A Query Sampling Method for Estimating Local Cost Parameters in a Multidatabase System *

Qiang Zhu Per-Åke Larson

Department of Computer Science
University of Waterloo, Canada, N2L 3G1

Abstract

In a multidatabase system (MDBS), some query optimization information related to local database systems may not be available at the global level because of local autonomy. To perform global query optimization, a method is required to derive the necessary local information. This paper presents a new method that employs a query sampling technique to estimate the cost parameters of an autonomous local database system. We introduce a classification for grouping local queries and suggest a cost estimation formula for the queries in each class. We present a procedure to draw a sample of queries from each class and use the observed costs of sample queries to determine the cost parameters by multiple regression. Experimental results indicate that the method is quite promising for estimating the cost of local queries in an MDBS.

1 Introduction

A multidatabase system (MDBS) integrates data from pre-existing autonomous local databases managed by heterogeneous database management systems (DBMS) in a distributed environment. It acts as a front end to multiple local DBMSs providing full database functionality to global users and interacts with the local DBMSs at their external user interfaces. A key feature of an MDBS is the local autonomy that individual databases retain to serve existing applications. Most differences between a conventional distributed database system (DDBS) and an MDBS are caused by local autonomy. These differences introduce new challenges for query optimization in an MDBS[5, 10, 12].

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

Among the challenges, a crucial one is that some local query optimization information, e.g., local cost functions, may not be available to the global query optimizer. It is, therefore, difficult for the global query optimizer to determine a good execution plan for a given (global) query. This problem does not exist in a conventional DDBS because all sites run the same distributed database management system. Its query optimizer can make use of both global and local information to produce a good execution plan for a given query. In an MDBS, methods to derive or estimate local query optimization information are required.

In [2], Du *et al.* proposed a calibration method to deduce necessary local information. The idea is to construct a local synthetic calibrating database with special properties and to run a set of queries against this database. Cost metrics for the queries are recorded and used to deduce the coefficients in a cost formula for the underlying local database system by using the properties of the calibrating database. This method demonstrates the possibility of obtaining necessary information from an autonomous local database system. However, this method has several shortcomings:

- The deduced cost formulas cannot be applied if the local access method for a query is not known, which is frequently the case in an MDBS.
- It may not be possible (or allowed) to create a calibrating database at the local site in an MDBS.
- Cost parameters deduced by using a synthetic calibrating database may not be valid for real databases because of different data distributions, database sizes, file structures, adjustable local system parameters and so on.
- The method does not take into consideration a dynamically changing environment.

In order to overcome these shortcomings, we propose a new method based on statistical sampling to establish cost (estimation) formulas for local database systems in this paper. The idea is to group all possible queries on a local database into classes according to available information so that the costs of the queries in each class can be estimated by the same formula. A sample of queries are drawn from each class and run against the underlying real database. The costs of the sample queries are recorded and used to derive a cost formula for the queries in the class by multiple regression. The coefficients of the cost formulas for local database systems are kept in a global catalog and retrieved during query optimization. The coefficients can also be dynamically adjusted according to real costs observed from execution of user queries. Since the cost formulas are derived by using real local databases and are dynamically adjustable, this method solves the problems mentioned above.

A number of sampling techniques have been used in query optimization in the literature, but all of them perform *data sampling* (i.e., sample data from underlying databases) instead of *query sampling* (i.e., sample queries from a query class). Muralikrishna and Piatetsky-Shapiro *et al.*[6, 9] discussed using data sampling to build approximate selectivity histograms. Hou and Lipton *et al.*[3, 4] investigated several different data sampling techniques, e.g., simple sampling, adaptive sampling and double sampling, to estimate the size of a query result. Olken *et al.*[7] considered the problem of constructing a random subset of a query result without computing the full result. All their work is about performing a given query against a sample of data and deriving properties for the underlying data (e.g., selectivities). This paper considers performing a sample of queries against (whole) underlying data and deriving a property about a query on its supporting DBMS (i.e., performance of the query on the DBMS).

The rest of this paper is organized as follows. Section 2 discusses how to classify queries on a local database system according to available information. Section 3 describes a regression cost formula for each query class. Section 4 investigates how to draw sample queries from each query class so that their costs can be used to derive the cost parameters. Section 5 gives some experimental results. Section 6 discusses a number of other related issues about using the presented method in an MDBS.

2 Classification of Queries

Different local DBMSs may adopt different local data models. At the global level of an MDBS, there usually is a common global data model. In our MDBS project, the global data model is assumed to be relational. Each local DBMS is associated with an MDBS agent which provides a relational interface if the local DBMS is non-relational. Hence, the global query optimizer in the MDBS may view all participating local DBMSs as relational ones. For simplicity, we also assume that all data are schematically and representationally compatible.

Given a query, the global query optimizer generates an execution plan that specifies how the query is decomposed into local queries and how the local results are integrated to produce the final result for the user. The way to decompose a query into local queries is not unique. If the global query optimizer knows the costs of local queries, it can choose a good (low cost) execution plan among a number of alternatives. However, unlike a traditional DDBS, the local cost functions are usually not known by the global query optimizer in an MDBS. In order to estimate the costs of local queries, we need to find a way to derive cost formulas for local database systems.

Many possible queries can be issued against a local database. It is not possible to estimate the costs of all queries by a single formula. Usually, the costs of the queries executed by using the same access method, e.g., sequential scan, can be estimated by the same formula. However, which access method is used for a local query may not be known at the global level in an MDBS. It depends on the local DBMS.

Fortunately, there are some common principles for choosing an access method for a query in most DBMSs. Based on available information and these common principles, we can group queries into more homogeneous classes. The costs of all queries in a class are estimated by the same formula. If sufficient information is available, we can classify queries such that each class corresponds to one access method. The estimated costs are expected to be more accurate in this case. If the available information is not sufficient, it is possible that queries executed by different access methods are put in the same class. Since the practical goal of query optimization in an MDBS is the same as those of many traditional query optimizers, that is, avoiding bad execution plans instead of achieving a truly optimal one, estimation errors can be tolerated to a certain degree.

Unlike a query optimizer in a traditional DDBS, the global query optimizer in an MDBS has limited information available. To classify queries, the following types of information can be made use of:

- *characteristics of queries*: such as unary queries, 2-way join queries and so on. This information can be obtained by analyzing a given query.

- *characteristics of operand tables*: such as the number of columns, the number of tuples, indexed columns and so on. This information can usually be obtained from the local catalog.

- *characteristics of underlying local DBMSs*: such as types of access methods supported. This information can be obtained from the documentation of a local DBMS.

