An Integrated Method for Estimating Selectivities in a Multidatabase System*

Qiang Zhu

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

Abstract

A multidatabase system (MDBS) integrates information from autonomous local databases managed by different database management systems (MDBS) in a distributed environment. A number of challenges are raised for query optimization in such an MDBS. One of the major challenges is that some local optimization information may not be available at the global level. We recently proposed a query sampling method to drive cost estimation formulas for local databases in an MDBS[22]. To use the derived formulas to estimate the costs of queries, we need to know the selectivities of the qualifications of the queries. Unfortunately, existing methods for estimating selectivities cannot be used efficiently in an MDBS environment. This paper discusses difficulties of estimating selectivities in an MDBS. Based on the discussion, this paper presents an integrated method to estimate selectivities in an MDBS. The method integrates and extends several existing methods so that they can be used in an MDBS efficiently. It extends Christodoulakis' parametric method so that estimation accuracy is improved and more types of queries can be handled. It extends Lipton and Naughton's adaptive sampling method so that both performance and accuracy are improved. Theoretical and experimental results show that the extended Lipton and Naughton's method described in this paper can be many times faster than the original one. In addition, the integrated method uses a new piggyback approach to collect and maintain statistics, which can reduce the statistic maintenance cost. The integrated method is designed for the MDBS in the CORDS project (CORDS-MDBS). Implementation considerations are also given in the paper.

Keywords: multidatabase system, query optimization, cost estimation, selectivity, data sampling.

1 Introduction

A *multidatabase system* (MDBS) integrates information from pre-existing autonomous *local (component) databases* managed by heterogeneous database management systems (DBMS), such as Oracle, DB2, Empress, IMS, in a distributed environment. It acts as a front end to the multiple local DBMSs to provide 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[2]. Most differences between a conventional distributed database system (DDBS) and an MDBS are caused by local autonomy. These differences raise new challenges for query optimization in an MDBS[13, 21]. However, to date only a few

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

papers have been published on query optimization in an MDBS[4, 12, 13, 21]. Many issues remain unsolved.

Among the many challenges for query optimization in an MDBS, the crucial one is that some local query optimization information, for example, local cost functions and some statistics about local databases, may not be available to the global query optimizer in the MDBS because of local autonomy. It is, therefore, difficult for the global query optimizer to determine a good execution plan for a global (multidatabase) query. There is no such problem in a conventional DDBS because all the sites run the same distributed database management system (DDBMS), which has control over all local databases. The query optimizer in such a DDBS can make use of both global and local information to produce a good execution plan for a given query. In an MDBS, new methods to derive or estimate local optimization information are required.

In [22], we proposed a query sampling method to derive local cost estimation formulas for autonomous local database systems in an MDBS. The costs of local queries resulting from a decomposition of a global query can be estimated by the cost estimation formulas. Based on local and other costs (like communication costs), the cost of an execution plan for a given query can be estimated. With the estimation, the global query optimizer in an MDBS can choose a good (low cost) execution plan from many alternatives. Simulation results show that this approach is quite promising[22].

To use the local cost estimation formulas to estimate the cost of a (local) query, however, requires the selectivity of the qualification of the query as an input. The selectivity of the qualification of a query is the percentage of the tuples satisfying the qualification in the operand table (for a unary query) or the Cartesian product of the operand tables (for a multi-dimensional query). Usually a selectivity is known after the relevant query is executed, while the global query optimizer needs the selectivities of local queries to choose an execution plan for a global query before executing the local queries. Therefore, good estimates of selectivities before the execution of the relevant queries are required.

For a query optimizer in a traditional DBMS or DDBS, there are three types of methods proposed, so far, for estimating selectivities: parametric methods, table(histogram)-based methods and sampling-based methods. A selectivity, in fact, reflects some properties of the underlying data. To estimate a selectivity, thus, needs some information about the underlying data.

A parametric method[3, 14, 17] makes assumptions about the underlying data, for example, uniform distribution and independence of columns, and it then uses certain formulas with parameters to estimate selectivities. This method is simple and efficient. However, its estimates may be inaccurate if the assumptions do not hold. Therefore, it may not be always feasible in an MDBS because it is sometimes hard to decide which assumptions are suitable for the data in an autonomous local database.

A table-based method[15, 19] scans (maybe part of) underlying data periodically to collect necessary statistics in a table and uses the statistics to estimate selectivities. This method makes no assumptions about the underlying data and may give better estimates than a parametric method. However, it would require storing and maintaining a large amount of detailed statistics about local databases if it were used in an MDBS. This requirement is hard to meet efficiently in a distributed and complicated MDBS environment.

A sampling-based method[6, 7, 8, 10, 11] performs a given query on a sample of underlying data and uses the query result to estimate the selectivity for the query. This method makes no assumptions about the underlying data and does not require storing and maintaining detailed statistics. Its estimates are usually accurate. However, it increases the cost of query optimization because the sampling is performed during optimization. The main difficulty of using this method in global query optimization in an MDBS is that it may be hard to find an efficient way to draw sample data from a local database because local storage structures of the database may be unknown or cannot be changed at the global level in an MDBS, unlike a DBMS or traditional DDBS in which the B^+-tree indexes, virtual columns of tables, and block readings can be used to implement a quite efficient sampling procedure.

Therefore, no single existing method can solve the problem of estimating selectivities in an MDBS well. How to design a good estimator of selectivities in an MDBS environment is the main topic of this paper. No such estimator has been found in the literature for an MDBS. We develop a feasible integrated method for estimating selectivities in an MDBS by integrating and extending several existing methods. This integrated method is designed for the Multidatabase System for the CORDS project (CORDS-MDBS). It preserves the advantages of the existing methods and improves performance and estimation accuracy.

In CORDS-MDBS, there is an MDBS agent for each local DBMS, which provides a uniform relational interface for the global query optimizer despite the fact that a local DBMS itself may not be relational[21]. Thus, in this paper we view all local DBMSs as relational ones.

