Class-based Query-Optimization for Minimizing Worst-Case Execution Times of Diagnostic Queries in Embedded Real-Time Systems

Nadra Tabassam and Roman Obermaisser
University of Siegen
nadra.tabassam@uni-siegen.de

Abstract— **Active diagnosis in real time embedded computer systems increases the overall reliability of the system by performing error detection and fault recovery. Real time databases and diagnostic queries are a common solution to realize active diagnosis. This paper presents a technique to optimize the diagnostic queries in a fault tolerant real time embedded system. A directed graph called the DMG (Diagnostic Multi-query Graph) based on the diagnostic symptoms and features is the input to the query optimization module for the processing of each query within a short worst case execution time. The diagnostic inference process is temporally and spatially decomposed by introducing intermediate inference steps called symptoms. These symptoms and diagnostic features extracted from the DMG are stored in an embedded database created in a Pervasive SQL server. The query execution is based on periods and each query node of the DMG has to finish within its time bound which is worst case execution time of the query. At first the estimated worst case execution time for each diagnostic query is calculated. After that the algorithm optimizes the diagnostic query using a class based query categorization technique. The access method for each query is selected on the basis of its type. For join queries the most optimized join order is calculated by estimating the selectivity factor based on the number of tuples present in each join order. Results presented in this context show that the diagnostic queries are optimized effectively and their estimated worst case execution time is minimized.**

Keywords-active diagnosis; embedded real time database; worst case execution time; directed multi-query graph

I. INTRODUCTION

Real time systems based on active diagnosis are different from other common systems in terms of their timing requirement for diagnosis. In the context of applications having active diagnosis of faults, each diagnostic task has to complete within a pre-defined deadline. Reliability is an important feature which is essential for the active diagnosis. If a real time application is unable to fulfill the requirement of fault diagnosis at run time within hard time bounds then ultimate consequences can be fatal. In order to maintain the timeliness of these applications, it is imperative to be guaranteed that all the tasks complete within the pre-defined deadline. Example systems which benefit from active diagnosis are health management systems [1], command/control systems [2], electric power distribution applications [3], air craft control systems [4].

The pervasiveness of embedded real time systems keeps increasing. These systems are virtually present in every aspect of our life. At the same time the amount of data these applications process, is also immensely increasing. It is extremely important to store and manage data efficiently. Current techniques used for data manipulation in real time embedded systems are often ad-hoc [5].

Database functionality deals with the systematic storage and manipulation of data. Embedded database systems within real time systems have several benefits: (i) re-usability of database systems decreases the development cost, (ii) safe and consistent manipulation of data enhances the design of the system, (iii) better maintainability during the software evolution, (iv) data remains logically and temporally consistent [6].

There are many embedded database systems for the processing and manipulation of real time data. The database systems provide solutions for problems like timeliness, effective resource management and concurrency control in the domain of active diagnostic applications. In active diagnostic applications which are using embedded databases systems, it should be guaranteed that all the deadlines are meet. An important aspect of these databases is how they are guaranteeing response times (i.e bounded worst case execution time) of a diagnostic query. Embedded database systems which are commercially available include Velocies [7], Polyhedra [8], Pervasive.SQL [9], Berkeley DB [10], and TimesTen [11]. All of these database systems deals with various schemes for transaction scheduling, concurrency control, logging and recovery. Research projects that are built using these embedded database systems are BeeHive [12], DeeDs [13] and ART-RTDB [14]. The implementation cost of these database systems is quite low and they can be optimized easily according to the requirements of a particular application.

Analysis of diagnostic queries can be implemented through different techniques. Diagnostic information stored and analyzed later is considered to be passive diagnosis. In contrast executing fault recovery mechanisms at run time, falls in the category of active diagnosis [15]. A key requirement is to execute all the diagnostic queries with in a deadline. Numerous control systems [16] are able to deal with the loss for a few cycles, whereas longer disruption may lead towards the overall failure of a system.

