Query Optimization in the ADDS Multidatabase System

T. Reyes, W. Lee, P. Olson, G. Thomas, G. Thompson, & B. Vassaur
Amoco Production Company

Abstract

The range of query optimization alternatives that can be incorporated into a heterogeneous distributed database system is quite large. There is an abundance of query optimization algorithms but a dearth of knowledge on how to choose, organize, implement, and put together query optimization techniques into an industrial–strength multidatabase system. This paper describes the query optimization strategies that have been implemented in ADDS, a multidatabase system that allows new applications to retrieve data from pre–existing heterogeneous databases without disrupting existing applications. ADDS' query optimizer uses only high–payoff but low–cost algorithms for reducing data retrieval and transmission costs. These techniques include composite schema replication, doing relational operations at the data sites, join and semijoin optimization, doing restrictions as early as possible, and common subquery elimination. Performance figures showing dramatic reductions in network traffic and substantial improvements in query execution speed indicate the soundness of our approach. The choice, organization, and sequencing of the chosen query optimization techniques are our key contributions.

0. Introduction

Many organizations have major databases and database applications built on top of IMS or some other nonrelational DBMS. With the widely acknowledged advantages of the relational model, almost all new database applications are being written on top of some relational DBMS. Many of these new applications are not being built from scratch and often need access to the existing nonrelational databases. The Amoco Distributed Database System (ADDS) [1 ,2] addresses this need by presenting new applications a uniform relational interface to pre–existing heterogeneous databases without requiring any modifications to the target databases or to the application programs that maintain these databases.

A number of research prototypes with goals similar to that of the ADDS system have reached various stages of development in the past ten years. These prototypes include Dataplex[4], Multibase [8], DDTS [3], and OMNIBASE [10]. To the best of our knowledge, these systems have remained prototypes and have not been deployed in live applications. Newer products such as INGRES/STAR [6] provide only partial integration of pre–existing databases using gateways to access IMS databases but allow distributed queries only to homogeneous (INGRES) databases. Other efforts at integrating heterogeneous databases such as the ISO OSI Remote Database Access Standards [5] cover only relational systems and ignore existing nonrelational databases.

The ADDS system has been released for use as the data retrieval component of a large reservoir engineering support system [13] and a generalized electronic toolbox since 1988. The experience gained by users with the first release provided the motivation for the development of a second release in 1989 which included the optimizations discussed in this paper. The rest of the paper is organized as follows: Section 1 provides a brief overview of the ADDS system. Section 2 discusses the placement of composite schemas. Section 3 outlines the reasons why query optimization can not be ignored in a multidatabase system. Section 4 contains our key contributions: a well organized implementation of carefully chosen query optimization techniques. Section 5 presents experimental results on the effectiveness of this query optimization approach. Section 6 contains our conclusions.

1. Overview of the ADDS System

An ADDS system [11] consists of one or more sites connected by a communications network as shown in Figure 1.

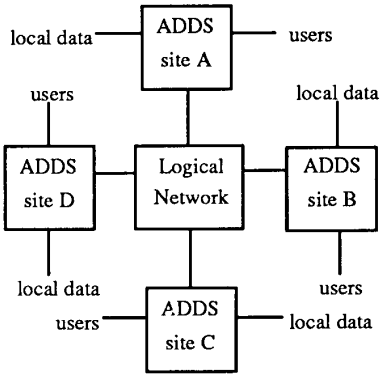

Figure 1.
An ADDS System

Each site has a copy of the composite schemas and the following processes: a task master, user interface processes, a temporary relations manager, a directory manager, a query compiler, and one or more servers that interface to local DBMSs as shown in Figure 2. Administrators of new applications define a composite schema [2] of the logical and physical schemas of the data that will be of interest to these applications. A composite schema provides the basis for mapping and materializing logical relations from the physical target databases. Once a composite schema is defined, users can submit queries against any one or more logical relations in that schema. For this purpose, ADDS supports two relational query languages: SQL and ADDS [9].