The rest of this paper is organized as follows. Section 2 reviews the main idea of the query sampling method we proposed in [22]. Section 3 derives the formulas used to estimate selectivities for unary queries and discusses the methods to maintain necessary statistics. Section 4 extends Lipton and Naughton's method for estimating selectivities for join queries and gives some simulation results. Section 5 discusses how to incorporate the methods presented in the previous sections into an implementation of CORDS-MDBS. A summary is given in the last section.

2 Multiple Regression Cost Model

As mentioned above, local cost functions may not be available at the global level in an MDBS because of local autonomy. A query sampling method is proposed in [22] to derive a multiple regression cost model for an autonomous local database system in an MDBS. The idea is to

1. divide queries on a local database into classes such that the costs of the queries in each class can be estimated by using the same formula,
2. draw sample queries from each query class,
3. use the observed costs of the sample queries and the multiple regression method in statistics to derive local cost estimation formulas for the queries performed on the local database.

Clearly, the costs of the queries that are executed by using the same access method, e.g., index-based scan or nested-loop join, can be estimated by using the same formula. Queries could be classified according to the access methods used. Unfortunately, the access method to be used for a local query may not be known at the global level in an MDBS. It depends on which local DBMS is used. As a regular local user, the MDBS has limited information available. Fortunately, there is still some available information that can be used to classify queries. The following are three types of such information:

- *characteristics of queries*: such as unary (projection, selection) queries or 2-way join queries. This information can be obtained from analyzing a given query.
- *characteristics of operand tables*: such as the number of columns, the number of tuples, and indexed columns in a table referred to by a query. This information can be obtained from a local or global catalogue.
- *characteristics of local DBMSs*: such as the types of access methods supported. This information can be obtained from the specification (manual) of a local DBMS and stored in the global catalogue. Note that some supported local access methods

and buffering strategies may not be visible to the MDBS.

Based on the available information, queries can be classified. The costs of queries in each class are estimated by using the same formula. If the available information is sufficient, we can classify queries such that each class corresponds to one access method. The cost estimates are expected to be quite accurate in this case. If the available information is not sufficient, it is possible that queries using different access methods can be put in the same class. In this case, it is better to put the queries using the access methods with similar performance behavior into the same class so that estimation errors are not expected to be large. Since the practical goal of query optimization is to avoid bad execution plans instead of to achieve a real optimal execution plan, estimation errors can be tolerated to a certain degree.

Since most common queries can be expressed by a sequence of selection (σ), projection (π), and join ($\bowtie$) operations, it is sufficient to consider these three types of operations only. The cost of a general query can be estimated by composing the costs of its component unary (σ, π) and binary ($\bowtie$) queries.

Let G be the set of all (unary and join) queries on a local DB i managed by a local DBMS j. Let R and S be 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. Without loss of generality, qualifications of (unary and join) 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 $\theta \in \{=, \neq, >, <, \geq, \leq\}$.

After a careful analysis, the following classification of queries are suggested in [22]:

$$G = G_{11} \cup G_{12} \cup G_{13} \cup G_{21} \cup G_{22} \cup G_{23}$$

where

$$\begin{aligned}
G_{11} &= \{\ \pi_\alpha(\sigma_F(R)) \mid F \textit{ has at least one conjunct } R.a = C, \textit{ where } R.a \textit{ is a clustered}-\textit{indexed column}\ \},\\
G_{12} &= \{\ \pi_\alpha(\sigma_F(R)) \mid \pi_\alpha(\sigma_F(R)) \textit{ not in } G_{11} \textit{ and } F \textit{ has at least one conjunct } R.a = C, \textit{ where } R.a \textit{ is an indexed column}\ \},\\
G_{13} &= \{\ \pi_\alpha(\sigma_F(R)) \mid \pi_\alpha(\sigma_F(R)) \textit{ not in } (\ G_{11} \cup G_{12}\ )\ \}.\\
G_{21} &= \{\ \pi_\alpha(R \bowtie_F S) \mid F \textit{ has at least one conjunct } R.a = S.b, \textit{ where } R.a \textit{ or } S.b \textit{ (or both) is a clustered}-\textit{indexed column}\ \},\\
G_{22} &= \{\ \pi_\alpha(R \bowtie_F S) \mid \pi_\alpha(R \bowtie_F S) \textit{ not in } G_{21} \textit{ and } F \textit{ has at least one conjunct } R.a = S.b, \textit{ where } R.a \textit{ or } S.b \textit{ (or both) is an indexed column}\},\\
G_{23} &= \{\ \pi_\alpha(R \bowtie_F S) \mid \pi_\alpha(R \bowtie_F S) \textit{ not in } (\ G_{21} \cup G_{22}\ )\ \}.
\end{aligned}$$

This classification is based on common available information. If more information about a local database system is available, the classification can be further refined.

For each query class, a sample of queries is drawn. For example, the following sample of queries is drawn from the query class G_{11}:

$$SP_{11} = \cup_{i=1}^{K} [\cup_{R_i.a\ \in\ CC_i'}\ \{\ \pi_\alpha(\sigma_{R_i.a=C_0}(R_i))\ \}]. \quad (1)$$

where R_1, $\cdots$, R_K are all the tables in the local database, CC_i' is a selected subset of the set of clustered-indexed columns in R_i, C_0 is a random constant in the domain of the column $R_i.a$, and α is a random subset of the set of columns in R_i. Sample queries are performed on the local database. Their costs are observed and used to estimate the regression coefficients of the following equations by multiple regression:

$$\begin{aligned}\widehat{Y}_{1k} &= \beta 0_{1k} + \beta 1_{1k} * N1_{1k}\\ &+ \beta 2_{1k} * S_{1k} * N1_{1k}, \qquad (k = 1, 2, 3) \quad (2)\end{aligned}$$

and

$$\begin{aligned}\widehat{Y}_{2k} &= \beta 0_{2k} + \beta 1_{2k} * N1_{2k}\\ &+ \beta 2_{2k} * N2_{2k} + \beta 3 * S_{2k} * N1_{2k} * N2_{2k},\\ & \qquad (k = 1, 2, 3) \quad (3)\end{aligned}$$

where $\widehat{Y}_{jk}$ ($j = 1, 2;\ k = 1, 2, 3$) is the estimated cost for a query in the class G_{jk}, S_{jk} is the selectivity of the query, Ni_{jk} ($i = 1, 2$) is the number of tuples in the ith operand table of the query, βi_{1k} ($i = 0, 1, 2$) and βi_{2k} ($i = 0, 1, 2, 3$) are the regression coefficients. (2) and (3) are then used to estimate the costs of queries.