Since most common queries can be expressed by a sequence of select (σ), project (π) and join ($\bowtie$), we consider only these three types of operations in this paper. The cost of a query composed from these operations can be estimated by composing the costs of the operations. In real systems a project is usually computed together with the select or join that it follows, so we will not consider it separately. In this paper, a select that may or may not be followed by a project is called a unary query. A join that may or may not be followed by a project is called a join query.

Let G be the set of all (unary and join) queries on a local $DB\ i$ managed by a $DBMS\ j$. Let R and S be two tables in $DB\ i$, α be a list of columns in R and/or S, F be a qualification of a query on R and/or S, and C be a constant in the domain of a relevant column. Without loss of generality, qualifications of queries are assumed to be in the conjunctive normal form. The basic predicates allowed are of the forms $R.a\ \theta\ C$ and $R.a\ \theta\ S.b$, where $\theta \in \{=, \neq, >, <, \geq, \leq\}$.

Since a unary query and a join query have different numbers of operands and are evaluated by totally different access methods, their costs cannot be estimated by the same formula. Hence they must be in different classes:

$$G = G_1 \cup G_2$$

where $G_1 = \{\textit{unary queries}\}$, $G_2 = \{\textit{join queries}\}$.

In G_1, the queries evaluated by a sequential scan method, an index-based scan method or a clustered-index-based scan method may have different performance, i.e., their cost formulas may not be the same. As mentioned above, although it is sometimes difficult to tell, from available information in an MDBS, which access method is to be used by a local DBMS for a given query, there is a common policy in most DBMSs, that is, if the qualification of a unary query has a conjunct $R.a = C$ in its qualification, where $R.a$ is an indexed (clustered-indexed) column, an index-based (clustered-index-based) scan method is employed to evaluate the query. The first class G_1 can, then, be further divided into three smaller classes:

$$G_1 = G_{11} \cup G_{12} \cup G_{13} \tag{1}$$

where

$$\begin{aligned} G_{11} = \{\ & \pi_\alpha(\sigma_F(R)) \mid \textit{F has at least one} \\ & \textit{conjunct } R.a = C, \textit{ where R.a is a} \\ & \textit{clustered} - \textit{indexed column}\ \}, \end{aligned} \tag{2}$$

$$\begin{aligned} G_{12} = \{\ & \pi_\alpha(\sigma_F(R)) \mid \pi_\alpha(\sigma_F(R)) \textit{ not in } G_{11} \\ & \textit{and F has at least one conjunct } R.a = C, \\ & \textit{where R.a is an indexed column}\ \}, \end{aligned} \tag{3}$$

$$G_{13} = G_1 - (\ G_{11} \cup G_{12}\ ). \tag{4}$$

Similarly, the class G_2 can be further divided into three smaller classes:

$$G_2 = G_{21} \cup G_{22} \cup G_{23} \tag{5}$$

where

$$\begin{aligned} G_{21} = \{\ & \pi_\alpha(R \bowtie_F S) \mid \textit{F has at least one} \\ & \textit{conjunct } R.a = S.b, \textit{ where R.a or R.b (or} \\ & \textit{both) is a clustered} - \textit{indexed column}\ \}, \end{aligned} \tag{6}$$

$$\begin{aligned} G_{22} = \{\ & \pi_\alpha(R \bowtie_F S) \mid \pi_\alpha(R \bowtie_F S) \textit{ not in} \\ & G_{21} \textit{ and F has at least one conjunct} \\ & R.a = S.b, \textit{ where R.a or S.b (or both) is} \\ & \textit{an indexed column}\ \}, \end{aligned} \tag{7}$$

$$G_{23} = G_2 - (\ G_{21} \cup G_{22}\ ). \tag{8}$$

G_{21}, G_{22} and G_{23} correspond to the clustered-index-based join method, index-based join method, and other join method(s) (e.g., merge join and sequential nested loop join), respectively.

In principle, any of the classes $G_{11} \sim G_{23}$ can be further divided into smaller classes if more information is available. For example, in some DBMSs (such as RDB/VMS) a sorted-indexed column $R.a$ may be supported to make the execution of a range query more efficient than using a sequential scan method. Thus, the class G_{13} can be further divided into two smaller classes — one for range queries, the other for other queries. As another example, any of the above classes can be divided into smaller classes according to the (estimated) sizes of results (or operands) of queries because a DBMS may adopt different processing and buffering strategies for different sizes of query results. The classification of queries may vary from one DBMS or application to another. In general, a refined classification is expected to yield better estimates because each query class is usually more homogeneous in performance. However, the overhead of maintaining the cost parameters grows as the number of query classes increases. A trade-off is required between estimation accuracy and overhead. For simplicity, we only discuss the classification at the level of (1) $\sim$ (8) in this paper. The ideas of the following discussion can be applied to a further refined classification.

The classification (1) $\sim$ (8) is suitable for many DBMSs. It is based on not only the characteristics of queries but also the characteristics of the local database and DBMS. On one hand, more classes may result from a refined classification if more information is available. On the other hand, some of the classes $G_{11} \sim G_{23}$ may be empty on a local database because some access methods may not be supported in the underlying DBMS. For example, G_{11} and G_{21} are empty for a local database managed by Empress because a clustered-index is not supported.

3 Cost Estimation Formulas

For each query class resulting from a classification, a formula is needed to estimate the cost of queries in the class. Since the implementation details of access methods in a DBMS, e.g., I/O numbers and buffer sizes, are not known, an exact analytical cost formula can not be obtained. Furthermore, we sometimes do not even know what access method(s) is (are) used for the queries in a class. Fortunately, regression methods in statistics can be used to estimate the value of one quantitative variable (dependent variable) by considering its relationship with one or more other quantitative variables (independent variables).

In our case, the dependent variable is the costs of the queries in a class. Clearly, the cost of a query is proportional to the numbers of tuples in the operand table(s) and the result table. Although tuple lengths of the operand table(s) and the result table may also affect the cost of a query, the effect is usually very small. Hence, tuple lengths are neglected in our cost formulas.

Let us consider the class $G_{1k}(k = 1,2,3)$. Let N_{1k} be the number of tuples in the operand table of a query in G_{1k}, S_{1k} be the selectivity of the query. Then $S_{1k} * N_{1k}$ is the number of tuples in the result table of the query. In general, a cost estimation formula for G_{1k} is $\widehat{Y}_{1k} = F_{1k}(N_{1k},\ S_{1k} * N_{1k})$, where the function F_{1k} yields an estimated cost $\widehat{Y}_{1k}$ for the query. F_{1k} varies from one database system to another. However, most database systems follow a similar pattern. Observing existing cost models, we find that the formula used to estimate the cost of a query in G_{1k} in most systems is of the following form:

$$\widehat{Y}_{1k} = \beta 0_{1k} + \beta 1_{1k} * N_{1k} + \beta 2_{1k} * S_{1k} * N_{1k}, \qquad (k = 1,2,3) \quad (9)$$

where the parameters $\beta 0_{1k}$, $\beta 1_{1k}$ and $\beta 2_{1k}$ reflect the initialization cost, the cost of retrieving a tuple from the operand table and the cost of processing a result tuple, respectively. In a traditional cost model, a parameter may be split up into several parts (e.g., $\beta 1_{13}$ may consist of I/O cost and CPU cost) and can be given by analyzing the implementation details of the employed access method. The formula can also be applied to an MDBS. However, the parameters cannot be analytically derived. To estimate the parameters, we view (9) as a regression equation and estimate the parameters (regression coefficients) by using the costs measured for sample queries drawn from the class. Since more than one independent variables (N_{1k}, S_{1k}) are involved in (9), it is a multiple regression problem.