The task master is the focal point of control and scheduling at each site. Upon receipt of a query from a user interface process, the task master forwards that query to the query compiler for processing. The query compiler analyzes, optimizes, and decomposes the query into one or more single–site subqueries which are sent back to the task master for dispatching. The task master classifies subqueries

into nonlocal or local subqueries. Nonlocal subqueries are forwarded to the task master at the target site. Local subqueries are assigned to the next available server at the local site.

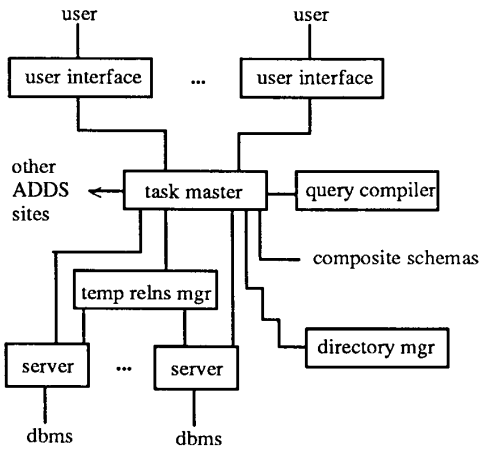

Figure 2. ADDS Process Architecture

The assigned server translates the subquery into a DBMS–specific form that is then executed to retrieve the desired data. Upon completion, the server tells the task master of its success. The task master instructs the temporary relations manager to save the server–retrieved data for use by subsequent subqueries. The temporary relations manager handles the passing of intermediate results from one subquery step to the next one until the final results are produced for eventual disposition by the original query owner.

2. Directory Placement

The placement of the composite schemas is a very important issue that strongly impacts query processing efficiency. Composite schemas can be placed in: (a) a central directory site, (b) partially replicated at participating sites, or (c) fully replicated at all sites. A central directory site simplifies directory maintenance and may be a viable alternative if most target databases are at the directory site. Fully replicated composite schemas eliminate all network traffic associated with referencing composite schemas. Partially replicated schemas save space and make directory updates slightly cheaper than in the fully replicated case but may still incur nonlocal schema references if a user logs on and submits queries at a nondirectory site. In the current ADDS system, composite schemas are fully replicated. The updating and replication of composite schemas are handled by a directory manager at each site.

3. Why Query Optimization Can Not Be Ignored?

A relationial model based multidatabase system can not ignore query optimization issues. Most relational queries can be formulated in many different ways [12] and the performance differences between these alternative formulations are often orders of magnitudes apart. In addition, the geographical distribution of target databases introduces another dimension of variability represented by the choices of where and how joins and other relational operations can be performed. In the case of ADDS, the widely varying capabilities of the different target DBMSs must also be considered as a constraint on the subqueries that can be given to weak target DBMSs and as an optimization resource in the case of strong relational target DBMSs.

4. Query Optimization in ADDS

The following description of the ADDS query optimizer depicts only the end result of a long series of deliberate implementation choices. Its simplicity gives no indication of the many potential pitfalls that can easily result in drastically reduced performance during query compilation or execution.

Because of its goal of co–existing with and not disrupting existing applications, ADDS leaves all storage access path optimizations to the target DBMSs and concentrates instead on generating an efficient sequence of single–site subqueries that can be handled by the servers that interface with the target DBMSs. The ADDS query optimizer does not try to improve or replace any optimization capabilities in the target DBMSs. Rather it tries to produce a set of single–site subqueries that are efficiently executable by the target DBMSs.