Figure 1 shows the estimated costs obtained by using (2) for some test queries on Oracle 6.0 (more details and other experimental results can be found in [22]). Experimental results demonstrate that the above query sampling method is quite promising for estimating local query costs in an MDBS. Although sampling techniques for query optimization can be found in the literature, all of them consider data sampling, that is, sample data from a database. Using query sampling, that is, sample queries from a query class, is our new idea.

For a given local unary or join query, the global query optimizer first identifies the query class it belongs to, then uses one of the formulas (2) and (3) to estimate the cost of the query. For a more general local query that consists of a sequence of unary and join subqueries, the global query optimizer can estimate the cost of the query by composing the costs of the subqueries. Based on the estimated costs, the global query optimizer can choose a good execution plan for a global query.

To use (2) and (3) to estimate the cost of a query, the selectivity S_{jk} of the query is required as an input. The following two sections discuss the method designed for CORDS-MDBS to estimate selectivities for unary queries and join queries, respectively.

3 Estimating Selectivities for Unary Queries

3.1 Alternative Methods

There are a number of possible methods of estimating selectivities for unary queries. However, not all of them can be applied effectively in an MDBS.

As mentioned before, a table-based method[15, 19] collects detailed statistics about underlying data and estimates a selectivity by looking up the table(s) containing the statistics. If this method were used in an MDBS, the following problems would occur:

- *How to collect and update many statistics about data in local databases.* Unlike a centralized DBMS, an MDBS integrates information from many local databases managed by different local DBMSs. There is usually a very large number of data accessible by an MDBS. It is very difficult to maintain detailed statistics about so many data. In other words, maintaining them causes too much overhead.

- *How to store the statistics in an MDBS.* Since there are so many detailed statistics that need to be kept, it is not suitable to keep them in the global catalogue. It appears that a separate statistical database is required for each local database. However, it is possible that such a statistical database cannot be stored in the site of the local database. Managing such statistical databases greatly increases the complexity of the MDBS.

- *How to retrieve the statistics in an MDBS.* To speed up the retrieval of statistics, some statistics may be replicated in a number of sites. How to replicate statistics and how to find required statistics need to be addressed.

Because of these problems, a table-based method will not be adopted in our MDBS.

A sampling-based method[6, 7, 8, 10, 11] draws a sample from underlying data and estimates a selectivity by observing the behavior of the relevant query on the sample. However, whether a sample can be drawn from a local database efficiently without knowing implementation, details of the database determines if this method is suitable for an MDBS. Unfortunately, for a unary query, there is no way to guarantee that sample tuples can be efficiently drawn from the operand table without knowing

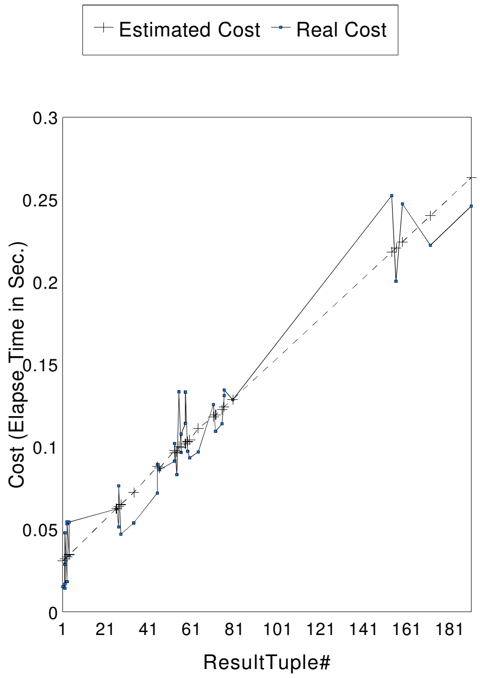

Figure 1 Costs of Test Queries in G11 on ORACLE

how the table is implemented in the database. For example, using a sampling-based method to estimate the selectivity of the qualification of the query $Q: \sigma_{R.a>4}(R)$ on table R may require drawing a sample of tuples from R. Since an MDBS can only interact with a local DBMS at its external interface, drawing a tuple from R may be expressed as a query $Q_s : \sigma_{R.a=C}(R)$ where C is a random value in the domain of $R.a$. The MDBS has no control on how to execute Q_s on the local DBMS because of local autonomy. The local DBMS may execute Q_s by scanning the whole table R. Therefore, the cost of drawing a sample tuple may be the same as the cost of executing the original query Q. The sampling-based method is obviously not suitable in this case. A sample tuple can be efficiently drawn from a table if there is an indexed dense key column[11][1] or a B^+-tree indexed column[11, 16] of the table. This condition cannot be guaranteed in an MDBS. Although we will not use a sampling-based method to estimate selectivities for unary queries because of the above observation, this method can be useful in estimating selectivities of join queries, as we will see in the next section.

A parametric method uses a set of formulas to estimate selectivities under some assumptions about the underlying data. It is simple and efficient. If the assumptions are met, it can give accurate estimates. After comparing possible estimation methods, we feel that a parametric method is suitable for estimating selectivities of unary queries in an MDBS. Issues that need to be considered are: what assumptions are appropriate to the underlying data, what estimation formulas are to be used, and how can the necessary simple statistics be maintained in the catalogue. In the next subsection, we first give the estimation formulas to be used in our MDBS. The other issues will be discussed later.

[1] By "dense" we mean that there are no gaps between two consecutive values appearing in the table.

3.2 Estimation Formulas

As in [22], the unary queries considered in this paper are of the form: $\pi_\alpha(\sigma_F(R))$, where α is a list of target columns from the table R and F is a qualification condition that is in the conjunctive normal form with basic predicates. A basic predicate is of the form: $R.a\ \theta\ C$, where $\theta \in \{=, \neq, <, >, \leq, \geq\}$, and C is a constant in the domain of $R.a$.