Let $\{Q_{1k}^{(m)} \mid m = 1, \cdots, M_{1k}\}$ be a set of sample queries drawn from G_{1k}, $N_{1k}^{(m)}$ be the number of tuples in the operand table of $Q_{1k}^{(m)}$, $S_{1k}^{(m)}$ be the selectivity of $Q_{1k}^{(m)}$, $Y_{1k}^{(m)}$ be the cost measured by executing $Q_{1k}^{(m)}$. Applying the method of least squares[8], we can derive a system of normal equations (omitted) for (9) that takes $Y_{1k}^{(m)}$, $N_{1k}^{(m)}$, $S_{1k}^{(m)}$ as inputs and produces the estimates of the coefficients $\beta 0_{1k}$, $\beta 1_{1k}$ and $\beta 2_{1k}$.

For $G_{2k}(k = 1,2,3)$, let $N1_{2k}$ and $N2_{2k}$ be the numbers of tuples in the operand tables of a query in the class G_{2k}, S_{2k} be the selectivity of the query. Then $S_{2k} * N1_{2k} * N2_{2k}$ is the number of tuples in the result table of the query. A general cost estimation formula is of the form: $\widehat{Y}_{2k} = F_{2k}(N1_{2k},\ N2_{2k},\ S_{2k} * N1_{2k} * N2_{2k})$, where $\widehat{Y}_{2k}$ is the estimated cost, F_{2k} depends on the database system. For many database systems, the following formula is applicable:

$$\widehat{Y}_{2k} = \beta 0_{2k} + \beta 1_{2k} * N1_{2k} + \beta 2_{2k} * N2_{2k} + \beta 3_{2k} * S_{2k} * N1_{2k} * N2_{2k}, \quad (k = 1,2,3) \; (10)$$

where the parameters $\beta 0_{2k}$, $\beta 1_{2k}$, $\beta 2_{2k}$ and $\beta 3_{2k}$ reflect the initialization cost, the cost of retrieving a tuple from the first operand table, the cost of retrieving a tuple from the second operand table, and the cost of processing a result tuple, respectively. Similar to (9), we view (10) as a regression equation, and estimate the parameters by using the regression method.

Let $\{Q_{2k}^{(m)} \mid m = 1, \cdots, M_{2k}\}$ be a set of sample queries drawn from G_{2k}, $N1_{2k}^{(m)}$ and $N2_{2k}^{(m)}$ be the numbers of tuples in the first and second operand tables of $Q_{2k}^{(m)}$ respectively, $S_{2k}^{(m)}$ be the selectivity of $Q_{2k}^{(m)}$, $Y_{2k}^{(m)}$ be the cost measured by executing $Q_{2k}^{(m)}$. Applying the method of least squares again, we can derive a system of normal equations (omitted) for (10) that takes $Y_{2k}^{(m)}$, $N1_{2k}^{(m)}$, $N2_{2k}^{(m)}$, $S_{2k}^{(m)}$ as inputs and produces the estimates of the parameters $\beta 0_{2k}$, $\beta 1_{2k}$, $\beta 2_{2k}$ and $\beta 3_{2k}$.

Let $\widehat{Y}_{tk}^{(m)}$ be the estimated cost of $Q_{tk}^{(m)}(t = 1, 2;\ k = 1, 2, 3;\ m = 1, \cdots, M_{tk})$ by using (9) or (10) with the estimated parameters. The standard error of estimation is given by[8]:

$$s_{tk} = \sqrt{\frac{\sum_{m=1}^{M_{tk}} (Y_{tk}^{(m)} - \widehat{Y}_{tk}^{(m)})^2}{M_{tk} - (2+t)}}.$$

It is an indication of the accuracy of estimation. The smaller s_{tk} is, the better the estimation is. The coefficient of multiple determination is defined as[8]:

$$r_{tk}^2 = 1 - \sum_{m=1}^{M_{tk}} (Y_{tk}^{(m)} - \widehat{Y}_{tk}^{(m)})^2 / \sum_{m=1}^{M_{tk}} (Y_{tk}^{(m)} - \overline{Y}_{tk}^{(m)})^2,$$

where $\overline{Y}_{tk} = [\sum_{m=1}^{M_{tk}} Y_{tk}^{(m)}]/M_{tk}$. r_{tk}^2 (≤ 1) is the proportion of variability in the query costs explained by the independent variables in (9) or (10). The larger r_{tk}^2 is, the better the estimation is.

Since (9) and (10) are based on analytic formulas in traditional cost models, they are expected to be good as cost estimation formulas for most database systems. If some of the formulas are not good for a database system, they could be improved via a transformation. For example, $N_{1k} * S_{1k}$ could be replaced by $(N_{1k} * S_{1k})^{\mu}$ in (9), where the parameter μ can be adjusted to a value such that r_{1k}^2 is high and s_{1k} is low. Furthermore, a transformation can be based on observations on the costs recorded for sample queries.

In addition to r_{tk}^2 and s_{tk}, the significance of a chosen formula can be tested by statistical hypothesis testing, like the F-test and t-test[8].

4 Sample Queries

Let us consider how to draw a sample of queries from each query class G_{tk} (population). There usually is a large number of queries in each class. It is too expensive or even impossible to perform all the queries to obtain cost information. We expect that a small number of sample queries can represent the whole population so that the estimation formulas derived from the costs of the sample queries can be used to give a good estimate for the cost of any query in the population. By having too small a sample, however, poor estimates of the regression parameters may result — leading to poor estimates of query costs. Thus, there is a minimum sample size requirement. A commonly used rule[8] is to sample at least $10 * (n+1)$ observations for a regression formula with n parameters. Therefore, we need at least 40 and 50 sample queries for the regression formulas (9) and (10) respectively. Let $U_{tk} (U_{1k} \geq 40;\ U_{2k} \geq 50)$ denote the minimum sample size we choose for the query class G_{tk}.

