Global Query Processing and Optimization in the CORDS Multidatabase System [*]

Qiang Zhu
Department of Comp. and Inf. Science
University of Michigan - Dearborn
Dearborn, MI 48128

Per-Åke Larson [†]
Department of Computer Science
University of Waterloo
Waterloo, Canada N2L 3G1

Abstract

A multidatabase system (MDBS) integrates information from autonomous pre-existing local databases managed by heterogeneous local database management systems in a distributed environment. To achieve good overall system performance, efficient global query processing and optimization techniques are required in the MDBS. In this paper, several new techniques such as query sampling, query probing, and piggybacking are introduced. These techniques were designed for an MDBS called the CORDS Multidatabase System. The paper gives an overview of the global query optimizer in the system, including its architecture and some critical implementation techniques.

Keywords: multidatabase system, distributed query processing, global query optimization, cost estimation, database system architecture

1 Introduction

The CORDS project was a collaborative research project involving the IBM Centre for Advanced Studies, IBM Research, and several North American universities. The main goal of the CORDS project was to provide users with an environment for designing, developing and managing distributed applications.

Virtually all applications require access to shared, persistent data. Frequently, such data is stored in and managed by a database system (DBS). It would be naive to assume that all data are managed by a single DBS; more typical would be a scenario where data are managed by multiple, heterogeneous DBSs distributed in a network.

To facilitate access to multiple pre-existing databases managed by heterogeneous DBSs, multidatabase systems (MDBS) have been proposed by database researchers[2, 8] (see Figure 1). An MDBS can only interact with local DBSs at their external user interfaces. A key feature of an MDBS is the local autonomy that individual DBSs retain to serve existing applications.

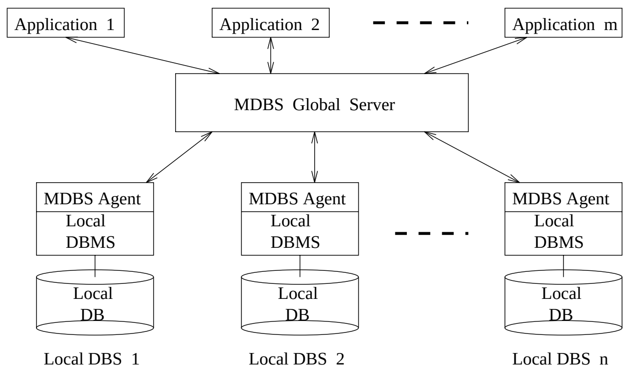

Figure 1: Multidatabase Services

As part of the CORDS project, a multidatabase system prototype, called CORDS-MDBS, was developed jointly by the University of Waterloo and Queen's University.

A user who interacts with an MDBS is called a global user. An MDBS provides a global user with a simple and consistent view of database access. A global user can issue a global query for retrieving data from several local databases. To achieve good overall system performance, global query optimization is needed.

A traditional distributed database system (DDBS) also integrates data from multiple local databases. However, distributed database systems were designed for a homogeneous environment; for instance, all sites use the same data model. Local autonomy was not emphasized in such systems. Therefore, although global

[*]Research supported by IBM Toronto Laboratory and Natural Sciences and Engineering Research Council (NSERC) of Canada

[†]Current address: Microsoft Corporation, One Microsoft Way, Redmond, WA 98052–6399

query optimization in an MDBS has some similarities to that in a traditional DDBS, there are some new challenges[5, 8, 9]:

(a) *Some local information needed for global query optimization, such as local cost parameters, may not be available at the global level.* Because of local autonomy, a local DBS may not expose all its information at the global level. Lack of local information increases the difficulty for the global query optimizer to choose a good strategy for executing a global query. A traditional distributed database management (DDBMS), however, runs at all local sites and can make use of both global and local information to perform global query optimization.

(b) *Implementation of local DBMSs cannot be changed.* In a traditional DDBS, all sites can cooperate at the internal level via its DDBMS. Most existing distributed query optimization techniques rely heavily on this fact. However, in an MDBS, the cross-system operations among local databases are expressed at the external interfaces of local DBMSs rather than through low level procedure calls. The MDBS cannot change the implementation of any local DBMS. Therefore many conventional distributed query optimization techniques may not be suitable for an MDBS.

(c) *Several levels of translations for requests and data are required.* Local DBSs may have different representations, namings, semantics, and data models for their data. The problem of efficient data integration during query processing is one of the most difficult problems for developing an MDBS. Efficient request transformations are also required during query optimization due to the possible heterogeneous local query languages.