There are usually two assumptions about the underlying data. One is the *distribution assumption*, which assumes the values of a column in an operand table follow a probability distribution, such as a uniform distribution. The other is the *independence assumption*, which assumes the values of two columns are independent. Like most database systems, the independence assumption is used in our MDBS. However, unlike many database systems, the values of different columns are allowed to follow different probability distributions instead of only one probability distribution. For different distributions, we will use different formulas to estimate selectivities.

3.2.1 Uniform Distribution

An often-used distribution for underlying data in a database is the uniform distribution, which assumes that each value appears in a column of a table with equal probability. Under this assumption, the estimation formulas for the selectivities of the basic predicates can be given by generalizing those in [14, 17]. Let $S(X)$ denote the selectivity of the qualification condition X. Then we have the following estimation formulas:

$$S(R.a = C) \approx \frac{|R|}{DV(R.a)} , \qquad (4)$$

$$S(R.a > C) \approx \frac{Max(R.a) - C}{Max(R.a) - Min(R.a)} , \qquad (5)$$

where $|R|$ is the cardinality of R, $DV(R.a)$ is the number of distinct values of $R.a$ in R, and $Max(R.a)$ and $Min(R.a)$ are the maximum and minimum values of $R.a$, respectively. For the conditions connected by logic connectives, we have

$$S(NOT\ X) = 1 - S(X) , \qquad (6)$$

$$S(X_1\ AND\ X_2) = S(X_1) * S(X_2) , \qquad (7)$$

$$S(X_1\ OR\ X_2) = S(X_1) + S(X_2) - S(X_1) * S(X_2) . \qquad (8)$$

Formulas (7) and (8) require the independence assumption. From (4) $\sim$ (6) and (8), selectivities $S(R.a \neq C)$, $S(R.a \geq C)$, $S(R.a < C)$, and $S(R.a \leq C)$ can be estimated. Therefore, the selectivity for any query $\pi_\alpha(\sigma_F(R))$ (i.e., the selectivity of the qualification: $S(F)$), can be estimated by using formulas (4) $\sim$ (8).

To use the above estimation formulas, we need to know some simple statistics about underlying data, that is, $|R|$, $DV(R.a)$, Max $(R.a)$, and $Min(R.a)$. In the literature[14, 17], it is assumed that the last three statistics can be obtained by using an index on $R.a$ and the first statistic can be found in a catalogue. An MDBS is different. Even if we know there exists an index on $R.a$, we cannot use it to get the statistics because we cannot access the physical index itself. How to obtain and maintain these statistics will be discussed later.

3.2.2 Non-Uniform Distribution

In many cases, the underlying data are not uniformly distributed. Often the data may follow a normal distribution or different types of Pearson distributions[3]. How can we estimate selectivities in these cases? Let $p(x)$ be the probability density function of the distribution fitted by the values of $R.a$. Then

$$S(R.a\ \theta_1\ C_1\ AND\ R.a\ \theta_2\ C_2) \approx \int_{C_1}^{C_2} p(x)dx , \qquad (9)$$

where $\theta_1 \in \{\geq, >\}$, $\theta_2 \in \{\leq, <\}$, $C_1 < C_2$ are two constants in the domain of $R.a$. From (6), (8), and (9), we can estimate all the basic predicates considered, because

$$S(R.a = C) = S(R.a \geq C\ AND\ R.a < C') , \qquad (10)$$

$$S(R.a > C) = S(R.a > C\ AND\ R.a \leq Max(R.a))\ , \quad (11)$$

where C' is a value less than the smallest upper value of C appearing in the table R. In practice, it may be difficult to find a required C'. A value close to C can be taken as an approximation to C'.

To evaluate the integral in (9), we consider the following three cases:

1) *An antiderivative $P(x)$ of $p(x)$ can be exactly derived.* In this case,

$$\int_{C_1}^{C_2} p(x)dx = P(C_2) - P(C_1)\ . \quad (12)$$

For example, for an exponential distribution $p(x) = 0.8e^{-0.8x}(x \geq 0)$,

$$P(x) = \int_0^x 0.8e^{-0.8x}dx = 1 - e^{-0.8x}.$$

Thus

$$\begin{aligned} \int_1^5 p(x)dx &= P(5) - P(1) \\ &= e^{-0.8*1} - e^{-0.8*5} \\ &= 0.431013\ . \end{aligned}$$

2) *$P(x)$ is difficult to derive exactly, but $C_2 - C_1$ is small* ((10) is the most likely case). In this case, we apply the Trapezoidal Rule[1] with one trapezoid to numerically approximate the integral in (9), namely,

$$\int_{C_1}^{C_2} p(x)dx \approx \frac{1}{2}[p(C_1) + p(C_2)](C_2 - C_1)\ . \quad (13)$$

For example, for a normal distribution $p(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2}$ $(-\infty < x < \infty)$,

$$\begin{aligned} \int_{1.0}^{1.2} p(x)dx &\approx \frac{1}{2}(\frac{1}{\sqrt{2\pi}}e^{-1.2^2/2} \\ &\quad + \frac{1}{\sqrt{2\pi}}e^{-1.0^2/2})(1.2 - 1.0) \\ &= 0.0436156\ . \end{aligned}$$

3) *$P(x)$ is difficult to derive exactly, and $C_2 - C_1$ is not small* ((11) is the most likely case). In this case, we partition the interval $[C_1,\ C_2]$ into n equal subintervals such that $h = (C_2 - C_1)/n$ is small. By the Trapezoidal Rule with n trapezoids, we have

$$\begin{aligned} \int_{C_1}^{C_2} p(x)dx \approx \\ [\tfrac{1}{2}p(C_1) + p(C_1 + h) + p(C_1 + 2h) \\ + \cdots + p(C_1 + (n-1)h) \\ + \tfrac{1}{2}p(C_2)] * h\ . \end{aligned} \quad (14)$$

Clearly, (13) is a special case of (14) with $h = C_2 - C_1$. The smaller h is, the better the approximation to the integral would be. However, more computations in (14) will be involved for small h. Trade-off is necessary in choosing a proper h.

Using formulas (6) $\sim$ (14), we can estimate selectivities for any of our unary queries.