There are various ways for a sample to be drawn from a population: probability sampling (simple random sampling, stratified sampling, cluster sampling, and systematic sampling), judgment sampling, and convenience sampling. We will use a method of mixed judgment sampling and simple random sampling to draw a sample of queries from a given query class, that is, use our knowledge about queries to restrict the class to a representative subset and then apply simple random sampling to draw a sample from the subset if the subset is still too large.

Assume there are $K (\geq 1)$ tables, $R_1, \cdots, R_K$, in $DB\ i$. Let CL_i, CC_i, CI_i and CN_i $(1 \leq i \leq K)$ be the sets of all columns, the clustered-indexed columns, indexed columns and non-indexed columns in the table R_i, respectively. For a composite indexed column, CI_i contains only its (simple) component columns. Similarly, CC_i contains only simple columns. Thus $CL_i = CI_i \cup CC_i \cup CN_i$. Let $C^{(i.a)}$ and $\tilde{C}^{(i.a)}$ be a given value and a randomly-chosen value in the domain of the column $R_i.a$, respectively. Let $\alpha^{(i)}$ be a given non-empty subset of CL_i, and $\tilde{\alpha}^{(i)}$ and $\tilde{\alpha}^{(ij)}$ be non-empty randomly-chosen subsets of CL_i and $CL_i \cup CL_j$, respectively. $\wedge$ denotes the logical connective AND, and $|X|$ denotes the cardinality of the set X. θ is a comparison randomly chosen from $\{<, >, \neq, =\}$.

Consider the class G_{11}. For each column $R_i.a \in CC_i$, there is an associated set $S_{R_i.a}$ of queries in G_{11}, i.e., the set of queries that have a conjunct $R_i.a = C^{(i.a)}$ in their qualifications. Clearly, $G_{11} = \cup_{i=1}^{K} [\ \cup_{R_i.a \in CC_i} S_{R_i.a}\ ]$. Let $d_{11} = \sum_{i=1}^{K} |CC_i|$.

(i) If $d_{11} = U_{11}$, i.e., the total number of clustered-indexed columns equals to the required minimum sample size, we choose a query from each $S_{R_i.a}$ as a sample query, which is used to estimate the performance of using the clustered-index on $R_i.a$. The whole sample takes into consideration all the clustered-indexes. The query chosen to represent the queries in $S_{R_i.a}$ is $\pi_{\tilde{\alpha}^{(i)}}(\sigma_{R_i.a=\tilde{C}^{(i.a)}}(R_i))$. The choice is based on the fact that the key part of the qualification of a query in $S_{R_i.a}$ that affects the performance of the query is most likely the conjunct $R_i.a = C^{(i.a)}$.

(ii) If $d_{11} > U_{11}$, we only select to consider a certain percent $\eta_{11}\%$ of the clustered-indexed columns for each table, where η_{11} is determined in Proposition 1 below. In other words, we consider $\lceil |CC_i| * \eta_{11}\% \rceil$ ($\lceil\cdot\rceil$ denotes the ceiling function) number of clustered-indexed columns for each R_i. Let $CC_i^{(\eta_{11}\%)}$ be a selected random subset of CC_i with the size $\lceil |CC_i| * \eta_{11}\% \rceil$. For each $R_i.a \in CC_i^{(\eta_{11}\%)}$, we draw a sample query $\pi_{\tilde{\alpha}^{(i)}}(\sigma_{R_i.a=\tilde{C}^{(i.a)}}(R_i))$. In fact, $\eta_{11}\% = 100\%$ for the case (i).

(iii) If $d_{11} < U_{11}$, we consider all the clustered-indexed columns for each table (i.e., $\eta_{11}\% = 100\%$) and choose ω_{11} (> 1) number of queries from $S_{R_i.a}$ for each $R_i.a \in CC_i$, where ω_{11} is determined in Proposition 1 below. We take $\pi_{\tilde{\alpha}_j^{(i)}}(\sigma_{R_i.a=\tilde{C}_j^{(i.a)}}(R_i))$ $(1 \leq j \leq \omega_{11})$ as sample queries from $S_{R_i.a}$. In fact, $\omega_{11} = 1$ for the cases (i) and (ii).

Therefore, the sample of queries drawn from G_{11} is:

$$SP_{11} = \cup_{i=1}^{K} \; [\; \cup_{R_i.a \,\in\, CC_i^{(\eta_{11}\%)}} \; [\; \cup_{j=1}^{\omega_{11}} \; \{\; \pi_{\tilde{\alpha}_j^{(i)}}(\sigma_{R_i.a=\tilde{C}_j^{(i.a)}}(R_i)) \;\} \;]\;] \;,$$

where η_{11} and ω_{11} are determined by the following proposition (proof omitted) to guarantee that the sample size is greater than or equal to the required minimum size U_{11}:

Proposition 1 *For $d_{11} \geq U_{11}$, if $\eta_{11} = 100 * U_{11}/d_{11}$ and $\omega_{11} = 1$, then $|SP_{11}| \geq U_{11}$. For $d_{11} < U_{11}$, if $\eta_{11} = 100$ and $\omega_{11} = \lceil U_{11}/d_{11} \rceil$, then $|SP_{11}| \geq U_{11}$.*

Similarly, for the class G_{12}, let $d_{12} = \sum_{i=1}^{K} |CI_i|$, we draw the following sample of queries:

$$SP_{12} = \cup_{i=1}^{K} \; [\; \cup_{R_i.a \,\in\, CI_i^{(\eta_{12}\%)}} \; [\; \cup_{j=1}^{\omega_{12}} \; \{\; \pi_{\tilde{\alpha}_j^{(i)}}(\sigma_{R_i.a=\tilde{C}_j^{(i.a)}}(R_i)) \;\} \;]\;] \;,$$

where $CI_i^{(\eta_{12}\%)}$ is a random subset of CI_i with the size $\lceil |CI_i| * \eta_{12}\% \rceil$, η_{12} and ω_{12} are determined by a proposition similar to Proposition 1 to guarantee a sufficient sample size.

For the class G_{13}, like G_{11} and G_{12}, we choose a query with a single predicate as a sample query. However, we need to consider not only a query with an equality predicate but also a query with a non-equality predicate. Since the costs of processing $\pi_{\alpha^{(i)}}(\sigma_{R_i.a \leq C^{(i)}}(R_i))$, $\pi_{\alpha^{(i)}}(\sigma_{R_i.a \geq C^{(i)}}(R_i))$ and $\pi_{\alpha^{(i)}}(R_i)$ are usually close to the costs of processing $\pi_{\alpha^{(i)}}(\sigma_{R_i.a < C^{(i)}}(R_i))$, $\pi_{\alpha^{(i)}}(\sigma_{R_i.a > C^{(i)}}(R_i))$ and $\pi_{\alpha^{(i)}}(\sigma_{R_i.a \neq C^{(i)}}(R_i))$ respectively, we use the later three types of queries to represent the former three types of queries. In other words, we consider only $<$, $>$, $\neq$ and $=$ for the predicates in the sample queries. For each R_i, we randomly select a number of columns to generate the sample queries.