(d) *Different and changing local capabilities are assumed.* Unlike a traditional DDBS that usually has the same capability for all sites, an MDBS may have different local capabilities at different sites. Furthermore, a local DBS can set the degree of sharing of its functionality and resources with others, and it may change its decisions without advance notice. Hence global query optimization becomes much more difficult.

(e) *More constraints need to be considered during global query optimization.* In an MDBS, there is less freedom for the global query optimizer to decide where to transfer data to and where to perform an operation.

Among the above challenges, the first challenge is crucial. Recently, several researchers have investigated this issue. In [3], Du *et al.* proposed a calibration method to deduce necessary local information. The idea is to construct a special local synthetic calibrating database and then run a set of special queries against this database. Cost metrics for the queries are recorded and used to deduce the coefficients in the cost formulas for the access methods supported by the underlying local database system by using the properties of the database and queries. Among several shortcomings of this method, a major one is that the access method for a query has to be known (violating local autonomy) in order to choose the right cost formula to estimate the cost. To overcome the shortcomings of Du *et al.*'s method, we proposed a query sampling method based on statistical analysis in [11] and a fuzzy query optimization approach based fuzzy set theory in [12, 13].

Most research on distributed query optimization done so far is for traditional DDBSs. Little work has been done on query optimization in an MDBS. To test the feasibility of new techniques for global query optimization in an MDBS, a global query optimizer was designed in the CORDS-MDBS. In this paper, we describe the query processing and optimization techniques adopted in the global query optimizer.

2 Query Optimization Techniques

As mentioned before, lack of query optimization information is the major problem for performing global query optimization in an MDBS. Several techniques such as query sampling, query probing, and piggybacking were proposed to solve this problem in the CORDS-MDBS. The basic ideas of these techniques are outlined in this section.

2.1 Query Sampling Technique

A global query is usually first decomposed into several local queries that can performed at relevant local DBSs. The way to decompose a global query is not unique. To achieve a good decomposition of a global query, the costs for performing local queries at local DBSs are needed. Unfortunately, such local cost information is usually not available at the global level in an MDBS.

A query sampling technique was suggested for the CORDS-MDBS. The idea of this technique is as follows: (1) queries at a local DBS are first classified into homogeneous classes; (2) a sample of queries are drawn from each query class; (3) the sample queries are performed at the local DBS and their costs are observed; (4) the observed costs are used to derive cost estimation formulas for the query classes based on multiple regression analysis. To estimate the cost of a query, the query class to which the query belongs is identified first, and the corresponding cost estimation formula is then used to give an estimated cost for the query.

To classify local queries, three types of information are utilized: characteristics of queries such as query syntax, characteristics of operand tables such as cardinality and index information, and characteristics of underlying local DBS such as supported access methods. To draw sample queries from each query class, a mixture of simple random sampling, stratified sampling, clustering sampling and judgment sampling is applied. A statistical procedure based on multiple regression analysis was developed to automatically select significant explanatory variables in a cost formula and estimate their coefficients in the formula.

Our experiments[10, 11] on three commercial DBMSs (ORACLE, DB2/6000 and EMPRESS) demonstrated that the query sampling method is quite promising in estimating local cost parameters in an MDBS.

Although a number of sampling techniques have been applied to query optimization in the literature[4, 6, 7], all of them perform data sampling (i.e., sampling data from databases) instead of query sampling (i.e., sampling queries from a query class). Query sampling is one of our new techniques.

2.2 Query Probing Technique

The basic idea of the query probing technique is to use some special queries, called probing queries, to discover missing information from a local DBS. There are several types of probing queries.

Catalog probing queries. A global query optimizer can make use of information stored in the multidatabase catalog to perform global query optimization. If some information is missing or out-of-date in the multidatabase catalog while the relevant local catalogs are accessible, the global query optimizer can perform probing queries on the local catalogs to obtain the desired information. Information obtained in this way can be either statistical information such as table cardinality or schema information such as available indexes.

Statistical probing queries. Some statistics can be obtained by performing probing queries directly on the relevant tables. For example, to obtain the cardinality $|R|$ of table R, the query $SELECT\ COUNT(*)\ FROM\ R$ can be used; or query $SELECT\ SUM(R.a)\ FROM\ R$ is performed first and $|R| = SUM(R.a)/AVG(R.a)$ is then calculated, if $AVG(R.a)$ is already known.