The above estimation method can be viewed as an extension to the method proposed by Christodoulakis in [3]. We extend the latter method in the following aspects:

- Christodoulakis' method is valid only for the query qualification $C \leq R.a \leq C + \Delta C$ with small ΔC, while our method is good for any ΔC. In fact, our method is valid for all qualifications that we consider.

- Christodoulakis only considered the normal distribution and several types of Pearson distributions, while our method, in principle, can be applied to any probability distribution.

- Christodoulakis' method uses a rough Riemann sum approximation to the integral of the probability density function, while our method employs the more accurate and efficient Trapezoidal Rule.

To use the above formulas, besides $Max(R.a)$ and $Min(R.a)$, some other simple statistics (like mean value and variance of underlying data) may also need to be maintained because some probability density functions may use them as parameters. How to maintain simple statistics in our MDBS is the issue discussed in the next subsection

3.3 Maintaining Statistics

As mentioned above, in order to use the formulas in the last two subsections to estimate selectivities, some simple statistics (such as maximum, minimum, mean, variance, the number, distribution type) about underlying data need to be obtained and maintained in the MDBS. Since there are not many statistics required for each column or table, they can be stored in the global catalogue together with schema information. They can be retrieved in the same way as schema information when the global query optimizer estimates the cost of a local query involved in an execution plan for a global query.

There are two differences between statistical information and schema information:

- Schema information should be exact in principle, while statistical information can tolerate certain imprecision because it is used for the estimation purpose.
- Schema information is changed infrequently, while statistical information may be changed every time the underlying data are changed (for example, insert, delete, or update) if accuracy is required.

To reduce the cost of maintenance, as in many other database systems, certain imprecise statistical information will be tolerated in our MDBS.

Two methods are adopted in our MDBS to obtain and maintain statistics:

- *A statistic utility is periodically invoked on the underlying database to collect and update the statistics stored in the catalogue.* The statistic utility not only computes the necessary statistics, like maximum and minimum, but also identifies what type of probability distribution the underlying data fit (if the user did not specify) by frequency ranking or regression analysis. However, the execution of the statistic utility increases the system load, which implies that the overall system performance may be degraded. Thus the statistic utility cannot be invoked very often and should be invoked when the system is not very busy (such as at night). Imprecise (out-of-date) statistics may, therefore, be used in an estimation procedure.
- *A piggyback method is used to collect statistics during query processing.* Besides making use of intermediate results of processing a given query, this method may also perform some additional side retrievals on underlying data during the query processing. Although additional side retrievals are not related to the processing of the query and may slow down the query processing slightly, the statistics collected from the results of the additional side retrievals can be used to improve the processing of many other queries. For example, a user issues the following global query Q on an MDBS:

  SELECT $R_1.a_1$, $R_1.a_2$
  FROM R_1
  WHERE $R_1.a_3$ IN (SELECT $R_2.b_1$
  FROM R_2) ;

  where $R_1(a_1,\ a_2,\ a_3)$ and $R_2(b_1,\ b_2)$ are two tables in two local databases at two different sites S_1 and S_2, respectively. One feasible execution plan for the query is performing the local query Q_l: *SELECT* $R_2.b_1$ *FROM* R_2 on site S_2, then transferring the result to S_1 and performing a final join there. Clearly, the MDBS can collect or update statistics about the column $R_2.b_1$ by observing the result of query Q_l during processing Q. To obtain statistics about the other column b_2 in R_2, the MDBS can perform the query Q_l' : *SELECT* $R_2.b_1$, $R_2.b_2$ *FROM* R_2 on site S_2 instead of Q_l, then analyze the result. Since both Q_l and Q_l' usually scan the table R_2 once, Q_l' increases the processing cost of the given query Q only slightly. In this way, statistics for the data referred to by popular queries can be maintained accurately and efficiently in the global catalogue, which mitigates the problem of out-of-date statistics caused by the restrictions of the first method.

The first method can be found in several existing DBMSs, while the second method is our new idea. Although modifying cost parameters by using runtime information during the execution of a query can be found in previous work[20], riding additional side retrievals piggyback on query processing has not been found in the literature.

4 Estimating Selectivities for Join Queries

In the last section, we applied a parametric method to estimate selectivities for unary queries. However, a parametric method usually cannot accurately estimate selectivities for join queries. For example, the selectivity of the join query $R_1 \bowtie_{R_1.a=R_2.b} R_2$ was estimated as $1/max\{DV(R_1.a),\ DV(R_2.b)\}$ (that is true only if each value in the column with the smaller cardinality has a matching value in the other column) in the formula of Selinger *et al.*[18] under the uniform assumption and was estimated as a constant value 0.3 (that has little significance) in the formula of Makinouchi *et al.*[14], and could not be estimated in Christodoulakis' formula[3].

As discussed before, a sampling-based method can give accurate estimates but may not be useful for estimating selectivities for unary queries because, drawing a sample from a table may require scanning the whole table, while it is sufficient to evaluate the original query on the table by scanning the table once. However, a sampling-based method may be beneficial for estimating selectivities for join queries, because evaluating a join query usually requires scanning some underlying data (for example, one of the operand tables) several times, and it is possible to reduce the number of times that the table is scanned. We will use a sampling-based method to estimate selectivities for join queries in our MDBS.

Two types of sampling-based methods for estimating selectivities are described in the literature. All were designed for (centralized) DBMSs. One type uses the sequential sampling technique[5, 9, 10, 11]. It is characterized by its sequential sample gathering and its stopping condition. That is, sample units are taken one at a time. Checking the outcome of each sample unit allows a decision to be made as to whether an additional sample unit is to be taken. The stopping criterion sets a lower bound on the required sample size for a given error constraint. Another type of methods uses the double sampling technique[7]. The sampling is conducted in two stages. In the first stage, a small sample is taken to estimate preliminary information of the data population, such as the mean and variance. Based on this preliminary information, the required sample size that guarantees that the estimate meets the precision requirement with a certain confidence level is computed. In the second stage, additional sample units are taken, and the final estimate is produced. A shortcoming of a double sampling method is that no theoretical guideline is available to determine the amount of sampling in the first stage, while a good estimation of the preliminary information of the data in the first stage is essential to the estimation accuracy. Because of this problem of the double sampling method and the simplicity of a sequential sampling method, a sequential sampling method is adopted in our MDBS.