The novelty of current work is based on the representation, transformation and processing of diagnostic queries. Diagnostic queries are represented in the form Of DMG. Fault diagnostic queries contains the SQL operations like select,

978-1-5386-0837-1/17/$31.00 ©2017 IEEE

project and join. These queries are based on the features and symptoms extracted directly from the sensors data. Queries are executed on the embedded database designed in PSQL. Transformation of these queries are based on the single node transformation of DMG. Processing of these diagnostic queries is the detection of faults. Minimization of Worst case execution time of these fault diagnostic queries in the time driven and non-preemptive real time systems has not performed before. Minimizing the worst case execution time for these diagnostic queries increase the optimization of fault diagnosis process which is a necessary constitute for the active diagnosis in real time systems.

In this work, a query optimization technique is develop for processing the diagnostic information (queries) within the stringent deadlines for fault diagnosis. Estimated worst case execution time for each query is calculated by using measurement based technique. Afterwards the class base categorization of each query is performed at run time to decrease the estimated worst case execution time. With this class definition for each query type and the selection of the join order with minimum execution time, query execution plan with minimum cost is found. Diagnostic queries are in the form of a diagnostic multi-query graph (DMG). Query optimization technique also reduces the overall estimated worst case execution time of the DMG.

The rest of the paper is structured as follows. Section II gives an overview of the related work. Section III describes the structural features of a DMG. Section IV describes the query optimization techniques implemented for diagnostic queries. Section V describes an example. Section VI describes the results and the paper closes with the conclusion and future work.

II. RELATED WORK

Timing analysis method range from the traditional response time analysis, capturing the dependencies between each part of the application on the basis of its execution times of tasks to probabilistic analysis techniques and weaker bounds with limits on the maximum number of violations [17]. There are some systems which can bear the violation of deadlines up to some extent. For these systems calculating an overly pessimistic WCET is not often feasible. Some systems are greatly effected by the average response time as compared to the WCET [18].

Probabilistic WCET analysis techniques employ a probability distribution of execution times with confidence levels. For example a method for computing probabilistic bounds on the execution times of task for the the worst case path of code is propose in [19].

In case of embedded system databases, research work until now is primarily focused on the evaluation and investigation of different algorithms related to the concurrency control and scheduling for sets of transactions [20]. Usually the performance of these algorithms along with the number of deadlines which are missed are studied. These algorithms are dramatically effected by the number of transactions that system has to process. There are number of algorithms that actually deals with the database over load situations [21].

For safety critical applications it should be guaranteed that deadlines hold for all use cases of the system. Also the WCET of the database transactions in the domain of event driven hard real time systems has been calculated [22]. Tasks are scheduled based on the preemptive scheduling technique known as EDF. However, this technique considers the non-distributed domain. There has been no research for the time driven and non-preemptive features of the real time databases.

III. DIAGNOSTIC MULTI-QUERY GRAPH

Diagnostic queries which are the input to the query optimization module are presented in the form of a directed graph. Such a DMG is comprised of nodes named as features, symptoms and faults. Each parent node of the DMG is considered to be a feature. A symptom is a node with both incoming and outgoing edges and faults are the end nodes with only incoming edges. Each symptom and each fault in the graph represents a rule, which is realized as a query on the diagnostic facts within a real time database, while edges in the DMG specify the input/output relationships via the real time database.

Each feature and symptom value within a DMG is a diagnostic information. This diagnostic information originally is the query to be executed on the real time database. It is a directed graph where each node itself is a query graph with different query based operations. A query inside a node can be structured as a query operator tree with query operators like selection, projection and join. The DMG is triggered in a cyclic manner with strict periodicity. The execution of each time period starts from the parents node and ends at a child nodes.

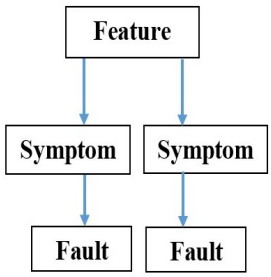

Fig. 1. Directed Multi-Query Graph

An important technique for the query transformation in a DMG is the ”single node transformation” as depicted in fig.1. A single node transformation is actually the transformation of a single query within one node of the DMG. Fig.2 shows that each query within the node of a DMG has to complete before its deadline. An important constraint in our system is that there is no collision between the different transactions. Each transaction will complete its execution before the next transaction accesses the database.