Performance measuring queries. Performance parameters about the underlying network and local DBSs can be measured by performing one or more probing queries. The performance parameters obtained by such probing queries can also be used to derive other missing desired information. For example, if the cardinality $|R|$ of table R is unknown, a performance measuring query like: $SELECT\ R.a,\ R.b,\ R.c\ FROM\ R\ WHERE\ R.a = 3$ can be performed. Two pieces of information that can be measured from the execution of this query are: (1) cost (elapse time) T and (2) result table size RS. Assume the cost estimation formula for this query is:

$$T = C_0 + C_1 * |R| + C_2 * RS$$

where C_0, C_1 and C_2 are known coefficients. In this case, the unknown variable is $|R|$ instead of T. An estimation formula $(T - C_0 - C_2 * RS)/C_1$ for $|R|$ can be obtained by solving this equation.

In fact, the sampling queries discussed in the last subsection can also be considered as the last type of probing queries.

2.3 Piggyback Method

Performing probing queries requires extra cost for global query optimization. To reduce such cost, a piggyback method is suggested. The idea of this method is to ride some side retrievals piggyback on the processing of a user query to get desirable information with just a slight additional cost.

Let us consider an example. Assume that tables $R_1(a_1, a_2, a_3)$ and $R_2(b_1, b_2, b_3, b_4)$ are at local sites A and B. For the following user query

$$\pi_{R_1.a_1,\ R_2.b_1}(R_1 \underset{R_1.a_2=R_2.b_2}{\bowtie} R_2),$$

a feasible execution plan is to execute the subquery Q: $\pi_{R_2.b_1,\ R_2.b_2}(R_2)$ at site B first, then transfer the result table S to site A, and perform the final join $\pi_{R_1.a_1,\ S.b_1}(R_1 \underset{R_1.a_2=S.b_2}{\bowtie} S)$ at site A. Clearly, statistical information such as maximums and averages about the first two columns of R_2 can be obtained by analyzing the intermediate result table S. However, no information about $R_2.b_3$ can be obtained during the query processing. To obtain information about $R_2.b_3$, an alternative subquery Q': $\pi_{R_2.b_1,\ R_2.b_2,\ R_2.b_3}(R_2)$ can be performed first instead of the original subquery Q. Although $R_2.b_3$ is not required in processing the given user query, desirable information about $R_2.b_3$ can be obtained by performing Q'. Notice that Q' only increases the processing cost slightly since both Q' and Q need to just scan table R once. Here $\pi_{R_2.b_3}(R_2)$ is a side retrieval piggybacked on the processing of the given query.

2.4 Other Optimization Techniques

The above query optimization techniques are unique in the CORDS-MDBS. In addition to these techniques, several advanced existing query optimization techniques are also suggested. Two of them are briefly described below.

Semantic query optimization. The idea of semantic query optimization is to make use of semantic knowledge, which is usually specified as integrity constraints, to transform a user query into a more efficient equivalent query. There are two types of such integrity constraints: global (intersite) ones and local ones. The global query optimizer can use global integrity constraints to identify semantically equivalent information at different sites and produce a query that requires less communication cost and is equivalent to the original user query. The local integrity constraints help the global query optimizer to produce efficient local queries obtained from decomposing a global query.

Adaptive query optimization. As mentioned, some optimization information may be missing or incomplete at the global level in an MDBS. Some execution plans generated by the global query optimizer at compile time may be inefficient or incomplete. Notice that some information may become available when the query is actually executed, such as intermediate result size and system load. To solve the above problem, adaptive query optimization is adopted. The idea is to improve or complete an execution plan adaptively based on up-to-date runtime information.

3 Global Query Optimizer

In this section, we discuss how to incorporate the query optimization techniques introduced in the last section into a global query optimizer designed for the CORDS-MDBS. We describe the query processing approach, procedure and components in the system. Note that not all components have been fully implemented.

3.1 Mixed Compilation and Interpretation Approach

There are two approaches for processing queries. One is the compilation approach. The other is the interpretation approach. The former performs comprehensive query optimization at compile time to generate an efficient execution plan. The same execution plan can be repeatedly invoked at run time. This approach is suitable for stored or embedded queries, which usually need to be executed many times. The interpretation approach, on the other hand, performs query optimization on the fly while a query is executed. No execution plan is stored. Since the time for query optimization affects the response time for a query, only simple query optimization is usually performed in this approach. The interpretation approach is suitable for ad hoc queries.