We extend the sequential sampling method proposed by Lipton and Naughton in [9, 10, 11] (they call it *adaptive sampling*) so that it can be used efficiently in an MDBS environment. For a join query $Q\ :\ \ R_1 \bowtie_F R_2$, the data population is the Cartesian product $R_1 \times R_2$. In Lipton and Naughton's method, each tuple r in R_1 incurs a sample unit $S_r = \{x \mid x\ is\ a\ concatenation\ of\ r\ and\ a\ tuple\ in\ R_2\}$ in $R_1 \times R_2$. Let $\bar{S}_r$ be the tuples in S_r that satisfy the qualification F. Lipton and Naughton's method repetitively draws a tuple r from R_1 (thus a sample unit S_r from $R_1 \times R_2$) and evaluates the qualified tuples $\bar{S}_r$ in S_r until the total number of qualified tuples obtained so far meets certain stopping conditions for a given error constraint. There are two stopping conditions in Lipton and Naughton's method, one for the cases in which a reasonable number of S_r have non-empty $\bar{S}_r$, and the other for those cases in which few S_r have non-empty $\bar{S}_r$. Without loss of generality, we consider here only the first stopping condition $y > k_1 \cdot b \cdot (1+e)/e^2$, where y is the number of qualified tuples accumulated so far, k_1 is a value associated with a given confidence level [2], b is the maximum size of all $\bar{S}_r$, and e is the desired limit of a relative error.

Lipton and Naughton's method was designed for a (centralized) DBMS. In a DBMS, it can make use of a B^+-tree index on a column of R_1, or disk page readings, or an index on an internal dense key column of R_1 to sample a tuple from R_1 without scanning the whole R_1. It is, hence, quite efficient. In an MDBS, however, the local storage structure of a table is not visible and cannot be changed at the global level. To sample a tuple from R_1, the whole table R_1 is usually scanned. Similarly, to find the tuples in R_2 that match a sample tuple in R_1, the whole table R_2 is usually scanned. Therefore, the lower bound of the complexity of Lipton and Naughton's method in an MDBS is $|R_1| + |R_2|$ (assume each tuple retrieval requires an I/O) where $|R_i|$ $(i = 1, 2)$ denotes the cardinality of R_i. If N tuples need to be drawn from R_1 so that the stopping condition is satisfied, the complexity of Lipton and Naughton's method in an MDBS is: $N * (|R_1| + |R_2|)$. Although this complexity is better than the complexity of computing the join query itself, that is $|R_1| * |R_2|$, because N is usually much smaller than $|R_1|$, Lipton and Naughton's method may still be quite slow in an MDBS because N can be a large number.

To improve Lipton and Naughton's method in an MDBS, we extend the method via the following three mechanisms:

- use a systematic sampling (instead of the simple sampling in the original Lipton and Naughton's method) to draw sample tuples from R_1.
- use a buffer B to hold relevant column values of a number of sample tuples from R_1.
- find matching tuples from R_2 for a number of sample tuples from R_1 (instead of just one sample tuple in the original Lipton and Naughton's method) during each scan of the table R_2.

For simplicity, let us consider the join query $R_1 \bowtie_{R_1.a=R_2.b} R_2$. It is not difficult to generalize the method for a general join query. Let B be a buffer that can hold m values of $R_1.a$ (assume $m < |R_1|$), and $K = \lceil |R_1|/m \rceil$ ($\lceil \cdot \rceil$ denotes the ceiling function). Our extension of Lipton and Naughton's method performs the following steps:

(a) initialize the variables $total\_match = 0$ and $total\_sample = 0$.

(b) choose a random number γ between 1 and K.

(c) execute the following query on R_1:

SELECT $R_1.a$
FROM R_1 ;

(d) if $K = |R_1|/m$, hold in buffer B the values of $R_1.a$ for the γth tuple, the $(\gamma + K)$th tuple, $\cdots$, and the $(\gamma + (m-1) * K)$th tuple retrieved from R_1.

(e) if $K > |R_1|/m$, hold in buffer B the values of $R_1.a$ for the γth tuple, $\cdots$, the $(\gamma + (m-2) * K)$th tuple, and a tuple randomly chosen among the $(\gamma + (m-1) * K)$th $\cdots$ $|R_1|$th tuples retrieved from R_1.

(f) execute the following query to count the number of the tuples in R_2 that match one of the sample tuples whose $R.a$ values are held in B:

[2] $k_1 \geq [\Phi^{-1}(\frac{1+\sqrt(p)}{2})]^2$ if the central limit approximation applies[11], where $\Phi(a) = 1/2\pi \int_{-\infty}^{a} e^{x^2/2} dx$ and p is the confidence level, otherwise $k_1 = 1/(1-\sqrt{p})$.

SELECT COUNT(*) INTO :x
FROM R_2
WHERE $R_2.b = C_1$ OR $R_2.b = C_2$
OR $\cdots$ OR $R_2.b = C_m$;

where C_i $(1 \leq i \leq m)$ are the values of $R_1.a$ held in the buffer B.

(g) $total\_match = total\_match + x$, and $total\_sample = total\_sample + m$.

(h) if $total\_match$ satisfies the stopping condition, that is, $total\_match > k_1 \cdot b \cdot (1+e)/e^2$, then

$$selectivity\_estimate = total\_match/(total\_sample * |R_2|) ,$$

and stop; otherwise go to step (b).

