Performance-Polymorphic Execution of Real-Time Queries

Thomas Padron-McCarthy Tore Risch
Department of Computer and Information Science
Linköping University
S-581 83 Linköping, Sweden
E-mail: {tompa,torri}@ida.liu.se

Presented at the First Workshop on Real-Time Databases: Issues and Applications
March 7-8, 1996, Newport Beach, California, USA

Abstract

We are developing an object-oriented real-time database system that includes a relationally complete query language. Unlike conventional query optimizers, our optimizer estimates the actual time to execute a query, rather than a rough average cost estimate. For predictability the optimizer must make reliable estimates of the time to execute a query. This requires index structures that are predictable. Furthermore, our data model is object-oriented, which is used for overloading queries with respect to trade-offs between execution time and quality. Such performance-polymorphic queries have particular optimization problems.

1 Introduction

Our work deals with the particular problems that are special for query processing in a real-time database system being developed for telecom and control applications.

A real-time database with a declarative query language needs some of the following properties:

- Transactions, queries, and other operations have deadlines.
- The time to execute a query or an operation is predictable, i.e. it does not vary significantly from time to time [28].
- The system estimates the real time to execute queries and operations, analogously to what is done in real-time programming languages [11].
- Contingency plans can be defined that are queries and operations executed when the system predicts that the deadline cannot be met [3]. In contingency plans there may be timeliness trade-offs regarding data consistency, precision, and completeness.
- Queries can be defined in terms of the time to execute other queries or functions.
- Classical query optimization [25] is a combinatorical problem. It is therefore important to perform as much as possible of the optimization at compile time.

Our research platform, AMOS [4], is extensible with user-defined data structures and operations, similarly to [15]. It is now being extended with query constructs and index structures to support real-time application.

In *active databases* [30] it is possible to specify rules that monitor changes to the state of the data base. Active rules commonly consist of events that are detected, a condition that is to be monitored, and an action to be performed when the condition becomes true. Active rules have been incorporated in AMOS [26] [27]. Such facilities are important for real-time applications, e.g., to monitor combinations of sensor data and to perform actions when 'interesting' situations occur. For real-time behaviour, the rule language will need to be complemented with timeliness constraints, e.g., for rule conditions and actions. Advanced applications may require the combination of active and real-time database facilities [1] and temporal facilities.

Since time is essential in a real-time database, it normally also needs temporal facilities [18] such as data items having associated time stamps where versions of data can be accessed based on these time stamps, and where it is possible to make historical queries over the evolution of data over time.

In this paper we concentrate on query processing with real-time constraints where a quality measure and a real-time cost can be defined for queries.

2 Reliable Real-time Response Estimates

The problem of database query processing consists of compiling and optimizing queries in a non-procedural query language into efficient query execution plans. For a given query there may be several execution plans with radically different performance profiles. State-of-the-art query processors therefore are based on having a cost model for the execution speeds of each execution plan [25]. In principle the task of the optimizer is to generate the cheapest execution plan from a search space of all plausible execution plans. The cost models in traditional relational databases are based on average-case data access behavior and are often very crude, since the cost is used only for comparing execution plans, not for trying to exactly estimate the cost to execute a query.

The purpose of a real-time query optimizer is not only to generate for each query the execution plan with the estimated shortest time to execute, but also to give a reliable estimate of the real execution time. This requires modifications of the classical query processing cost models to deal with worst-case real-time behavior rather than average behavior [21]. The cost models also have to be less crude than traditional query cost models, since the estimates are not only used for comparing plans to get the cheapest one, but also for approximating the worst-case time to execute the query. These cost models must therefore reflect the actual execution times on the hardware and environment where it will be run, preferably in a portable way [17].

The cost model of a real-time query optimizer needs to be verified against its actual real-time behavior to show that it mirrors the measured real-time behavior, analogously to what can be done for real-time programming languages [11].