Since stored and embedded queries are assumed in our environment, it appears that the compilation approach should be adopted. However, inaccurate or incomplete information at the global level in an MDBS may cause the query optimizer to generate an inefficient execution plan for a query at compile time. To reduce the chance for performing expensive reoptimization at run time, some part of a query may be left uncompiled in the system if desired information is not available at compile time. In other words, an incomplete execution plan may be generated. The uncompiled part of a query is then optimized and interpreted at run time based newly available information. Hence, our approach is a mixed compilation and interpretation approach.

3.2 Related System Components

To process a global query, the global query optimizer needs to interact with several other components in the system (see Figure 2).

Request coordinator. A global query issued by a user is first passed to a module, called the request coordinator. The request coordinator coordinates the processing of user requests. For a query, it first invokes a parser to parse the query and then invokes the global query optimizer to generate an execution plan for the query. At run time, once the request coordinator receives a request from the user to execute the plan, it invokes an execution coordinator to coordinate the execution of the plan. After the query result is generated, the request coordinator return it to the user. For other user requests, such as updates and schema integration, the request coordinator invokes other relevant modules to process them.

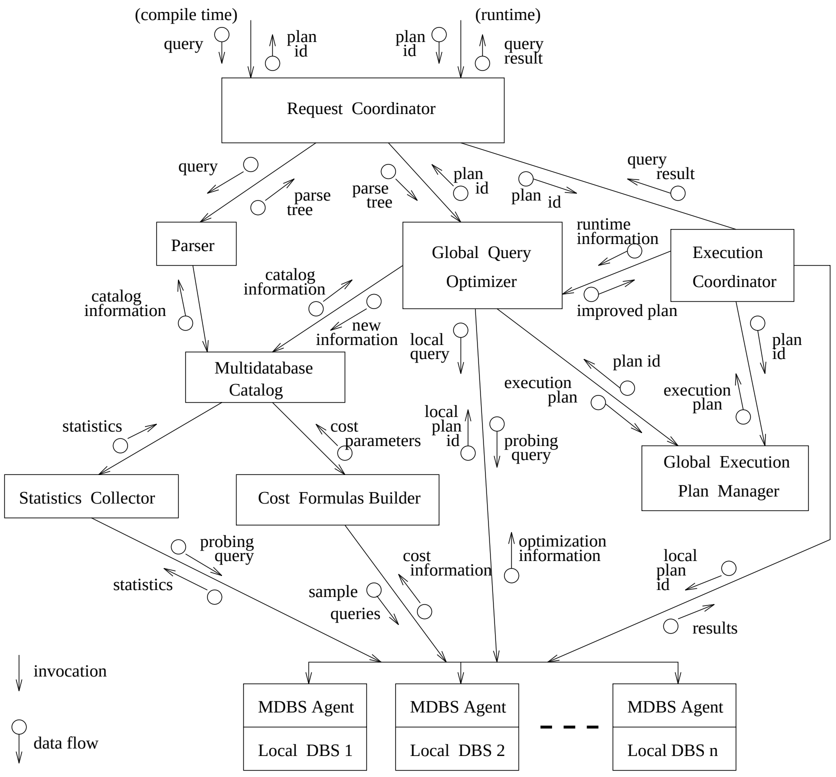

Figure 2: Components Related to Global Query Optimizer

Parser. This module is built by using YACC (Yet Another Compiler Compiler). It checks the syntax and semantics of a user request such as a query. The semantics checking is based on the schema information provided by the multidatabase catalog in the system.

Multidatabase catalog. This module maintains schema information, statistic information, and local cost parameters about the MDBS and its local DBSs. Such information is needed during query processing and optimization. Schema information is provided by a global DBA (database administer) via certain commands. Statistic information is collected by a module called statistics collector. Local cost parameters are estimated by a module called cost formulas builder. The multidatabase catalog is implemented on top of the EAN X.500 Directory Service.

Cost formula builder. This module derive local cost estimation formulas by using the query sampling method described in the last section. The derived cost formulas (coefficients) are saved in the multidatabase catalog, which will be utilized by the global query optimizer during query optimization. This module may need to be invoked to improve cost formulas every time when significant changes, such as index definitions and data volume, take place in the relevant local database.

Statistics collector. This module is periodically invoked to collect and update statistics stored in the multidatabase catalog by performing the probing queries on local DBSs, as described in the last section.