Each relational query is translated into an internal tree representation on which the following optimizing transformations are made:

1. A disjunctive query with the *DISTINCT* option may be transformed into a UNION of subqueries with the *DISTINCT* option on the UNION operator. Similarly, a disjunctive query with the *ALL* option may be transformed into a UFID of subqueries with the *ALL* option on the UFID operator. The UFID operator is a cardinality–preserving variant of the UNION relational operator. The UFID operator is defined as: R1 UFID R2 <==> R1 UNION *ALL* (R2 DIFFERENCE R1), where R1, R2 are union–compatible relations whose attributes are in one–to–one correspondence such that corresponding attributes are defined on the same domain [9].

   Either transformation gets rid of the OR condition and increases the number of eligible selection conditions that can be performed earlier, i.e., that can be pushed down the subquery trees. This transformation is not always desirable especially if the entire query can be done by one target DBMS. Therefore, this transformation is applied only if the query spans more than one target DBMS.

2. A query containing a sequence of associative and commutative binary relational operations is collapsed into an n–ary query tree. For example, the join sequence (JOIN r1 (JOIN r2 r3)) is collapsed into (JOIN r1 r2 r3). This transformation lays the groundwork for subsequent steps to discover an efficient evaluation order for these n–ary operations.

3. Selection conditions are pushed up the query tree to allow the creation of new derivative selection conditions. For example, the query "JOIN * (r1; SELECT * FROM r2 WHERE r2.x > 0) WHERE r1.x = r2.x" may be transformed into "JOIN * (r1; r2) WHERE r2.x > 0 AND r1.x = r2.x AND r1.x > 0". The selection condition "r1.x > 0" is derived from the other two predicates.

4. Selection conditions and their derivatives are pushed down the query tree. For example, the query "JOIN * (r1; r2) WHERE r2.x > 0 AND r1.x = r2.x AND r1.x > 0" may be transformed into "JOIN * (SELECT * FROM r1 WHERE r1.x > 0; SELECT * FROM r2 WHERE r2.x > 0) WHERE r1.x = r2.x".

5. N–ary operations are transformed into a tree of binary operations. N–ary UNIONs are transformed into a complete binary tree of UNIONs to maximize potential parallelism. Finding the most desirable evaluation order for n–ary JOINs is a difficult problem. The ADDS query optimizer uses a simple heuristic to arrive at a desirable solution quickly. Given a list of candidate join operands, it repeatedly applies the following preference hierarchy to pick out two join operands whose result is then added to the candidate list until only two candidates are left:

a. Equi–join on 2 keys, i.e., both relations have performance keys on the columns being joined;

b. Equi–join on 1 key, i.e., one of the relations has a performance key on its join column;

c. Equi–join that is amenable to semijoin optimization, i.e., at least one of the join operands is a SELECT subquery that retrieves data from one target database;

d. Any equi–join;

e. Any join;

f. Cartesian product only as a last resort.

6. An equi–join involving at least one SELECT subquery undergoes semijoin optimization. Semijoin optimization determines the most desirable equi–join condition and the semijoin direction. For example, given the query "JOIN * (r1; r2) WHERE r1.x = r2.x AND r1.y = r2.y", the semijoin optimizer determines whether to use "r1.x = r2.x" or "r1.y = r2.y" as the semijoin condition. In addition, it determines whether r1 or r2 will have its subquery search condition augmented by a semijoin predicate. If the chosen condition is "r1.x = r2.x" and its direction is towards r2, the following actions will be taken at query execution time:

   a. The temporary relations manager will eliminate null & duplicate values on the joining column (r1.x) of r1's subquery result and make this set of semijoin values available to the server assigned to do the r2 subquery.

   b. The server for r2 uses the temporary relations manager supplied semijoin values for r1.x to modify r2's subquery predicate into something like "SELECT * FROM r2 WHERE r2.x IN (value1, value2, ...)".

   The semijoin optimizer applies a cost estimation strategy to try to pick a semijoin condition and direction that would yield the greatest data reduction. Semijoin optimization finds its greatest use in reducing the amount of data retrieved from huge IMS databases.