If a selected column $R_i.a \in CN_i$, we generate four sample queries (with non-empty random subsets of CL_i as the target project lists) with qualifications $R_i.a < \tilde{C}_1^{(i.a)}$, $R_i.a > \tilde{C}_2^{(i.a)}$, $R_i.a \neq \tilde{C}_3^{(i.a)}$ and $R_i.a = \tilde{C}_4^{(i.a)}$, respectively.

If $R_i.a \in CC_i \cup CI_i$, we generate three sample queries with qualifications $R_i.a < \tilde{C}_1^{(i.a)}$, $R_i.a > \tilde{C}_2^{(i.a)}$ and $R_i.a \neq \tilde{C}_3^{(i.a)}$, respectively.

If m_i columns are selected from the table R_i, at least $\sum_{i=1}^{K} 3 * m_i$ sample queries will be generated by the above procedure. Let $d_{13} = \sum_{i=1}^{K} 3 * |CL_i|$. If $d_{13} \geq U_{13}$, we consider only a certain percent $\eta_{13}\%$ of all columns of each table, where η_{13} is determined in Proposition 2 below. If $d_{13} < U_{13}$, we select all columns for each table and use multiple random values for each column to generate sufficient sample queries. Let $CL_i^{(\eta_{13}\%)} = CC_i^{\eta_{13}} \cup CI_i^{\eta_{13}} \cup CN_i^{\eta_{13}}$ be a random subset of CL_i with the size of $\lceil |CL_i| * \eta_{13}\% \rceil$, where $CC_i^{\eta_{13}} \subseteq CC_i$, $CI_i^{\eta_{13}} \subseteq CI_i$, and $CN_i^{\eta_{13}} \subseteq CN_i$ (some subsets may be empty). Then the sample of queries drawn from G_{13} is

$$\begin{aligned} SP_{13} = \cup_{i=1}^{K} \; [\; \cup_{j=1}^{\omega_{13}} \; [\; [\; \cup_{R_i.a \in (CC_i^{\eta_{13}} \cup CI_i^{\eta_{13}})} \\ \{\; \pi_{\tilde{\alpha}_{1j}^{(i)}}(\sigma_{R_i.a<\tilde{C}_{1j}^{(i.a)}}(R_i)),\; \pi_{\tilde{\alpha}_{2j}^{(i)}}(\sigma_{R_i.a>\tilde{C}_{2j}^{(i.a)}}(R_i)), \\ \pi_{\tilde{\alpha}_{3j}^{(i)}}(\sigma_{R_i.a\neq\tilde{C}_{3j}^{(i.a)}}(R_i)) \;\}] \;\cup\; [\cup_{R_i.b \in CN_i^{\eta_{13}}} \\ \{\; \pi_{\tilde{\alpha}_{4j}^{(i)}}(\sigma_{R_i.b<\tilde{C}_{1j}^{(i.b)}}(R_i)),\; \pi_{\tilde{\alpha}_{5j}^{(i)}}(\sigma_{R_i.b>\tilde{C}_{2j}^{(i.b)}}(R_i)), \\ \pi_{\tilde{\alpha}_{6j}^{(i)}}(\sigma_{R_i.b\neq\tilde{C}_{3j}^{(i.b)}}(R_i)), \pi_{\tilde{\alpha}_{7j}^{(i)}}(\sigma_{R_i.b=\tilde{C}_{4j}^{(i.b)}}(R_i)) \;\}]]]. \end{aligned}$$

where η_{13} and ω_{13} are determined in the following proposition (proof omitted) to guarantee a sufficient sample size:

Proposition 2 *For $d_{13} \geq U_{13}$, if $\eta_{13} = 100 * U_{13}/d_{13}$ and $\omega_{13} = 1$, then $|SP_{13}| \geq U_{13}$. For $d_{13} < U_{13}$, if $\eta_{13} = 100$ and $\omega_{13} = \lceil U_{13}/D_{13} \rceil$, where $D_{13} = \sum_{i=1}^{K}[3*(|CC_i|+|CI_i|)+4*|CN_i|]$, then $|SP_{13}| \geq U_{13}$.*

Drawing sample queries from a join query class is more complicated than drawing sample queries from a unary query class. One rule used in the above sampling procedure for a unary query class is that there is at least one sample query on each operand (table). However, the rule may not be good for the sampling procedure for a join query class because the number of possible operands (table pairs) is usually large. Even if we draw only one sample query for each pair of joining tables, the sample size may be much larger than the required minimum sample size, which may not be good because performing a sample join query is usually expensive. Another problem is that not every pair of columns can be referred in a join predicate. A joining column pair must be comparable by comparison operators.

Let us first consider the class G_{21}. Let

$$\Delta_1 = \{\ \{R_i.a,\ R_j.b\} \mid R_i.a \textit{ or } R_j.b \textit{ (or both) is a clustered} - \textit{indexed column, and } R_i.a \textit{ and } R_j.b \textit{ are comparable, and } 1 \le i,\ j \le K\ \}$$

contain all valid joining column pairs for G_{21}. It is not difficult to show (proof omitted)

Proposition 3 *Let* $\sigma = \sum_{i=1}^{K} |CL_i|$, $\delta_1 = \sum_{i=1}^{K} |CC_i|$, *then* $|\Delta_1| = \delta_1 * (2 * \sigma - \delta_1 + 1)/2$.

Each $\{R_i.a,\ R_j.b\} \in \Delta_1$ is associated with a set $S_{\{R_i.a,\ R_j.b\}}$ of queries in G_{21}, i.e., the queries that have $R_i.a = R_j.b$ as a conjunct in their qualifications. Clearly, $G_{21} = \cup_{\{R_i.a,R_j.b\}\in\Delta_1} S_{\{R_i.a,\ R_j.b\}}$.