Global execution plan manager. All execution plans generated by the global query optimizer are managed by a module called global execution plan manager. This module not only stores execution plans but also keeps some relevant information such as identifiers, timestamps and validity. An execution plan can be retrieved via the module by an identifier when it is needed.

Execution coordinator. When this module receives an execution request for a query, it interacts with the global execution plan manager to get the corresponding execution plan. If this plan is invalid, it invokes the global query optimizer to re-optimize the query and generate a new execution plan. For a valid execution plan, this module coordinates and monitors execution of local execution plans contained in the global plan at local DBSs as well as the data transmissions among local DBSs. During the execution, this module may provide some useful runtime information for the global query optimization to perform adaptive query optimization. In the end, this module returns the query result to the request coordinator, which in turn returns the result to the user.

MDBS agent for local DBS. Any local request sent by the MDBS to a local DBS is through an MDBS agent. An MDBS agent provides a uniform relational interface, which is the Microsoft ODBC (Open Database Connectivity), between a local DBS and the MBDS global server. If the local DBS is a relational one, its MDBS agent is rather straightforward. However, if the local DBS is non-relational, efficient transformations are desired. Currently, three types of local DBSs are supported, i.e., relational, hierarchical and network.

There are a number of other modules in the CORDS-MDBS including schema integration, global transaction management, and communication client/server libraries. Their discussion can be found in [1].

3.3 Modules in the Global Query Optimizer

The global query optimizer is a major subsystem of the CORDS-MDBS. It consists of several smaller modules (see Figure 3). It performs query optimization to generate a good execution plan for a query at compile time. It also performs adaptive query optimization to improve an execution plan or complete an incomplete execution plan at run time.

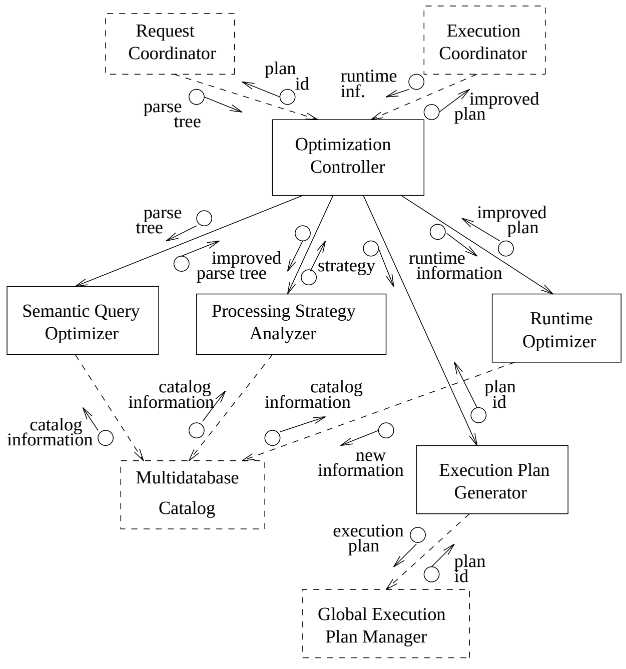

Figure 3: Multidatabase Services

Optimization controller. This is a control module of the global query optimizer. At compile time, it accepts the query parse tree from the request coordinator. It passes the tree to the semantic optimizer for improvement. It then invokes the processing strategy analyzer to choose an efficient execution strategy for the improved query. Finally the execution plan generator is invoked to generate an execution plan according to the chosen strategy. At run time, optimization controller passes runtime information to the runtime optimizer to improve/complete an execution plan adaptively.

Semantic query optimizer. This module transforms a user query expressed in the form of parse tree into another more efficient equivalent query based on the semantic information provided by the multidatabase catalog. Some heuristic rules, such as performing unary relational operations as early as possible, are also used to rewrite a user query.

Processing strategy analyzer. A query processing strategy describes the major decisions for processing a global query such as how to decompose the query, where to perform local queries and how to integrate local results. However, such a processing strategy for a query is usually not unique. Using the local cost parameters estimated by the query sampling method, a module called processing strategy analyzer analyzes alternative processing strategies for the given query and choose a good one among them. This module may also perform some probing queries on local DBSs to obtain/update some information required for the analysis.

Execution plan generator. Based on the chosen processing strategy, the execution plan generator generates the detailed execution plan for the query and passes it to the global execution plan manager to manage for later use.