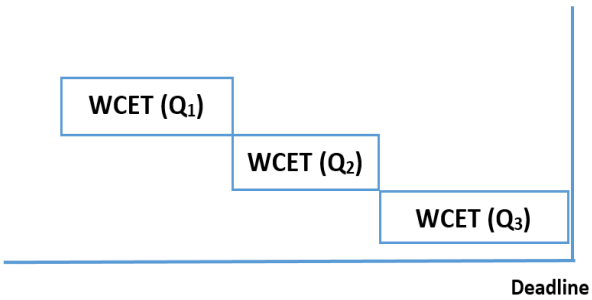

Fig. 2. WCET for queries

The overall DMG is defined by the following equations:

$$G = (N, E) \tag{1}$$

$$G(N) = (n_1, n_2, ..., n_i) \tag{2}$$

$$G(E) = (e_1, e_2, ..., e_i) \tag{3}$$

Where N is a pair of nodes and E is the set of directed edges. Edges of the DMG also represent the data and precedence constraints of the connected nodes. An entry node is a node with no parent while a node with no child is a exit node. Weight of each edge is the number of tuples transferred from the parent to the child after the query execution is completed within one node. The query that is associated with the particular node is $Q(n_i)$. The weight for each node n_i is its estimated worst case execution time denoted by $EWCET(Q(n_i))$.

A. Calculation of Estimated Worst Case Execution Time

$EWCET(Q(n_i))$ is provided as an essential input for the query optimization. The method used for estimating the execution time of a query measures the end-to-end execution time for a subset of possible executions (test cases). There is a difference between the worst case execution time and the estimate of the WCET, where the term use estimated WCET is the derivation of estimates for the maximum execution time of diagnostic queries. A measurement based method is employed for estimating WCET of query [23]. For example in our experimental evaluation five thousand possible executions are selected as a set of queries. By measuring the maximum execution time for these set of queries, the maximum and minimum observed execution time is derived. End to end time measurement of these queries generates the estimation time but not the actual bound. So the $EWCET(Q(n_i))$ is the CPU time consumed by a query till its execution is completed [22].

IV. Query Optimization of Diagnostic Queries

Each node of the DMG has a query associated with it. Query operations in the form of selection, projection and join are being extracted from each node in order to execute it. After the query is extracted from the DMG, it requires optimization so that the fault can be diagnosed within the deadline defined for the query. The most important task is to optimize the diagnostic queries in a manner that they decreases the estimated WCET and meet the system's deadline. For this purpose a class based query optimization algorithm is being presented. The algorithm generates optimized query execution plans on the basis of the query's cost (minimum in terms of time). The algorithm comprise of three steps.

- Finding the class of the query based on the types of queries.
- On the basis of the query type, class based query categorization is used to find the best access method for the query with minimum execution time.
- The optimized order of the join relations is selected on the basis of the selectivity ratio of tuples. The selectivity ratio of the join order is actually the time taken by a query to extract the number of tuples within a certain join order. Join orders with lower selectivity ratio (join combination which requires to read the minimum number of tuples) or in other words having lower processing time for extracting the tuples are considered.

There are multiple access methods like full table scan, B^+-tree scan for accessing the relations in the query but in the case of our real time system the access method selected for the query should consider the type of query. Instead of considering all the access methods the one which is most suitable is selected on the basis of a query type. Class categorizations for simple queries are as follows:

TABLE I
Classes For Simple Query Types

Name	Query Type	Operation	Access Method
A	Simple queries	select	Full table scan
B	Search queries	>,<,=	B-tree scan

For selecting the best access method for simple select queries the class categorization is described in Table I. Class categorization for selecting the best access method for join queries is described in Table II.

TABLE II
Classes For join Query Types

Name	Query Type	Join Access Method
C	Inner table of join fits in the memory	Nested loop
D	Join is an equi-join	Hash
E	Not an equi-join	Sort Merge

Before selecting the join order for the diagnostic query, it is important to find the access method for the relations in the query. The access method for the join query is selected by using class based categorization from TABLE II. After this the best join order is selected. Join order optimization has been a research issue for many years. For this purpose the left deep tree is generated particularly on the join predicate of the query. Left deep trees are preferred because they minimize the overall memory utilization of the query, because in the case of real time systems the memory utilization is an important constraint [24].