If $|\Delta_1| \ge U_{21}$, we randomly select U_{21} joining column pairs in Δ_1. Let Δ_1' be the set of selected joining column pairs. For each $\{R_i.a,\ R_j.b\} \in \Delta_1'$, we choose $\pi_{\tilde{\alpha}^{(ij)}}(R_i \bowtie_{F^{(ij)}} R_j)$ as a sample query to represent the queries in $S_{\{R_i.a,\ R_j.b\}}$, where $F^{(ij)}$ is $[R_i.d\, \theta_1\ \tilde{C}^{(i.d)} \wedge R_i.a = R_j.b \wedge R_j.e\, \theta_2\ \tilde{C}^{(j.e)}]$. Thus the sample drawn from G_{21} in this case is

$$SP_{21} = \cup_{\{R_i.a,R_j.b\}\in\Delta_1'}\{\ \pi_{\tilde{\alpha}^{(ij)}}(R_i \bowtie_{F^{(ij)}} R_j)\ \}. \quad (11)$$

If $|\Delta_1| < U_{21}$, we choose more than one sample query from $S_{\{R_i.a,\ R_j.b\}}$ for some $\{R_i.a,\ R_j.b\} \in \Delta_1$. Since the execution of a sample join query is often expensive, we choose the exact required minimum number of sample queries. To do so, we select a random subset Δ_1'' of Δ_1 such that $(\omega_{21} + 1) * |\Delta_1''| + \omega_{21} * (|\Delta_1| - |\Delta_1''|) = U_{21}$ ($\omega_{21} \ge 1$ integer). It can be easily shown that the condition can be satisfied if $\omega_{21} = \lfloor U_{21}/|\Delta_1| \rfloor$ ($\lfloor\cdot\rfloor$ denotes the floor function) and $|\Delta_1''| = U_{21} - \omega_{21} * |\Delta_1|$. If we choose $\omega_{21} + 1$ sample queries from S_τ for each $\tau \in \Delta_1''$ and ω_{21} sample queries from S_ν for each $\nu \in \Delta_1 - \Delta_1''$, we get a sample of queries with the size U_{21}. The following is the sample of queries drawn from G_{21} in this case:

$$SP_{21} = \cup_{\{R_i.a,R_j.b\}\in\Delta_1''}[\cup_{n=1}^{\omega_{21}+1}\{\pi_{\tilde{\alpha}_n^{(ij)}}(R_i \bowtie_{F_n^{(ij)}} R_j)\}]$$
$$\cup_{\{R_s.a,R_t.b\}\in(\Delta_1-\Delta_1'')}[\cup_{m=1}^{\omega_{21}}\{\pi_{\tilde{\alpha}_m^{(st)}}(R_s \bowtie_{F_m^{(st)}} R_t)\}] \quad (12)$$

where $F_n^{(ij)}$ is $[R_i.d\ \theta_{1n}\ \tilde{C}_n^{(i.d)} \wedge R_i.a = R_j.b \wedge R_j.e\ \theta_{2n}\ \tilde{C}_n^{(j.e)}]$, $F_m^{(st)}$ is $[R_s.d\ \theta_{3m}\ \tilde{C}_m^{(s.d)} \wedge R_s.a = R_t.b \wedge R_t.e\ \theta_{4m}\ \tilde{C}_m^{(t.e)}]$.

Similarly, for the query class G_{22}, let

$$\Delta_2 = \{\ \{R_i.a,\ R_j.b\} \mid R_i.a \textit{ or } R_j.b \textit{ (or both) is an indexed column, and } R_i.a \textit{ and } R_j.b \textit{ are comparable, and } 1 \le i,\ j \le K\ \} - \Delta_1 .$$

Proposition 4 *Let* $\sigma = \sum_{i=1}^{K} |CL_i|$, $\delta_1 = \sum_{i=1}^{K} |CC_i|$, $\delta_2 = \sum_{i=1}^{K} |CI_i|$, *then* $|\Delta_2| = \delta_2 * [2 * (\sigma - \delta_1) - \delta_2 + 1]/2$.

If $|\Delta_2| \ge U_{22}$, a random subset Δ_2' of Δ_2 with the size U_{22} is selected. Replacing Δ_1' by Δ_2' in (11), we get the sample SP_{22} drawn from G_{22} in this case.

If $|\Delta_2| < U_{22}$, a random subset Δ_2'' of Δ_2 with the size of $U_{22} - \omega_{22} * |\Delta_2|$ is selected, where $\omega_{22} = \lfloor U_{22}/|\Delta_2| \rfloor$. Replacing Δ_1, Δ_1'' and ω_{21} by Δ_2, Δ_2'' and ω_{22} in (12) respectively, we get the sample SP_{22} drawn from G_{22} in this case.

For the query class G_{23}, let

$$\Delta_3 = \{\ \{R_i.a,\ R_j.b\} \mid \textit{neither } R_i.a \textit{ nor } R_j.b \textit{ has an (clustered or not) index on it, and } R_i.a \textit{ and } R_j.b \textit{ are comparable, and } 1 \le i,\ j \le K\ \} .$$

It can be shown that $|\Delta_3| = \delta_3 * (\delta_3 + 1)/2$, where $\delta_3 = \sum_{i=1}^{K} |CN_i|$.

If $|\Delta_3| \ge U_{23}$, a random subset Δ_3' of Δ_3 with the size U_{23} is selected. To get the sample SP_{23} drawn from G_{22} in this case, we replace Δ_1' by Δ_3' in (11) and replace $R_i.a = R_j.b$ in $F^{(ij)}$ by $R_i.a\ \theta_3\ R_j.b$.

If $|\Delta_3| < U_{23}$, a random subset Δ_3'' of Δ_3 with the size of $U_{23} - \omega_{23} * |\Delta_3|$ is selected, where $\omega_{23} = \lfloor U_{23}/|\Delta_3| \rfloor$. The sample SP_{23} drawn from G_{23} in this case is obtained by (1) replacing Δ_1, Δ_1'' and ω_{21} by Δ_3, Δ_3'' and ω_{23} in (12) respectively; (2) replacing $R_i.a = R_j.b$ in $F_n^{(ij)}$ and $R_s.a = R_t.b$ in $F_m^{(st)}$ by $R_i.a\ \theta_{5n}\ R_j.b$ and $R_s.a\ \theta_{6m}\ R_t.b$ respectively.

Following the procedure in this section, a sample of queries can be drawn for each query class. However, a sample may include a few extreme queries. such as those whose result tables are extremely large (e.g., millions of tuples). An extreme query is rarely used in practice. Hence, we can remove them from a sample because our purpose is to give good cost estimates for common queries in practice, not the extreme queries. The value U_{tk} should be properly chosen so that the real sample size after removing extreme queries is greater than or equal to 40 for a unary query class or 50 for a join query class.

If the sample queries for a class on a local database system do not follow the performance pattern described by the relevant cost formula, an improved cost formula or a refined classification is required.

5 Experimental Results

In our experiments, the method described in the last three sections was used to estimate the cost parameters of two local database systems (DBS). Each local DBS contains a database, a DBMS and an MDBS agent that provides the global MDBS server with a uniform relational ODBC (Open Database Connectivity) interface. The DBMSs used are Oracle 6.0 and Empress 4.6. They run on IBM RISC System/6000 model 550 and model 220, respectively. All local requests from the global MDBS server are passed to the relevant MDBS agent in a local DBS.

