Replication Gives High Performance Query Processing in Relational Models Extended with Transaction Time

Christian S. Jensen* Leo Mark

Department of Computer Science

University of Maryland

College Park, MD 20742

April 1990

In our research area, query processing in the relational model extended with transaction time [1, 2, 3, 4], replication of data combined with differential computation techniques is fundamental for the achievement of high performance.

Our research project can be described as follows.

We have defined a static rollback relational data model with tuple time-stamping [5, 6]. For each relation defined by the user, the system maintains a backlog that records all requests for changes to its user-defined relation. The backlogs contain all the data necessary for answering any query without replication. We have designed an extended query language that allows access to previous states of relations (including backlogs) and allows to ask queries about the change history of relations. We also have developed vacuuming facilities to control otherwise ever-growing amounts of data in a transaction time database [7].

To support queries efficiently, we maintain a *cache* of views stored as either actual *data* (true replication) or as *pointer structures* (pseudo replication). A special *cache index* is used for fast identification of cache entries potentially useful as outsets in *differential* (i.e. incremental or decremental) computations of queries. The cache index is augmented with *statistics* about the cache entries. We use *state transition networks* for enumerating query plans. The number of alternative query plans is reduced by *pruning rules*, and plan selection can be done using the statistics of the cache index in conjunction with either *dynamic programming* techniques or *heuristic* techniques (e.g. the A* algorithm). Our framework is the first to integrate pointer/data storage and incremental/decremental/re-computation.

Our replicated data management problems can be characterized as follows.

1. Which statistics should be kept about cached views?

*Supported by Aalborg University, Denmark. Correspondence via e-mail to jensen@brillig.umd.edu

The cost of maintaining statistics should be lower than the advantages gained from using them (see below). The following are possible candidate statistics: Fixed/time-dependent, cardinality, tuple size, up-to-date status, number of times used, when materialized, first time used, last time used, initial computation cost, and cost of usage.

Closely related problem are how precise statistics we need (precision costs) and how the statistics are efficiently maintained.

2. Which views should be cached, and should data or pointer caching be chosen?
   This is one possible application of the mentioned statistics. Statistics reflecting query patterns should be given special interest.

3. For which cached views should eager propagation of updates be chosen? For which views are a threshold triggered strategy appropriate, and for which should a lazy strategy be chosen?
   There are obvious advantages to use eager update for views representing current states of relations, and it is likely that threshold triggered and lazy update should be adopted for almost all other views. Statistics will again play a central rôle.

4. How are statistics used for cache management? What is the significance of individual kinds of statistics? Are simple protocols from buffer management (e.g. Least Recent Used) appropriate?

In the past, techniques of view materialization and incremental computation (data [8], pointers [9, 10, 11]) have been used in centralized, non-temporal relational databases. Recently, efforts have been put into the update of distributed materialized views [12, 13, 14, 15], also for non-temporal databases. The introduction of view-materialization together with incremental computation has lead to significant gains in performance.

The use of the techniques is largely unexplored in temporal databases (transaction time and logical time) where they have a great potential (cf. [16]). Furthermore, in transaction time databases as ours, we expect techniques of view materialization and differential computation to be essential for the achievement of high performance. Other alternatives such as no caching (and high computation costs) or no backlogs (and storage of each individual state instead) are unrealistic for large databases. In our setting, some views never get outdated and it is possible to predict (with arbitrarily fine precision) the outdatedness of some other views—this is not the case in the standard relational model.

References

[1] Christian S. Jensen, Leo Mark, and Nick Roussopoulos. Incremental Implementation Model for Relational Databases with Transaction Time. Technical report, CS-TR-2275, UMIACS-TR-8963, Department of Computer Science. University of Maryland, College Park, MD 20742, June 1989.

[2] Christian S. Jensen and Leo Mark. Queries on Change in an Extended Relational Model. Technical report, CS-TR-2299, UMIACS-TR-89-80, Department of Computer Science. University of Maryland, College Park, MD 20742, August 1989. To appear in *IEEE Transactions on Knowledge and Data Engineering*.

[3] Christian S. Jensen, Leo Mark, Nick Roussopoulos, and Timos Sellis. A Framework for Efficient Query Processing Using Caching, Cache Indexing, and Differential Techniques in the Relational Model Extended with Transaction Time. Technical report, R-89-45, The University of Aalborg, Institute of Electronic Systems. Dept. of Math. and Comp. Sci., Frederik Bajers Vej 7E, DK-9220 Aalborg Øst, Denmark, December 1989.

[4] Christian S. Jensen, Leo Mark, Nick Roussopoulos, and Timos Sellis. Using Caching, Cache Indexing, and Differential Techniques to Efficiently Support Transaction Time. Technical report, CS-TR-2413, UMIACS-TR-90-25, Department of Computer Science. University of Maryland, College Park, MD 20742, February 1990. Submitted for publication.

[5] Edwin McKenzie and Richard Snodgrass. Extending the Relational Algebra to Support Transaction Time. In *Proceedings of the ACM SIGMOD '88*, pages 467–477, 1988.

[6] Richard Snodgrass and Ilsoo Ahn. A Taxonomy of Time in Databases. In *Proceedings of the ACM SIGMOD '85*, pages 236–246, 1985.

[7] Christian S. Jensen and Leo Mark. A Framework for Vacuuming Temporal Databases. Technical report, CS-TR-2516, UMIACS-TR-90-105, Department of Computer Science. University of Maryland, College Park, MD 20742, August 1990. Submitted for publication.

[8] Kathryn C. Kinsley and James R. Driscoll. Dynamic Derived Relations Within the RAQUEL II DBMS. In *Proceedings of the 1979 ACM Annual Conference*, pages 69–80, October 1979.

[9] Nick Roussopoulos. View Indexing in Relational Databases. *ACM Transactions on Database Systems*, 7(2):258–290, June 1982.

[10] Nick Roussopoulos. The Logical Access Path Schema of a Database. *IEEE Transactions on Software Engineering*, 8(6):563–573, November 1982.

[11] Nick Roussopoulos and Hyunchul Kang. Principles and Techniques in the Design of ADMS±. *IEEE Computer Magazine*, 19(12):19–25, December 1986.

[12] Arie Segev and Weiping Fang. Optimal Update Policies for Distributed Materialized Views. Technical report, LBL-26104, Computer Science Research Department, Lawrence Berkeley Laboratory, 1 Cyclotron Road, Berkeley, California 94720, 1989.

[13] Arie Segev and Jooseok Park. Updating Distributed Materialized Views. LBL 24882, School of Business Administration and Computer Science Research Department, Lawrence Berkeley Laboratories, University of California, Berkeley, CA 94720, May 1988.

[14] Arie Segev and Weiping Fang. Currency-Based Updates to Distributed Materialized Views. In *Proceedings of the Sixth International Conference on Data Engineering*, pages 512–520, February 1990.

[15] Arie Segev and Jooseok Park. Maintaining Materialized Views in Distributed Databases. In *Proceedings of the Fifth International Conference on Data Engineering*, pages 262–270, February 1989.

[16] Leslie Edwin McKenzie. An Algebraic Language for Query and Update of Temporal Databases. Ph.D. Dissertation, 88-050, The University of North Carolina at Chapel Hill, Department of Computer Science, CB 3175, Sitterson Hall, Chapel Hill, NC 27599-3175, October 1988.