The systematic sampling used in step (d) and (e) allows us to draw m sample tuples during one scan of R_1. It would scan R_1 m times if the original Lipton and Naughton's method were used. The query in step (f) finds the number of the tuples in R_2 that match at least one of the m sample tuples from R_1 during one scan of R_2. It would scan R_2 m times if the original Lipton and Naughton's method were used. For the purpose of comparison, assume that each tuple in R_1 has about the same number of matching tuples in R_2. If N tuples from R_1 need to be drawn to satisfy the stopping condition, the complexity of the original Lipton and Naughton's method, as said before, is $N * (|R_1| + |R_2|)$, while the complexity of the extended Lipton and Naughton's method is $\lceil N/m \rceil * (|R_1| + |R_2|)$. Therefore, the complexity of the extended Lipton and Naughton's method can be up to m times smaller than that of the original Lipton and Naughton's method. If $N \leq m$, the complexity of the extended Lipton and Naughton's method reaches the lower bound, that is, $|R_1| + |R_2|$. Furthermore, the extended Lipton and Naughton's method also improves accuracy of estimation because it may draw more sample tuples (without increasing cost) before the stopping condition is met than the original method. In summary, the extended Lipton and Naughton's method presented above is faster and more accurate than the original one. Note that, if $m = 1$, the extended method reduces to the original one.

Simulation experiments to show the improved performance are performed on a local database managed by the relational DBMS Oracle 6.0 on IBM RISC System/6000 model 550. The original and extended Lipton and Naughton's methods are used for the join query $R_1 \bowtie_{R_1.a=R_2.b} R_2$ where R_1 and R_2 are two tables in the local database with 500 and 1670 tuples, respectively. For different numbers (N) of sample units drawn from $R_1 \times R_2$, a performance comparison of the methods is shown in Figure 2. The figure demonstrates that the extended Lipton and Naughton's method is much better than the original one, and performance increases as m increases.

5 Implementation Consideration

Figure 3 shows how to incorporate the methods presented in the previous sections into our MDBS. A statistical utility, called Statistics Collector, is invoked periodically. It collects local statistics and derives local cost parameters by performing probing or sampling queries. Statistics and local cost parameters are managed by the Global Catalogue Subsystem. After the Global Query Optimizer accepts a user's global query, it analyzes a number of alternative decomposition strategies. During the analysis, the costs for the local queries resulting from a decomposition are estimated. The estimation is done by the Cost Estimator module using the cost parameters and statistic information retrieved from the Global Catalogue. The Cost Estimator may also perform some probing queries[3] to obtain some information and/or sampling data required by the estimation. Based on estimated costs, the Global Query Optimizer selects a good (cheap) exe-

[3]The queries in steps (c) and (f) of the extended Lipton and Naughton's method can be considered as probing queries suggested in [21].

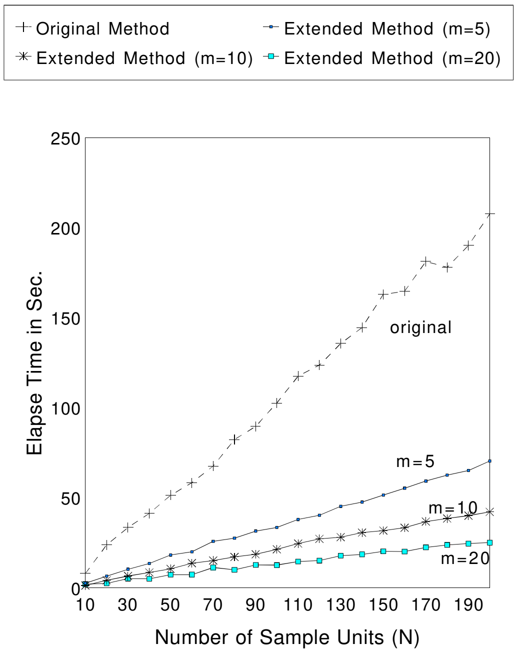

Figure 2 Performance of Original and Extended Lipton's Methods

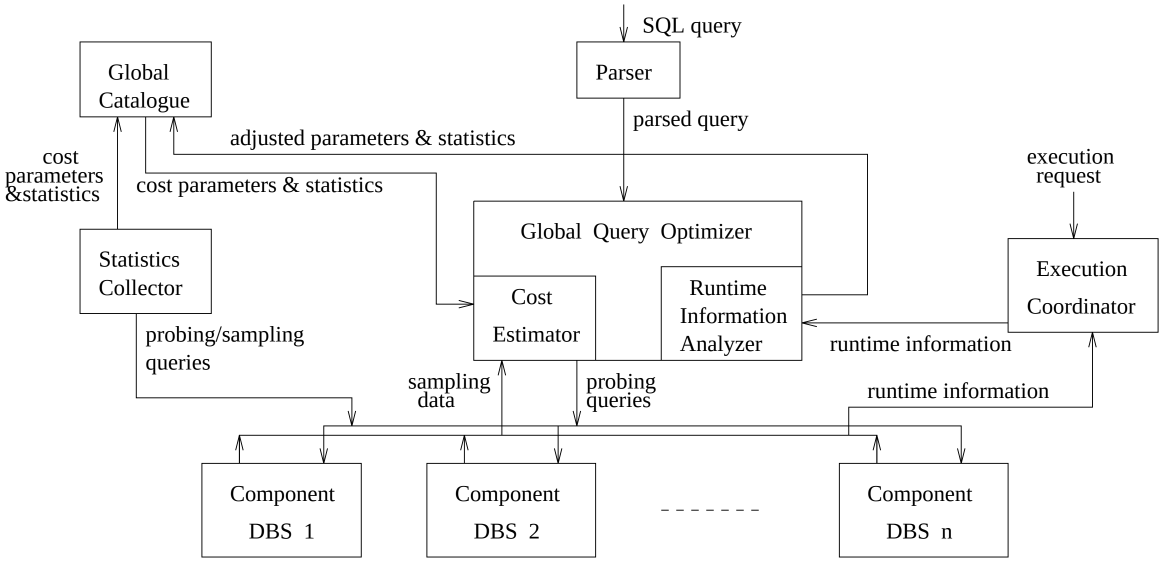

Figure 3 Cost Estimation in CORDS-MDBS

cution plan for the given query. The execution plan may ride some additional side retrievals piggyback on query processing to collect some statistics, as mentioned in Section 3. When the execution plan is executed, some runtime information is collected by the Execution Coordinator and passed to the Runtime Information Analyzer in the Global Query Optimizer. The latter analyzes the runtime information to decide if some cost parameters and statistics need to be updated. Adjusted information, if any, replaces the old information in the Global Catalogue.