Another reason for unreliable cost estimates with traditional query processing is that the optimizer estimates the cost at query compile time based on the data available at that time. Some data, such as run-time supplied comparison values, are not available then and the cost estimates become unreliable. Query optimization at run time is often not a viable method either, since the cost of query optimization can be very high. Methods should be developed to deal with this problem, such as optimizing simple queries dynamically, developing methods to switch between different query plans at run time, limiting the possible data values of unbound variables in the queries, or partitioning the query into smaller fragments which can be used to calculate approximate answers [19][20].

3 Data Structures for Stable Response Time

An important property of real-time systems is that the response time should be *predictable*, and that the worst-case behaviour must not vary significantly. Traditional disk-based databases are not very suitable for real-time applications, since the access time can vary heavily (a factor of 1000 to 10000 for processing a page of data [29]) dependent on if accessed data is on disk or in the main-memory buffers of the DBMS. That problem can be overcome by designing main-memory resident databases [7], which is the approach used in our project. Still, even with main-memory databases the access time can vary heavily dependent on how well storage structures are organized. For example, if hashing is used the access time is very much dependent on how random the hash function is. It is very difficult to estimate the worst-case access time of an unpopulated hash table, as the access time of a hash table depends on the hash function, on the overflow management policies used, and on reorganization policies. We are investigating how the use of linear hashing [12] [13]and parallel execution [9] can alleviate some of these problems.

4 Performance Polymorphism

In time-critical situations it is sometimes possible to define simplified algorithms that are to be executed when the deadline approaches. These simplified algorithms have timeliness trade-offs regarding data consistency, precision, and completeness. For example, rather than computing the latest value of some value derived from the database, an extrapolation over time of old values could be used [14]. There are also techniques for finding approximate answers to queries for a significantly lower cost than completely executing the query [8][19][20].

A cost-based query processor that estimates the actual execution time can be modified to generate several execution plans of varying quality for a given real-time query, along with the expected worst-case time to execute each plan [20]. Sometimes such plans can be generated by applying predefined methods for approximating certain query operations, e. g. using methods from [8] and [14].

Furthermore, the user can provide alternative versions of a query, with different real-time performance and quality. At run-time, based on the available real time to execute the query, the system can choose the highest-quality execution plan. This is called *performance polymorphism* [10][11]. Performance polymorphism is based on the performance of the execution, in contrast to classical polymorphism which is based on types of arguments. Notice that performance polymorphism in our understanding of the term is a more

general concept than contingency plans in ECA rules [3] since performance polymorphic choices can automatically be made by the query optimizer anywhere in an execution plan.

Performance polymorphism, in a wide sense of the term, has been investigated in relational real-time systems, such as the contingency plans discussed in HiPAC [3] and the query partitioning in CASE-DB [20]. However, in order to naturally capture this and other forms of polymorphism, an object-oriented data model is advantageous. Several real-time object-oriented data models have been defined, e. g. RTSORAC [2][22]. Some real-time systems combine performance polymorphism and object orientation, such as the Flex programming language [10][11] and the ROMPP data model [31].

As far as we have been able to determine, all previous work concerning such object-oriented performance polymorphism where it is possible for the user to define multiple versions of a function or method with different performance, has concentrated on providing a programming-language interface. No declarative query language and therefore no query optimization is provided.

Our approach is to introduce performance polymorphism into AMOS, which is an object-oriented database system with a relationally complete query language. Our schemas are built up by types (i.e. classes) and functions (i.e. methods), similar to IRIS [5]. The functions in AMOS are defined using an object-oriented query language and are optimized using both traditional and specialized query optimization techniques [15]. AMOS functions can be defined in terms of other AMOS functions.

Analogously to regular polymorphic functions, the system needs a method to resolve which *resolvent function* to invoke in a performance polymorphic function call. For example, a simple strategy is to always invoke the resolvent function with the highest quality (and slowest execution). When this causes the overall execution time constraints of the compiled function (i.e. query) to be exhausted the system needs a strategy to decrease the quality of the resolvent function calls.