In an MDBS, sometimes, not all local data and functionalities (operations) are exported to the global users because of local autonomy. Two exported local databases and their information are given in Table 1. Queries against DB 1 and DB 2 are classified according to Section 2. There are six classes of queries on DB 1, i.e., $G' = G'_{11} \cup G'_{12} \cup G'_{13} \cup G'_{21} \cup G'_{22} \cup G'_{23}$. There are, however, only two classes of queries on DB 2, i.e., $G'' = G''_{12} \cup G''_{13}$. For each class, a sample of queries are drawn by the sampling procedure described in Section 4 and executed on the corresponding local DBS. Observed costs are used to derive the parameters for the cost formula for each class by multiple regression.

Table 2 shows all cost estimation formulas derived for the query classes on DB 1 and DB 2. The coefficients of multiple determination in Table 2 tell us that most estimation formulas account for over 90% of the variability in the query costs. Even for the worst case G''_{12}, the estimation formula explains about 78% of the variability in the query costs. The standard errors are also acceptable, compared with the magnitudes of the average values of the observed costs. Figure 1 $\sim$ 4 show the comparisons of the estimated costs with the observed costs of some test queries using the cost estimation formulas in Table 2. The estimated costs for majority of the test queries have relative errors below 30%.

The experimental results could be further improved by changing the estimation formulas for some query classes (e.g., G''_{12}) or refining the classification. We also find that (1) the relative errors for small costs may be large if the cost formula is derived by using some large costs; (2) system contention may affect the experimental results significantly. The reason for (1) is that the cost formula is usually dominated by large costs used to derive it, while the small costs may not follow the same the formula because of different buffering and processing strategies used for the queries with small costs. This problem could be resolved by refining the classification according to the sizes of query results, as suggested in Section 2. A simple way to mitigate problem (2) is to perform a sample query multiple times and use the average of the observed costs to derive a cost formula.

The experimental results show that the presented method is quite promising for estimating query costs in an MDBS environment where only limited information is available. Besides, the method is also robust because it could give reasonable cost estimates even if the classification were not ideal.

6 Other Issues

Global query optimization: Section 2 $\sim$ 5 discuss how to derive cost formulas for autonomous local DBSs in an MDBS. A utility in the MDBS can be developed according to the presented method to derive necessary local cost parameters. The utility is invoked when a local database is established and reinvoked when necessary to reflect changes made on the database. Unlike a conventional DDBS, the derived cost parameters are not built into the global query optimizer. They are stored in the global catalog of the MDBS as part of the information about the local database and retrieved during query optimization. One advantage of this approach is that new local DBSs can be easily added into the MDBS. Using the local estimated costs and some other costs (e.g., communication costs), the global query optimizer chooses an execution plan for a query with a cost as low as possible.

Adaptive cost estimation formulas: Since sample queries are usually performed while a local DBS in use, the derived cost parameters contain the contention factor in the system. The contention factor changes over time, which implies that the cost parameters derived long time ago may not reflect the current status. Thus, an estimated cost may be far from a real cost. However, the utility mentioned above can not be invoked very often because performing sample queries on a local DBS may increase the system load. To overcome this problem, the costs of user queries can be

TABLE 1 Local Database Information

Exported Local Database	Tables in Exported Local Database (Ri -- table, |Ri| -- tuple #, |CLi| -- column #, |CIi| -- index #, |CCi| -- clustered-index #)														Exported Operation
DB 1 on Oracle	Ri	1	2	3	4	5	6	7	8	9	10	11	12	13	both unary & join queries
	|Ri|	500	25600	3150	1670	9870	15090	5000	11200	21500	1100	8900	12000	900	
	|CLi|	4	3	6	3	4	5	9	6	3	13	7	3	2	
	|CIi|	1	0	2	0	1	1	3	0	1	0	3	1	0	
	|CCi|	0	1	0	0	1	1	0	1	0	1	1	1	1	
DB 2 on Empress	Ri	1	2	3	4	5	6	7	8	9					unary queries only
	|Ri|	5000	12000	28000	4500	30200	7500	8000	19000	22100					
	|CLi|	5	4	3	6	5	10	3	2	7					
	|CIi|	1	1	1	2	1	4	1	0	3					
	|CCi|	0	0	0	0	0	0	0	0	0					

TABLE 2 Derived Cost Estimation Formulas

DBMS	query class	Cost Estimation Formula (N, N1, N2 -- tuple #, S -- selectivity)	coefficient of multiple determination	standard error (seconds)	average cost (seconds)
Oracle	G11'	0.17480e-1 - 0.43017e-6*N + 0.16116e-2*S*N	0.912680	0.25360e-1	0.99492e-1
	G12'	0.47064e-1 - 0.25280e-5*N + 0.22295e-2*S*N	0.874070	0.55564e-1	0.12308e+0
	G13'	0.10991e+0 + 0.23455e-4*N + 0.14553e-2*S*N	0.988098	0.94005e+0	0.73053e+1
	G21'	0.12026e+1 + 0.11396e-3*N1 + 0.57236e-4*N2 + 0.17102e-2*S*N1*N2	0.997065	0.12582e+2	0.59243e+2
	G22'	0.10800e+1 + 0.25083e-3*N1 + 0.17094e-3*N2 + 0.17254e-2*S*N1*N2	0.999622	0.69946e+1	0.59508e+2
	G23'	0.35339e+1 + 0.31707e-1*N1 + 0.15481e+0*N2 + 0.14705e-2*S*N1*N2	0.996773	0.34880e+4	0.13555e+6
Empress	G12''	0.31663e+0 + 0.22439e-6*N + 0.38784e-2*S*N	0.775814	0.50926e-1	0.46606e+0
	G13''	-0.17823e+1 + 0.33742e-2*N + 0.71374e-3*S*N	0.876967	0.13570e+2	0.52089e+2