7. The partially optimized binary query tree is placed in canonical order form to facilitate the detection of common subqueries. This transformation imposes a uniform ordering on commutative operations. For example, the query "UNION * (JOIN * (r2; r1); DIFF * (r4; r3))" and the query "UNION * (DIFF * (r4; r3); JOIN * (r2; r1))" can both be transformed into their equivalent canonical order form of: "UNION * (DIFF * (r4; r3); JOIN * (r1; r2))".

8. Redundant subqueries are detected, eliminated, and replaced with references to their previously computed canonical representative subqueries. Two subqueries are considered equal if and only if:

   a. Their query predicates including any semijoin predicates, projected fields, and sort specifications, are formally equal, and

   b. Their subquery operands or source relations are equal.

   Under formal equality, the correlation names of column references need not match provided the same correlation name is not bound to two different relation names in the resulting optimized query. Moreover, a subquery S1 whose projected fields are a subset of the projected fields of another subquery S2 is still considered equal to S2 if both S1 and S2 do not require elimination of duplicate tuples.

   A bottom–up traversal of the query tree examines each node testing it for equality against nodes that have been seen before. New nodes become canonical representative nodes and have their projected fields expanded to increase the chances of future matches. Canonical representative nodes are kept in a hash table to facilitate efficient subquery matching. A semijoin–receiving common subquery node is declared redundant only if its parent join–type node is declared redundant. This limitation recognizes the fact that no subquery step can receive semijoin values from more than one source.

9. Projections are pushed down the query tree to further reduce the amount of retrieved data and shrink the temporarily expanded list of projected fields in the surviving common subquery nodes. At the same time, the *DISTINCT* keyword is propagated down the query tree to assure the removal of duplicate tuples in intermediate results.

10. Finally, the optimized query graph is broken up into schedulable single–site subqueries. This decomposition step decides where JOINs and other relational operations are done. In the simplest case where all the leaf nodes in the query graph belong to one site, the entire query graph is packaged as one big query for that site. At the other extreme is the case where each leaf node belongs to a different site and each node, whether a leaf or an internal node, becomes a schedulable subquery. In the current ADDS system, all schedulable internal nodes representing relational operations are done at the original query submission site.

    This decomposition strategy deliberately makes the false assumption that all target DBMSs at all sites can handle all relational query operations. This strategy allows ADDS to exploit the optimization capabilities of strong target DBMSs such as DB2 and SQL/DS by passing them maximal size subqueries. The servers that interface to weak target DBMSs understand this strategy and will break up a complex query into a set of simpler subqueries. After decomposing a complex query, the server sends the set of simpler subqueries back to its local task master for dispatching. Eventually, a complex query is broken up into:

    a. retrieval subqueries that can be handled by even the weakest of target DBMSs, and

    b. relational operation subqueries that can be handled by the local ADDS relational servers.

    The net effect is to have most relational operations performed at the target data sites.

5. Experimental Results

To provide an indication of the effectiveness of some of these optimizations, two sets of experiments were run. The first set focused on measuring reductions in network traffic and the second set focused on measuring improvements in query execution speeds. To measure differences in network traffic that can be associated with shifting from a centralized directory to a fully replicated directory, a simple SELECT query was run to retrieve data from a nonlocal target database. The normalized results are summarized in Table I. The amount of network traffic reduction is very sensitive to the size of the composite schema and to the amount of data actually retrieved. However, an order of magnitude reduction in network traffic is not unreasonable because composite schema sizes of 220 Kbytes are not unusual among the current ADDS applications. The other two entries in Table I summarize the results of running: (a) a JOIN query that retrieves data from two databases at a nonlocal site, and (b) a complex 29–step query containing 12 common subquery steps.