Another trade-off is whether to choose the resolvent at query compile time (early binding) or at run time (late binding). The first is the preferred choice, since it makes it easier to statically estimate the performance of a function. However, in some cases the latter is necessary, and then strategies are needed to estimate the performance of late bound performance-polymorphic function calls. The query optimizer should automatically choose early binding when possible. When late binding cannot be avoided the system can optimize each resolvent separately and then estimate the time to execute the performance polymorphic call in terms of the real-time to execute its resolvents, similar to [6].

An interesting problem is how to optimize execution plans with several performance polymorphic function calls. In that case the optimizer can make many different trade-offs between quality and execution time at different function calls.

5 Summary and ongoing work

Our aim is to provide performance polymorphism in an object-oriented database system with a relationally complete declarative query language. In this paper we have outlined the requirements and problems to be solved. We are currently modifying a dynamic-programming-based query optimizer to use a real-time cost model with quality trade-offs, rather than a traditional cost model. We are also developing and investigating main-memory data structures with predictable and stable response times. We will incorporate some of these data structures into our object-oriented database, and develop a cost model for them. We will also provide primitives for the user to specify time and quality requirements in the object-oriented query language. We plan to verify our approach by applying it to a realistic real-time measurement example.

Acknowledgements

This project is funded by NUTEK, the Swedish National Board for Industrial and Technical Development, as part of ISIS, the Competence Center for Integrated Systems for Control and Information.

6 References

[1] Berndtsson M., Hansson J. (Eds.): *Proceedings of the First International Workshop on Active and Real-Time Database Systems (ARTDB-95)*, Springer-Verlag, 1995

[2] Cingiser DiPippo L., Wolfe V. F.; Object-based Semantic Real-Time Concurrency Control, *14th IEEE Real-Time Systems Symposium*, Raleigh-Durham, NC, December 1993, pp 87-86

[3] Dayal U., Blaustein B., Buchmann A., Chakravarthy U., Hsu M., Ledin R., McCarthy D., Rosenthal A., Sarin S.: The HiPAC Project: Combining Active Databases and Timing Constraints, *SIGMOD RECORD*, Vol. 17, No. 1, March 1988, pp 51-70

[4] Fahl G., Risch T., Sköld M.: AMOS - An Architecture for Active Mediators, *International. Workshop on Next Generation Information Technologies and Systems (NGITS '93)* Haifa, Israel, June 1993, pp 47-53

[5] Fishman D. H. et al: Overview of the Iris DBMS, *Object-Oriented Concepts, Databases, and Applications*, ACM

press, Addison-Wesley Publ. Comp., 1989

[6] Flodin S., Risch T.: Processing Object-Oriented Queries with Invertible Late Bound Functions, In *Proceedings of the 21st International Conference on Very Large Databases*, Zurich, Switzerland, September 11-15, 1995

[7] Garcia-Molina H: Main Memory Database Systems: An Overview, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 4, No. 6, December 1992, pp 509-516

[8] Hou W., Ozsoyoglu G., Taneja B.K.: Processing Aggregate Relational Queries with Hard Time Constraints, *Proceedings of the ACM SIGMOD Conferenc*e, 1989

[9] Karlsson J., Litwin W., Risch T.: LH*lh : A Scalable High Performance Data Structure for Switched Multicomputers. *The 5th International Conference on Extending Database Technology (EDBT'96)*, Avignon, France, March 1996.