Figure 1 Costs of Test Queries in G11 on ORACLE

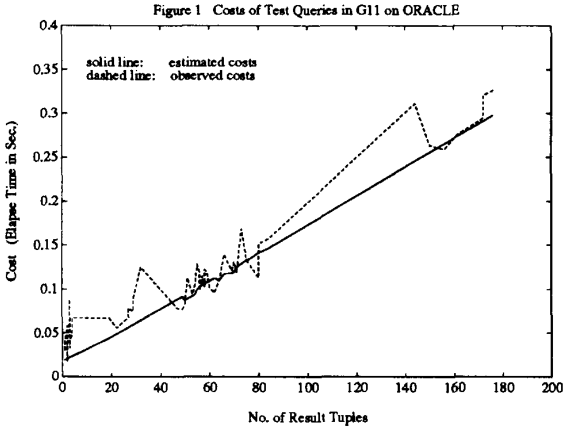


Figure 2 Costs of Test Queries in G22 on ORACLE

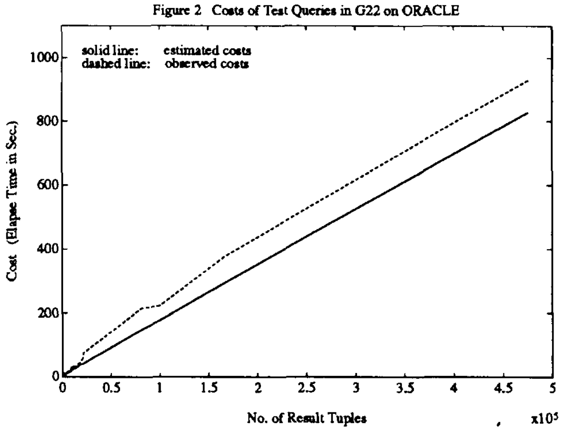


Figure 3 Costs of Test Queries in G12'' on EMPRESS

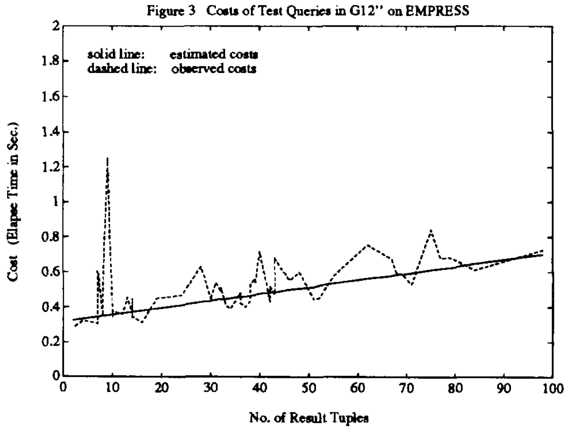
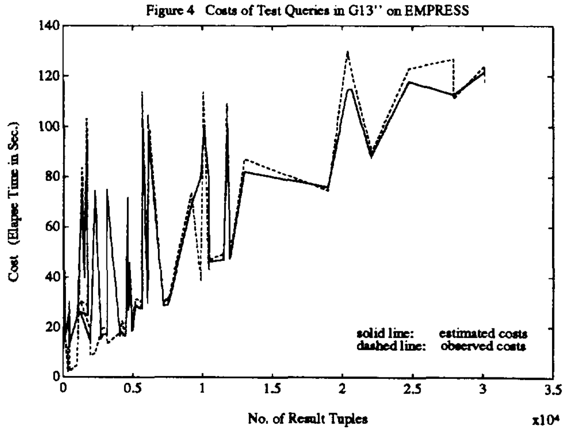


Figure 4 Costs of Test Queries in G13'' on EMPRESS



observed and made use of. The parameters of the relevant cost estimation formula can be adaptively modified by using the costs of user queries as new sample queries.

Estimation of selectivities: To use the derived cost formulas, we need to know the number(s) of tuples in the operand table(s) and the selectivity of the qualification of a given query. The number of tuples usually can be found in the catalog of a local database system. Selectivities can be estimated by a parametric method[1], a table-based method[6, 9] or data-sampling-based methods[4]. However, new issues need to be solved in an MDBS environment, for example, how to draw sample data from local tables efficiently under the restriction that only local external user interfaces can be used if a data-sampling-based method is adopted. These issues are addressed in a separate paper[11], and an integrated method for estimating selectivities in an MDBS is presented in the paper.

7 Conclusion

In this paper we have proposed a method that employs a query sampling technique and multiple regression to estimate the cost parameters of an autonomous local database system in an MDBS. Experimental results show that the presented method is quite promising. Most derived cost formulas account for over 90% of the variability in the query costs. The estimated costs for the majority of the test queries are within 30% error of the real costs.

The advantages of this method are: (1) it only uses information available at the global level in an MDBS — no special privilege is required from a local database system, so local autonomy is preserved; (2) it uses real local databases to derive cost parameters, so the cost formulas are expected to reflect the real situation in practice; (3) the derived local cost parameters are stored in the global catalog instead of built into the global query optimizer, so a new local DBS can easily be added into the MDBS; (4) a cost parameter can be adaptively improved by using observed costs of user queries or new sample queries, so the cost formulas can reflect a dynamically changing environment; (5) the method is robust.

This work is only the beginning of more research that needs to be done to solve the problem of estimating costs in an MDBS. In future work we intend to investigate (1) how to refine a classification by analyzing observed costs; (2) how to dynamically transform a cost estimation formula to reflect a changing environment; (3) how to draw sample queries that represent most practically-used queries by making use of the information, such as foreign keys and knowledge about applications; (4) how to estimate the costs of other operations in real queries, e.g., GROUP-BY, aggregate functions; (5) how to determine the order in which a local DBS executes a sequence of operations.

References

[1] S. Christodoulakis. Estimating record selectivities. *Inf. Sys.*, 8(2):105–115, 1983.

[2] W. Du et al. Query optimization in heterogeneous DBMS. In *Proc. of VLDB*, pp 277–91, 1992.

[3] W. C. Hou et al. Error-constrained COUNT query evaluation in relational databases. In *Proc. of SIGMOD*, pp 278–87, 1991.

[4] R. J. Lipton et al. Practical selectivity estimation through adaptive sampling. In *Proc. of SIGMOD*, pp 1–11, 1990.

[5] H. Lu et al. On global query optimization in multidatabase systems. In *2nd Int'l workshop on Res. Issues on Data Eng.*, p 217, Tempe, 1992.

[6] M. Muralikrishna et al. Equi-Depth histograms for estimating selectivity factors for multi-Dimensional queries. In *Proc. of SIGMOD*, pp 28–36, 1988.

[7] F. Olken et al. Simple random sampling from relational databases. In *Proc. of 12th VLDB*, pp 160–9, 1986.

[8] R. C. Pfaffenberger et al. *Statistical Methods.* IRWIN, 1987.

[9] G. P. Shapiro at al. Accurate estimation of the number of tuples satisfying a condition. In *Proc. of SIGMOD*, pp 256–76, 1984.

[10] Q. Zhu. Query optimization in multidatabase systems. In *Proc. of the 1992 CAS Conference*, vol.II, pp 111–27, Toronto, Nov. 1992.

[11] Q. Zhu. An integrated method of estimating selectivities in a multidatabase system. In *Proc. of the 1993 CAS Conference*, vol.II, pp 832–47 Toronto, Oct. 1993.

[12] Q. Zhu and P.-Å. Larson. Establishing a fuzzy cost model for query optimization in a multidatabase system. In *Proc. of the 27th Hawaii Int'l Conf. on Sys. Sci.*, Maui, Hawaii, Jan. 1994.