QUERY PROCESSING STRATEGY IN A DISTRIBUTED DATA BASE

P. Bodorik
School of Computer Science
Technical University of Nova Scotia
P. O. Box 1000
Halifax, Nova Scotia B3J 2X4

ABSTRACT

This paper addresses several aspects of Query Processing Strategy (QPS) formulation in a Distributed Data Base (DDB). Assumptions on the DDB environment for further discussion about the QPS problem are stated. "Implied" 2-V terms are defined, which, if included in a query and cascaded joins are available for processing of 2-V terms, may lead to a lower cost strategy. Formulation of QPS is shown to be a problem which is at least as difficult as an NP-hard problem by showing that a subset of the QPS formulation problem is transformable to the travelling salesperson problem. A brief description of a model of a relational DDB is given which can be used for evaluation of QPS formatters. An algorithm which uses the model and develops a space of strategies available for query execution is outlined.

UNE STRATEGIE DE TRAITEMENT DES RECHERCHES DANS UNE BASE DE DONNEES REPARTIES

RESUME

L'auteur aborde plusieurs aspects de la formulation de la Stratégie du Traitement des Recherches (STR) dans une Base de Données Réparties (BDR). Il propose quelques assomptions sur l'environnement des BDR pour de plus amples discussions sur la question des STR. Les termes 2-V "implicites" sont définis, qui, si on les inclut dans une recherche et dans des groupages en cascade, deviennent disponibles pour le traitement des termes 2-V et peuvent résulter en une stratégie de recherche moins dispendieuse. L'auteur prouve que la formulation des STR est un problème au moins aussi épineux qu'un problème de type-NP en démontrant qu'un sous-ensemble du problème de la formulation des STR peut être converti en un problème de "commis-voyageur". On présente une brève description d'une BDR relationnelle et l'on propose son utilisation pour évaluer les programmes de mise en forme de STR. Enfin, on propose un algorithme qui utilise le modèle et développe un environnement de stratégies pour l'exécution de la recherche.

QUERY PROCESSING STRATEGY IN A DISTRIBUTED DATA BASE

P. Bodorik
School of Computer Science
Technical University of Nova Scotia
P. O. Box 1000
Halifax, Nova Scotia B3J 2X4

INTRODUCTION

An integrated Distributed Data Base (DDB) can be briefly defined as a Data Base (DB) in which data elements are distributed over geographically separate network nodes. Users are provided with a unified Data Model (relational, hierarchical, network) and a Data Base Management System (DBMS) which offers a query language, data definition and manipulation languages, data access paths, concurrency control mechanism, etc. One of the existing "operations and control" (Toth (1980)) problems in a DDB, the Query Processing Strategy (QPS) problem, is addressed herein for an integrated DDB based on a relational data model. It is assumed that a query language is based on a relational calculus (Cod (1971)) in which queries are further restricted to a conjunctive normal form which was found to be particularly suitable for many existing query languages (SQUARE, QBE, QUEL, SEQUEL) (Ullman (1980)). Each term is assumed to have at most two relational variables.

Informally, a query in a conjunctive normal form is a sequence of formulas (terms) concatenated by AND Boolean operators. For example, if R_i are relations, i = 1, 2, 3, ..., a, b, c, ... are attributes, k_1, k_2, k_3, ..., are constants, and $*_1$, $*_2$, ..., are arithmetic relational operators, then a conjunctive normal form query is: $(R_1 *_1 R_2.b *_2 R_3.c *_3 k_1 \ldots) \wedge \ldots \wedge (R_1.d *_m \ldots)$. Note that relational variables in the above terms do not have to be distinct. Moreover, each term having more than one arithmetic relational operator can be transformed into an equivalent query which is a conjunction of terms in which each term has exactly one arithmetic relational operator (Wong (1976)).

A query is processed by execution of generalized joins which process 2-V terms, restrictions which process one variable terms, and projections for removal of unnecessary attributes. Sequencing of these operations, decisions about locations of their execution and transfer of data is according to the Query Processing Strategy (QPS) formulated by a QPS formatter. Different QPS will result in a different cost (time delay or monetary) of execution and the optimization of a processing strategy for a given query entails formulation of a strategy with a minimum cost.

A static QPS assumes a priori knowledge of execution costs of different sequences of relational operators. A priori cost of processing and data transfer can be determined with acceptable accuracy given that the size of data elements is known. The critical, and a very difficult parameter to estimate, is the size of temporary relations (partial results) that are formed during query execution (Delenobel (1981), Toth (1980), Epstein (1980)).

Since a static strategy is based on a priori size estimates of partial results, some authors of formatters of a static QPS propose some form of a "threshold" mechanism for its execution. Should there be a high discrepancy between a priori size estimates and actual sizes of partial results during the processing of a query, some special action is taken (Mahmoud (1979)). Very little attention has been given to this subject, even though it is an integral part of a static QPS.

