[11] Z. X. Li and Y. X. Zhang, "Design and implementation of embedded rfid middleware," in *Proc. International Conference on Medical Physics and Biomedical Engineering (ICMPBE)*, 2012, vol. 33, pp. 587-596.

[12] X. Mingyao and X. F. Li, "Embedded database query optimization algorithm based on particle swarm optimization," in *Proc. Seventh International Conference on Measuring Technology and Mechatronics Automation*, 2015, pp. 429–432.

[13] R. P. Padhy, M. R. Patra, and S. C. Satapathy, "RDBMS to NoSQL: reviewing some next-generation non-relational database's," *International Journal of Advanced Engineering Sciences and Technologies*, vol. 11, no. 1, pp. 015–030, 2011.

[14] C. Strauch, *Course of Studies Computer Science and Media (CSM) University Hochschule der Medien, Stuttgart (Stuttgart Media University)*, 2009.

[15] A. Salminen, "NoSQL seminar 2010 @ TUT," 2010.

[16] I. Gharbi, H. Gharsellaoui, and S. Bouamama, "A hybrid genetic based approach for real-time reconfigurable scheduling of os tasks in uniprocessor embedded systems," in *Proc. 17th International Conference on Enterprise Information Systems, (ICEIS)*, 2015, pp. 385-390.

[17] T. V. V. Kumar, V. Singh, and A. K. Verma, "Generating distributed query processing plans using genetic algorithm," *in Proc. International Conference on Data Storage and Data Engineering*, 2010, pp. 173-177.

[18] D. Kossmann, "The state of art in distributed query optimization," in *Proc. ACM Computing Surveys*, September 2000, vol. 32, no. 4, pp. 423-469.

[19] A. Aljanaby, E. Abuelrub, and M. Odeh, "A survey of distributed query optimization," in *Proc. The International Arab Journal of Information Technology*, January 2005, vol. 2, no. 1, pp. 48-57.

Selmi Boubaker was born in Bizerte-Tunisia on September 29, 1988. Mr Selmi Boubaker received his master degree in computer science from Higher Institute of Computer Sciences and Management, University of Kairouan-Tunisia in 2013. Now, he is a Ph.D Student in computer science at National Engineering School of Manouba, Manouba University-Tunisia.

He is a temporary assistant at the Higher Institute of Management of Sousse, Sousse University-Tunisia. After that, he was working as a substitute teacher for two years at Sousse University and Kairouan University. From April, 2014 to December, 2014 he worked as an IT Support in an international Company of Trading, and also in 2012 as an ERP administrator in Pharmaceutical Company.

Hamza Gharsellaoui received the B.S. degree in computer science from Tunis El-Manar University, Tunis, Tunisia, in 2004, and the M.S. degree in industrial computer science from National Institute of Applied Sciences and Technology (INSAT), Carthage University, Tunis, Tunisia, in 2007. He did research in computer science at National Institute of Applied Sciences and Technology, Carthage University, Tunis, Tunisia to receive the Ph.D. degree, in 2013. He was a Researcher in computer science at Al-Jouf College of Technology, Technical and Vocational Training Corporation, Sakaka, Kingdom of Saudi Arabia (KSA). He is a Part-time Researcher and Assistant Professor at ENIC School, Carthage University in Tunisia.

H. Gharsellaoui is active in several projects and in other interesting international collaborations. He was with National Engineering School of Carthage, Carthage University, Tunisia.