A. Selection of Best Join Order

The optimized order of the relations present within the join is selected on the basis of the selectivity factor. The selectivity factor is actually the ratio of tuples required by each join order. This ratio is between the read and write require: by the total number of tuples present in the table and the number of tuples needs to be extracted during a certain join order. The selectivity factor is based on the following database parameters.

TABLE III
DATABASE PARAMETERS

Parameter	Description
R	Relation in a database
t_i	Set of all tuples over relation R
$(R_1 \bowtie R_2 \bowtie ... \bowtie R_i)$	Join order of relations within query
$Q(t_i)$	Query predicate/condition over the tuple ti
$D_r((R_1 \bowtie R_2 \bowtie ... \bowtie R_i)t_i)$	Disk reads require for join order
$D_w((R_1 \bowtie R_2 \bowtie ... \bowtie R_i)t_i)$	Disk write require for the join order

$$A = [n(D_r(R_1)) + n(D_r(R_2)) + ... + n(D_r(R_i))] \quad (4)$$

$$B = [n(D_w(R_1)) + n(D_w(R_2)) + ... + n(D_w(R_i))] \quad (5)$$

$$C = n(D_r(\exists t_i \in (R_1 \bowtie R_2 \bowtie ... \bowtie R_i)Q(t_i))) \quad (6)$$

$$D = n(D_w(\exists t_i \in (R_1 \bowtie R_2 \bowtie ... \bowtie R_i)Q(t_i))) \quad (7)$$

Equation (4) and (5) represent the number of disk reads and disk writes required by the full relation scan present in the query. Equation (6) and (7) represents that there exists a tuple ti in the relation order $(R_1 \bowtie R_2 \bowtie ... \bowtie R_i)$ on which the predicate $Q(t_i)$ is true. Variables C and D give the number of disk reads and disk writes required for the selection of tuples based on the predicate $Q(t_i)$ within a certain join order. The selectivity factor for each join order is defined by equation (8):

$$Sf_j = \frac{A+B}{C+D} \quad (8)$$

The solutions space Sp is based on the solution set of different selectivity factors for each join combination.

$$Sp(Sf) = (Sf_1, Sf_2, Sf_3 ... Sf_i)$$

The best selectivity factor for the join order will be

$$Sp(Sf) = min(Sf_1, Sf_2, Sf_3 ... Sf_i) \quad (9)$$

So overall the selection of access method along with the best join order based on the minimum selectivity factor, generates the optimized query.

V. EXAMPLE SCENARIO

This section presents an example diagnostic query which executes on the database schema created in Pervasive.SQL. The query optimization module is implemented by using JDBC interfacing with PSQL. The database contains tables based on the sensor information from vehicles. Pervasive SQL has been used because it has numerous benefits over the conventional database systems as described in Section I.

By using this query categorization based on defined query classes, our optimizer determines the type of queries and the access method which is appropriate for a particular type of query. This will minimize the overall time required for query execution.

The DMG presented in Fig.1 contains the queries which need to be processed. Each node in a DMG contains a query which contributes to the detection of a fault. The overall DMG has a cyclic time period, so processing of each query inside a node has a certain deadline associated with it and the query has to complete within that.

The first node of the DMG contains the features of the query. In our example database, car diagnostic data is taken into account for fault diagnosis in the system. The input data for the DMG is generated from the car sensors.

Select Fault from OxygenSensors O JOIN EngineSensors AS E on O.TempID=E.TempID JOIN TemperatureSensors AS T ON T.TempID=E.TempID where $O.AirRatio > 0.1$ and $E.FuelRatio > 0.2$ and $O.OilTemp > 0.3$.

Fig. 3. Diagnostic query

The query presented in Fig. 3 is extracted from the DMG, based on the following features and symptoms.

A. Features

The three features of the car sensors in which the query has to look during execution are the oil temperature, fuel ratio and air ratio.

B. Symptoms

A symptom node comprised of the values which are actually given as an output by the sensors. These values are actually the current data generated by the sensors in the time period of 15ms. Symptoms in the presented query are the values present in the where clause which are AirRatio and FuelRatio and OilTemp. Each symptom value is compared with the threshold values. If the generated symptom value is greater than the threshold value then it is considered to be a fault in the system.

At the end on the basis of both features and symptoms a fault will be diagnosed and the query will be executed. The proposed algorithm implements the query optimization in the following steps.