6 Conclusion

An execution plan for a global query in an MDBS specifies how to decompose the given query into local (component) queries and how to integrate the results of local queries into the final result for the given query. Different decomposition strategies result in different execution plans. To choose a good (cheap) execution plan for a given global query, the costs of the local queries resulting from a decomposition need to be estimated. How to estimate local costs on autonomous local (component) database systems in an MDBS is a challenging problem because some local information may not be available at the global level.

A query sampling method[22] used to estimate the costs of local queries in our DBMS is reviewed in this paper. The idea of the method is (1) classifying (local) queries, (2) drawing sample queries from each query class, and (3) using the observed costs of the sample queries and the multiple regression method to derive a cost estimation formula for each query class. Experimental results show that this method is quite promising for estimating local costs in an MDBS.

To use the derived formulas to estimate the costs of local queries, the selectivities of the qualifications of the queries are required as inputs. Thus, a method is required to estimate selectivities. There are three types of estimation methods proposed for a (centralized) DBMS in the literature — parametric methods, table-based methods and sampling-based methods. However, none of them can be used in an MDBS to handle all cases efficiently. Difficulties of estimating selectivities in an MDBS are discussed. Based on the discussion, this paper proposes a method that integrates and extends several existing methods so that they can be efficiently used in an MDBS to estimate selectivities.

The integrated method applies a parametric method to estimate selectivities for unary queries and a sampling-based method to estimate selectivities for join queries. The selectivity of a general query can be estimated by using the selectivities of the unary and join queries that comprise the query. The proposed method extends Christodoulakis' method so that the method can handle more types of unary queries, include more data distributions, and be more accurate. The method employs a new piggyback approach to collect and maintain statistic information about local databases so that the cost of maintaining statistics can be reduced. The method extends Lipton and Naughton's adaptive sampling method to estimate selectivities for join queries. Theoretic analysis and simulation results show that the extended Lipton and Naughton's method can be many times faster than the original Lipton and Naughton's method. In addition, accuracy of estimation is also improved.

The integrated method is designed for the CORDS-MDBS that is being developed at University of Waterloo and Queens' University. Implementation considerations are given in this paper as well.

Trademarks

IBM, DB2, IMS and IBM RISC System/6000 are trademarks of International Business Machines Corporation, Armonk, N.Y. Oracle is a trademark of Oracle Corporation. Empress is a trademark of Empress Software Inc.

Acknowledgements

I would like to express my sincere thanks to my supervisor Professor Paul Å. Larson for his valuable suggestions and encouragement. I am grateful to Lauri Brown and Wendy Powley for their help in setting up the simulation environment. I would also like to thank all the members of CORDS-MDBS group for many useful discussions. Finally, I would like to thank all referees for their valuable comments.

References

[1] M. L. Bittinger and B. B. Morrel. *Applied Calculus*. Addison-Wesley, 1984.

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

[3] S. Christodoulakis. Estimating record selectivities. *Information System*, 8(2):105–115, 1983.

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

[5] P. J. Haas and A. N. Swami. Sequential sampling procedures for query size estimation. In *Proceedings of VLDB*, pages 341–50, 1992.

[6] W. C. Hou et al. Statistical estimators for relational algebra expressions. In *Proceedings of PODS*, pages 276–87, 1988.

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

[8] Y. Ling and W. Sun. A supplement to sampling-based methods for query size estimation in a database system. *SIGMOD Record*, 21(4):12–15, Dec. 1992.

[9] R. Lipton and J. Naughton. Estimating the size of generalized transitive closures. In *15th International Conf. on VLDB*, pages 165–72, 1989.

[10] R. Lipton and J. Naughton. Query size estimation by adaptive sampling. In *Proceedings of ACM PODS*, pages 40–46, 1990.

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

[12] H. Lu, B.-C. Ooi, and C.-H. Goh. On global multidatabase query optimization. *SIGMOD Record*, 21(4):6–11, Dec. 1992.

[13] 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.

[14] A. Makinouchi et al. The optimization strategy for query evaluation in RDB/V1. In *Proceedings of the 7th International Conf. on VLDB*, pages 518–29, 1981.

[15] M. Muralikrishna and D. J. DeWitt. Equi-Depth histograms for estimating selectivity factors for multi-Dimensional queries. In *Proceedings of SIGMOD*, pages 28–36, 1988.

[16] F. Olken and D. Rotem. Random sampling from b^+trees. In *Proceedings of the 15th International Conf. on VLDB*, pages 269–78, 1989.

[17] P. G. Selinger and M. Adiba. Access path selection in distributed data base management systems. In *Proceedings of the International Conference on Data Bases*, pages 204–15, Aberdeen, Scotland, 1980.

[18] P. G. Selinger et al. Access path selection in relational database management systems. In *Proceedings of ACM SIGMOD*, pages 23–34, 1979.

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

[20] C. T. Yu, L. Lilien, K. C. Guh, M. Templeton, D. Brill, and A. Chen. Adaptive techniques for distributed query optimization. In *IEEE 1986 International Conference on Data Engineering*, pages 86–93, Los Angeles, USA, 1986.

[21] Qiang Zhu. Query optimization in multidatabase systems. In *Proceedings of CASCON'92 vol.II*, pages 111–27, Toronto, Canada, Nov. 1992.

[22] Qiang Zhu and P.-Å. Larson. A query sampling method of estimating local cost parameters in a multidatabase system. Technical Report CS93-42, University of Waterloo, Canada, 1993.

[23] Qiang Zhu and P.-Å. Larson. Establishing a fuzzy cost model for query optimization in a multidatabase system. To Appear in the Proceedings of the 27th Hawaii International Conference on System Sciences, 1994.

About the Author

Qiang Zhu is a Ph.D. candidate in the Department of Computer Science, University of Waterloo, under the supervision of Professor P.-Å. Larson. He holds an M.Eng. and an M.Sc. in Computer Science and Applied Mathematics, respectively. He was a principal developer of a relational database management system. He is currently involved in the MDBS research project within the CORDS project. His current research interests include distributed database systems and query optimization. His Internet address is *qzhu@maytag.uwaterloo.ca*.