[10] Kenny K. B., Lin K.-J.: Structuring large real-time systems with performance polymorphism, *Proc. Real-Time Systems Symposium*, Orlando, FL, 1990, pp 238-246

[11] Kenny K. B., Lin K.-J.: Building Flexible Real-Time Systems Using the Flex Language, *IEEE Computer*, May 1991, pp 70-78

[12] Larsson P.-Å.; Dynamic Hash Tables, *Communications of the ACM*, Vol. 31, No. 4, April 1988

[13] Litwin W: Linear hashing: A new tool for file and table addressing, In *Proceedings of the 6th Conference on Very Large Databases*, New York, 1980, pp 212-223

[14] Liu, J. W. S., Lin K.-J., Shih, W.-K., Yu A. C.: Algorithms for Scheduling Imprecise Computations, *IEEE Computer,* May 1991, pp 58-68

[15] Litwin W., Risch T.: Main Memory Oriented Optimization of OO Queries Using Typed Datalog with Foreign Predicates, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 4, No. 6, December 1992

[16] Loborg P., Risch T., Sköld M., Törne A., Active Object Oriented Databases in Control Applications, *19th Euromicro Conference of Microprocessing and Microprogramming*, vol. 38, 1-5, pp 255-264, Barcelona, Spain 1993

[17] Nilsen K., Rygg B.: Worst-Case Execution Time Analysis on Modern Processors, *ACM PLDI Workshop on Languages, Compilers, and Tools for Real-Time Systems*, La Jolla, CA, June 1995

[18] Ozsoyoglu G., Snodgrass R. T.: Temporal and Real-Time Databases: A Survey, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 7, No. 4, August 1995

[19] Ozsoyoglu G., Du K., Guruswamy S., and Hou W.: Processing Real-Time, Non-Aggregate Queries with Time-Constraints in CASE-DB, *Proc. IEEE Data Engineering Conf 8*, Tempe, AZ, Feb. 1992

[20] Ozsoyoglu G., Guruswamy S., Du K. and Hou W.: Time-Constrained Query Processing in CASE-DB, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 7, No. 6., December 1995

[21] Pang H. H., Carey M. J., Livny M., Managing Memory for Real-Time Queries, *Proceedings of the ACM SIGMOD Conference*, 1994

[22] Prichard J. J., Cingiser DiPippo L., Peckham J., Wolfe V. F.: RTSORAC: A Real-Time Object-Oriented Database Model, *Proc. 5th Int. Conf. on Database and Expert System Applications (DEXA 94)*, Athens, Greece, December 1994, pp 601-610

[23] Ramamritham K.: Real-Time Databases, *Distributed and Parallel Databases 1* (1993), pp 199-226

[24] Risch T., Sköld M.: Active Rules based on Object Oriented Queries, *IEEE Data Engineering bulletin*, Vol. 15, No. 1-4, Dec. 1992, pp 27-30

[25] Selinger P. G., Astrahan M. M., Chamberlin D. D., Lorie R. A., Price T.G.: Access Path Selection in a Relational Database Management System, In *Proceedings of the ACM-SIGMOD Conference*, Boston, MA, June 1979, pp 23-34

[26] Sköld M., Risch T.: Using Partial Differencing for Efficient Monitoring of Deferred Complex Rule Condition, *The 12th International Conference on Data Engineering (ICDE'96)*, New Orleans, Louisiana, February 1996

[27] Sköld M., Falkenroth E., Risch T.: Rule Contexts in Active Databases - A Mechanism for Dynamic Rule Grouping. In *Rules in Database Systems* (RIDS 95), Athens, Greece, September 25-27, 1995, Springer Lecture Notes in Computer Science, ISBN 3-540-60365-4, pp 119-130, 1995

[28] Stankovic J. A.: Misconceptions About Real-Time Computing, *IEEE Computer*, Vol. 21, No. 10, October 1988

[29] Stone H. S.: *High-Performance Computer Architecture*, 3d Ed, Addison-Wesley 1993

[30] Widom J., Ceri S. (Eds.): *Active Database Systems - Triggers and Rules For Advanced Database Processing*, Morgan Kaufmann, 1996

[31] Zhou L., Rundensteiner E. A., Shin K. G.: *Schema Evolution for Real-Time Object-Oriented Databases*, Technical Report CSE-TR-199-94, University of Michigan, March 1994