A purely dynamic QPS does not use a priori cost estimation at all, the strategy is dynamically formulated in the query's execution phase. Redundant processing, communication, or both are allowed so that an optimal QPS minimizing a given cost objective is obtained. If the cost objective is a total query response time, both redundant processing and communication is introduced.

A synthetic approach is suggested by some authors (Mahmoud (1979), Toan (1981)). In static strategies based on a priori cost estimates, a threshold mechanism can be used which is a dynamic element. Similarly, dynamic approaches use some form of a priori estimation, an element of a static approach, as well. Note that most researchers concentrated their efforts on a static QPS.

This paper addresses several aspects of QPS formulation. A review of the research addressing the problem of QPS in a DDB is made and assumptions on the DDB environment for further discussion of the QPS problem are stated. "Implied" 2-V terms are defined, which, if included in a query and cascaded joins are available for processing of 2-V terms, may lead to a lower cost strategy. Formulation of a QPS is shown to be a problem which is at least as difficult as an NP-hard problem by showing that a subset of a QPS problem is transformable to the travelling salesperson problem. A brief description of a model of a relational DDB is given which can be used for evaluation of QPS formatters. An algorithm which uses the model and developes a space of strategies available for query execution is outlined. Since the algorithm develops a space of available strategies, it can also be used for QPS formulation.

LITERATURE REVIEW

One of the earliest works on QPS optimization was done by Wong and Yossefi (Wong (1978)). They decompose a query into a sequence of one-variable and two-variable queries and show that most of the time it is profitable to perform one-variable queries as soon as possible. A two-variable query is reduced into a set of one-variable queries by tuple substitution. In other words, one tuple is taken from one relation at a time and applied to the other relation, thus creating a one-variable query. A two-variable query is in this way transformed by a tuple substitution into a sequence of one-variable queries.

In a network environment, a tuple substitution method to process two-variable terms may lead to unacceptable delays. To alleviate this problem, Epstein, Stonebraker, and Wong (Epstein (1978)) abandon tuple substitution and instead they suggest that all local processing be performed first, followed by a feasible strategy which moves all results of one-variable processing to the destination node. This feasible strategy is improved upon by a "greedy" algorithm. The cost objective considered is either communication cost or CPU processing time. This approach was adopted for SDD-1.

In Toth (1978) rules for decomposition of queries are presented. Optimal QPS is devised for a special class of sub-queries which can be processed optimally. These sub-queries are processed first by an optimal strategy, thus reducing the size of intermediate results that are then processed by a sub-optimal strategy.

Cellery and Meyer (Cellery (1980)) consider query processing in a multi-query environment. Their strategy's objective is to minimize the mean response time of queries. Optimization is performed by scheduling of data transfer and a work-load on local processors. A heuristic is proposed which performs optimization of response time of queries at the expense of redundant processing and transmission.

A QPS for a DDB which uses a "virtual ring" method for concurrency control is discussed in Toan (1979). The optimization method uses dynamic query decomposition facility with threshold policies while using the control token circulation on the virtual ring of processors to solve conflicts of access between queries and updates.

In certain situations, it may be advantageous to precede a join by a semi-join (Smith (1975)), a tactic borrowed from OPS optimization in a central DB. The method suggested by Henever and Yao (Henever (1979)) is completely based on the use of semi-joins. They suggest that all joins should first be performed on appropriate projected "joining" attributes (i.e., that semi-joins should be executed first), while whole relations are moved to the destination node concurrently. Results of semi-joins are also moved to the destination node where they are processed with the original relations to obtain the final result.

A combined static and dynamic decomposition of user queries for MICROBE DDB is suggested by Toan (Toan (1981)). It assumes a decentralized query management and a broadcast network.

The problems of RA and QPS were attacked simultaneously in Paik (1979). Large amounts of redundant data are avoided in order to minimize update synchronization of multicopy data and storage requirements. Assuming a minimum amount of redundant data optimally allocated over the network a several-fold static optimization is performed on queries. Some dynamic optimization elements in query execution phase are introduced as well.

ASSUMPTIONS

Optimization of a QPS in a central DB has led to a generally accepted set of rules (Ullman (1980)) which apply to a DDB as well, and which are assumed herein. These rules are:

(i) One variable operators (restrictions and projections) should be performed as early as possible as they reduce the size of relations which are operands of more costly 2-V operators (terms).

(ii) Where applicable, one variable operators should be performed (grouped/cascaded) together. Obviously, if there are several restrictions to be applied on one relation, these should be applied simultaneously in one pass through the relation. For example, $(R.a \geq 2)$ ^ $(R.b \leq 4)$ specifies two restrictions on the same relation R. Clearly, both of these restrictions can be applied simultaneously on one pass through the relation with only a negligible additional cost. Simultaneous execution of such one variable terms is called a cascaded one variable term.

(iii) Similarly, if there are several 2-V terms specified for the same two relations, these should be applied simultaneously as well. Such terms are called cascaded 2-V terms or a cascaded join.