- It selects the class of the query on the basis of its type. The presented query belongs to the class D where the joins are equi-joins so the selected access method will be the hash join method.

- The maximum observed execution time is the time required by the query to execute with the selected access method and join order.

Three tables oxygensensors(O), EngineSensors(E) and TemperatureSensors(T) are joined in the example query. All the join orders are considered and the best one is selected on the basis of the value of the selectivity factor. All the join orders are exhaustively searched by the query optimization module for finding their selectivity factor ratio. Data required for query execution are populated with sensor data every 15ms. For three joins we can have six possible left deep join trees. The join cost for each join combination is calculated on the basis of the selectivity factor.

TABLE IV
COST ESTIMATION OF JOIN ORDERS

Access method	Join Orders	SF	ET
Hash Join	$O \bowtie (E \bowtie T)$	87%	910
	$E \bowtie (O \bowtie T)$	56%	721
	$T \bowtie (E \bowtie O)$	43%	643
	$O \bowtie (T \bowtie E)$	31%	577
	$E \bowtie (T \bowtie O)$	51%	702
	$T \bowtie (O \bowtie E)$	27%	567

Table IV shows the execution time taken by the overall query. Selectivity factor for each join order is shown. So overall the join order with minimum selectivity factor is selected. The minimum execution time is the required output for diagnostic query.

VI. RESULTS

This section represents the results of the diagnostic query optimization performed over the queries with different numbers of joins.

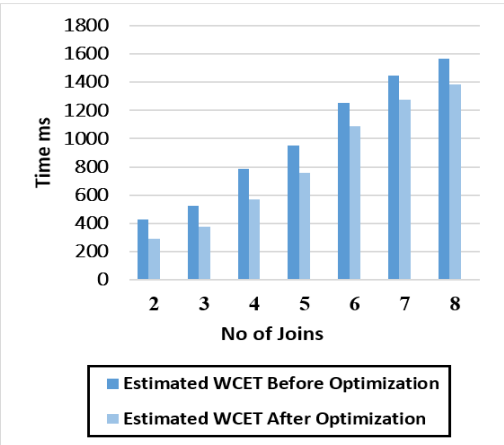

Fig. 4. Comparison of EWCET before and after optimization

Fig. 4 presents the comparison between the estimated worst case execution time of the diagnostic queries before and after optimization. Different number of joins are taken into consideration for analyzing the query performance. It is observed that the estimated worst case execution time for diagnostic queries are decreased after optimization. The execution of queries within their time bound makes the active diagnosis of our faults more reliable. To make sure that all faults of the system are detected within their time bound, the estimated worst case execution time of diagnostic queries are measured before and after the optimization. It is clearly seen from the results that after the optimization all the diagnostic queries remain within the bound of their estimated worst case execution time, which is the most important objective to achieve for our real time embedded system. Because if the estimated worst case execution time of diagnostic queries after the optimization increases from their time bound (EWCET before optimization) then it is obvious that the fault detection of our real time system does not falls in the category of active diagnosis. So overall optimization of these diagnostic queries enables the system to be more fault tolerant. Decreasing overall estimated worst case execution time is important in our proposed real time system, so that fault of a system can be diagnose before any critical condition may occur, which may leads the system to the breakage.

VII. CONCLUSIONS AND FUTURE WORK

The paper has presented an optimization technique for diagnostic queries based on a directed multi-query graph in real time systems. The proposed technique enables the diagnostic queries to complete before their deadlines. The embedded database system used in experimental evaluation is Pervasive SQL. Estimated WCET for diagnostic queries is calculated by measurement based method. Diagnostic queries are optimized by extracting them from a DMG and by applying a class based categorization technique. The class based categorization technique selects the best access method for queries based on the query type. After the selection of the access method, the best join order is selected by calculating the selectivity factor. The search space for join orders has been minimized by implementing left depth tree. Results shows that the overall estimated worst case execution time has decreased after the query optimization. Multi-Query transformation of the entire DMG is considered to be the research objective for future work.

ACKNOWLEDGMENT

This work has been supported in part by the European project FP7 DREAMS under project No. 610640 and by the German Research Foundation (DFG) under project ADISTES (OB384/6-1, 629300).