Table I. Normalized Reduction in Network Traffic

Optimization Tactic	Unoptimized	Optimized
Schema replication	9.8	1.0
Do join at data site	25.4	1.0
Common subquery elimination	31.2	1.0

To factor out the radical effects of changing from a centralized to a replicated directory structure, the second set of experiments were conducted under an artificially localized system where all target sites and databases are located on the same physical computer system. This set of experiments measured only the improvements to the query processing speeds without any network communications overhead. Therefore, the normalized results shown in Table II represent a lower bound on the speed improvements that can be expected for the same queries that were used in the first set of experiments.

Table II. Normalized Query Execution Speedups Under a Fully Localized System

Optimization Tactic	Unoptimized	Optimized
Schema replication	1.0	2.0
Do join at data site	1.0	2.1
Common subquery elimination	1.0	3.1

A careful implementation of the previously described optimizations plus an enhancement of existing code has also resulted in approximately the same query compilation speeds under both releases of ADDS.

6. Summary and Conclusions

Most of the research on the problems of implementing heterogeneous distributed database systems has been done on prototype systems that were not designed to support real live industrial applications. The ADDS multidatabase system is one of the few systems that have actually been deployed in industrial applications. Despite an abundance of query optimizatioin algorithms, the state of the practice in putting it all together into a working system seems to continue to lag. The key contribution of this paper is the description of a query optimizer organization that not only doubles query execution speeds and reduces network traffic by an order of magnitude but also achieves good query compilation rates (about 1 join query per second on an 800–user 3090 mainframe).

References

1 Breitbart, Y., Tieman, L., "ADDS – Heterogeneous Distributed Database System", in Distributed Data Sharing Systems, Schreiber, F., Litwin, W. (editors), North Holland, 1984, pp. 7–24.

2 Breitbart, Y., Olson, P., Thompson, G., "Database Integration in a Distributed Heterogeneous Database System", Proceedings of the International Conference on Data Engineering, Los Angeles, CA, February 5–7, 1986, pp. 301–310.

3 Ceri, S., Pelagatti, G., Distributed Databases Principles and Systems, McGraw–Hill, New York, 1984, pp. 375–381.

4 Chin, Wan Chung, "Dataplex: An Access to Heterogeneous Distributed Databases", CACM vol 33, no 1, Jan 1990, pp. 70–80.

5 Corfman, M., "Open SQL – ISO OSI Remote Database Access Standards", panel presentation at 14th International Conference on Very Large Data Bases, Long Beach, CA, 1988.

6 Date, C., A Guide to INGRES, Addison–Wesley, Reading, Massachusetts, 1987, pp. 331–339.

7 Lee, W., Olson, P., Thomas, G., Thompson, G., "A Remote User Interface for the ADDS Multidatabase System", in Proceedings of the Second Oklahoma Workshop on Applied Computing, University of Tulsa, March 18, 1988, pp. 194–204.

8 Landers, T., Rosenberg, R., "An Overview of Multibase", Proceedings of the 2nd International Symposium on Distributed Databases, Berlin, West Germany, September 1982.

9 Reyes, T., Thomas, G., Thompson, G., "Implementation of a Multi–Language Query Compiler for a Heterogeneous Multidatabase System", in Proceedings of the Workshop on Applied Computing '89, Stillwater, OK, March 30–31, 1989, pp. 34–40.

10 Rusinkiewicz, M., Loa, K., Elmagarmid, A., "Distributed Operation Language for Specification and Processing of Multidatabase Applications", CSD–TR–834, Purdue University, Computer Sciences Department, December 1988.

11 Thompson, G., Breitbart, Y., "Design Issues in Distributed Multidatabase Systems", in Proceedings of Workshop on Applied Computing, Stillwater, OK, October 10, 1986, pp. 38–46.

12 Ullman, J., Principles of Database and Knowledge–Base Systems, Volume II: The New Technologies, Computer Science Press, Rockville, MD, 1989, pp. 633–725.

13 Vassaur, B., "RESS (Reservoir Engineering Support System) Test", Technical Report F86–C–0006 (86142ART0066), Amoco Production Company, Tulsa Research Center, May 1986.