Runtime optimizer. Some execution plans generated by the global query optimizer at compile time may be based on inaccurate information. Such execution plans may not be good. Furthermore, if some desired information is missing, an incomplete execution plan may be generated. A module, called runtime optimizer, in the CORDS-MDBS performs query optimization to adaptively improve a poor plan or complete an incomplete plan based on runtime information at run time.

The basic part of the global query optimizer described here is currently operational. The global query optimizer can successfully generate an efficient execution plan for a global query. Based on the plan, the system can answer users' global queries to retrieve data from different local DBSs (relational, hierarchical and network) including ORACLE, DB2/6000, EMPRESS, IMS, and VAX DBMS.

4 Conclusion

A multidatabase system is a distributed system that provides a global interface to heterogeneous preexisting local DBSs. To process a global query in such a system efficiently, global query optimization is required. However, local autonomy, which is the key feature of an MDBS, poses new challenges for performing global query optimization in an MDBS. A major challenge is that some local information that is needed for global query optimization may not be available at the global level.

In this paper, a global query optimizer in the multidatabase system CORDS-MDBS is discussed. To tackle the above major challenge, new query optimization and processing techniques such as query sampling, query probing, piggyback method, and mixed compilation and interpretation approach are adopted in the query optimizer. In addition, some advanced existing query optimization techniques such as semantic query optimization and adaptive query optimization are utilized as well. The architecture of the global query optimizer and its related components in the CORDS-MDBS are described. The global query optimizer currently supports access to several relational, hierarchical and network local DBSs including ORACLE, DB2/6000, EMPRESS, IMS and VAX DBMS. This optimizer shows the feasibility of developing an efficient MDBS with strong local autonomy. The architecture can be used as a reference model for developing a similar system.

Our work is only the beginning of more research that needs to be done in order to completely solve the challenges for global query optimization in an MDBS. In the future, besides ameliorating the global query optimizer, we also plan to incorporate our idea about fuzzy query optimization[12, 13] into the system.

References

[1] G.K. Attaluri, D.P. Bradshaw, N. Coburn, P.-Å. Larson, P. Martin, A. Silberschatz, J. Slonim, and Qiang Zhu. The CORDS multidatabase project. *IBM Systems Journal*, 34(1):39–62, 1995.

[2] M. W. Bright, A. R. Hurson, and S. H. Pakzad. A taxonomy and current issues in multidatabase systems. *IEEE Computer*, pages 50–59, Mar. 1992.

[3] W. Du, R. Krishnamurthy, and M. C. Shan. Query optimization in heterogeneous DBMS. In *Proceedings of VLDB*, pages 277–91, 1992.

[4] R. J. Lipton and J. F. Naughton. Practical selectivity estimation through adaptive sampling. In *Proceedings of SIGMOD*, pages 1–11, 1990.

[5] H. Lu and M.-C. Shan. On global query optimization in multidatabase systems. In *2nd Int'l workshop on Research Issues on Data Eng.*, page 217, Tempe, Arizona, USA, 1992.

[6] F. Olken and D. Rotem. Simple random sampling from relational databases. In *Proceedings of 12th VLDB*, pages 160–9, 1986.

[7] G. P. Shapiro and C. Connel. Accurate estimation of the number of tuples satisfying a condition. In *Proceedings of SIGMOD*, pages 256–76, 1984.

[8] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, Sept. 1990.

[9] Qiang Zhu. Query optimization in multidatabase systems. In *Proceedings of the 1992 IBM CAS Conference, vol.II*, pages 111–27, Toronto, Canada, Nov. 1992.

[10] Qiang Zhu and P.-Å. Larson. Solving local cost estimation problem for global query optimization in multidatabase systems. Technical Report CIS-TR-001-96, University of Michigan - Dearborn, 1996.

[11] Qiang Zhu and P.-Å. Larson. A query sampling method for estimating local cost parameters in a multidatabase system. In *Proceedings of the 10th IEEE International Conference on Data Engineering*, pages 144–53, Houston, Texas, Feb. 1994.

[12] Qiang Zhu and P.-Å. Larson. Establishing a fuzzy cost model for query optimization in a multidatabase system. In *Proceedings of the 27th IEEE/ACM Hawaii International Conference on System Sciences*, pages 263–72, Maui, Hawaii, Jan. 1994.

[13] Qiang Zhu and P.-Å. Larson. Query optimization using fuzzy set theory for a multidatabase system. In *Proceedings of the 1993 IBM CAS Conference*, pages 848–59, Toronto, Canada, Oct. 1993.