REFERENCES

[1] T. Abdelzaher, C.D. Gill, R. Rajkumar, and J.A. Stankovic. Distributed real-time and embedded systems research in the context of GENI. Technical report, National Science Foundation Workshop, 2006.
[2] A.-M. Grisogono. The implications of complex adaptive systems theory for C2. Technical report, Defence Science and Technology Organisation, 2006.
[3] E.O. Schweitzer, D. Whitehead, A. Guzman, Y. Gong, and M. Donolo. Advanced real-time synchrophasor applications. In Proc. of the 35th Annual Western Protective Relay Conference, 2008.

[4] U.Shanker, M.Misra, and A.Sarje. Distributed real time database systems: background and literature review Distributed and Parallel Databases, pp. 127-49, April 2008.
[5] J.Ganssle, The Art of Designing Embedded Systems, Second Edition, Newnes is an imprint of Elsevier
[6] Y. Jayanta Singh et al.Dynamic Management of Transactions in Distributed Real-Time Processing System,international journal of database management systems,Vol.2,No.2,May 2010.
[7] MBrane Ltd. http://www.mbrane.com.
[8] Polyhedra Plc. http://www.polyhedra.com.
[9] Pervasive Software Inc. http://www.pervasive.com.
[10] Sleepycat Software Inc. http://www.sleepycat.com.
[11] TimesTen Performance Software. http://www.timesten.com.
[12] J. A. Stankovic, S. H. Son, and J. Liebeherr. Real-Time Databases and Information Systems, chapter BeeHive: Global Multimedia Database Support for Dependable,Real-Time Applications, pp.409422. Kluwer Academic Publishers, 1997.
[13] S. F. Andler, J. Hansson, J. Eriksson, J. Mellin, M. Berndtsson, and B. Eftring. DeeDS Towards a Distributed and Active Real-Time Database System. ACM Sigmond Record, Volume 25, 1996.
[14] D. Isovic, M. Lindgren, and I. Crnkovic. System development with real-time components.In Proceedings of ECOOP Workshop - Pervasive Component-Based Systems,France, June 2000.
[15] H. Kopetz. Real-Time Systems, Design Principles for Distributed Embedded Applications. Springer, 2011.
[16] G. Heiner and T. Thurner. Time-triggered architecture for safety-related distributed real-time systems in transportation systems. In Proc. of 28th Int. Symposium on Fault-Tolerant Computing, pages 402407, June 1998
[17] Z. A. H. Hammadeh, S. Quinton, and R. Ernst, Extending typical worst-case analysis using response-time dependencies to bound deadline misses, in Proceedings of the International Conference on Embedded Software, New Delhi, India, October 2014.
[18] J. Maki-Turja and M. Sj odin, Response-time analysis for transactions with execution-time dependencies, in 19th International Conference on Real-Time and Network Systems (RTNS11), Sep. 2011.
[19] H. Zeng, M. Di Natale, P. Giusto, and A. Sangiovanni-Vincentelli, Using Statistical Methods to Compute the Probability Distribution of Message Response time in controller area network, Industrial Informatics, IEEE Transactions on, vol. 6, no. 4, pp. 678 691, Nov. 2010.
[20] G. Bernat, A. Colin, and S. Petters, WCET analysis of probabilistic hard real-time systems, 23rd IEEE Real-Time Systems Symposium (RTSS02), 2002, pp. 279288.
[21] V. C. S. Lee, K.-Y. Lam, and B. Kao. Priority Scheduling of Transactions in Distributed Real-Time Databases. Real Time Systems, Vol. 16, No. I, pages 31-62, January 1999.
[22] A. Munnich and G. Farber,Calculating worst-case execution times of transactions in databases for event-driven, hard real-time embedded systems", International Symposium on Database Engineering and Applications, 2000.
[23] I.WENZEL, B.RIEDER, R.KIRNER and P.PUSCHNER, "Measurement-based worst-case execution time analysis", 3rd IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems ,Washington, 2005.
[24] Y.E.Ioannidis, Y.Kang, Left-deep vs Bushy Trees: An Analysis of Strategy Spaces and Its Implications for Query Optimization, ACM SIGMOD Conference on Management of Data, Denver, USA, 1991,