Additional well accepted tactic assumed herein involves execution of all local joins (Mahmoud (1979), Epstein (1978)). Those joins that can be executed without any transfer of data are executed first.

In a network environment, in which the load on processors is not balanced, it may be possible to utilize available processing power of a lightly loaded system by migration of work load to it. Research on QPS optimization so far allowed only a limited form of a choice of the location of operator execution. One variable operator is never migrated to a different network node for its execution. As it reduces the size of its operand, it is executed at the network node where the operand resides. For a two variable operator, its execution is allowed at either one of the two network nodes, "homes" of the two operands. However, movement of both operands of a 2-V operator into some third, lightly loaded node, is not considered.

As was stated earlier, queries are assumed to be in a conjunctive normal form with each term having at most two variable relational operators. In addition, queries are assumed to be defined on a global view of a data base (Mahmoud (1979)), which greatly simplifies the formulation of a QPS as optimization of queries containing union operators is avoided.

Creation and transfer of redundant copies of relations is not considered and therefore the following discussion applies to a DB distributed in a point-to-point network.

IMPLIED 2-V TERMS

A conjunctive normal form query can be represented by a query graph as defined in Ullman (1980) and shown in Figure 1(a). Nodes of the graph represent relations, while edges represent 2-V qualification terms. Since one-variable terms are assumed to be executed first, which is a well established tactic, they are not included in the query graph. A QPS formatter considers them in the cost calculations, but it primarily deals with the sequencing of 2-V terms (generalized joins) and movement of data.

Consider a sequence of joins which produce the answer to a simple query of Figure 1(a). After the first join is executed, for example ($R_1.a > R_2.b$) giving a partial result, relation R_{12}, the remainder of the query can be represented by the graph shown in Figure 1(b). Note that a query graph may be multi-edged either due to the original query having more than one join defined on two relations, or to the processing of 2-V terms. Multi-edges are represented by one edge labelled by more than one 2-V operator as shown in Figure 1(d) -- they represent cascaded joins which, as previously stated, can be executed for a cost of one join.

Consider query $Q = (R_1.a > R_2.b) \wedge (R_2.b > R_3.c)$ shown in Figure 1(a). Because of the transitivity of the "greater than" operator, an additional 2-V term, although not specified by the query, will be satisfied, i.e., ($R_1.a > R_3.c$). The question arises whether this implied 2-V term should be included in the formulation of a processing strategy by the QPS formatter. Immediately it can be observed that if the implied 2-V term is included an additional join must be processed which, of course, seems to be a bad tactic. However, this is not the case if cascaded processing of joins is available. The graph of query Q which is modified to include the implied 2-V is shown in Figure 1(c). After execution of the first join, for example ($R_1.a > R_3.c$), the graph for the remainder of the query is shown in Figure 1(d). There are two joins remaining, however, they can be cascaded because they both apply to the same relations. Effectively, only one cascaded join remains to be executed.

Processing of a query can be divided into two phases, (i) QPS formulation, and (ii) QPS execution. It will be shown that inclusion of implied 2-V terms will increase the number of available strategies to be considered by the QPS formatter, some of these may have a lower cost than those available without their inclusion. Therefore, implied 2-V terms may lead to a QPS with a decreased query execution cost. However,

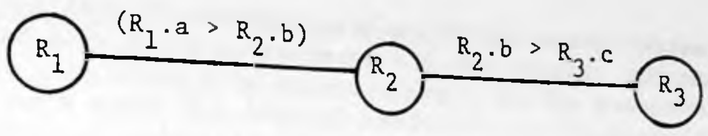


Figure 1(a) Graph of a Simple Query

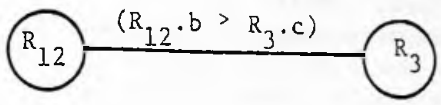


Figure 1(b) Modified Query Graph to Reflect Processed 2-V term $(R_1.a > R_2.b)$

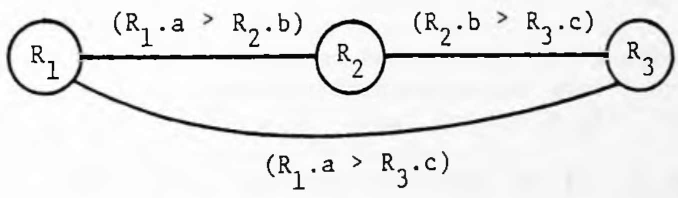


Figure 1(c) Graph of a Query Shown in Figure 1(a) Modified to Include Implies 2-V Term $(R_1.a > R_3.c)$

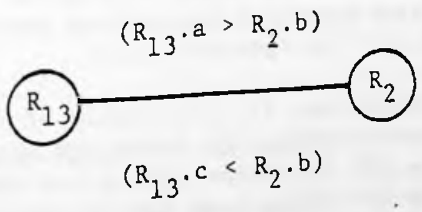


Figure 1(d) Modified Query to Reflect Processed 2-V Term $(R_1.a > R_3.c)$

the QPS formatter must consider a larger number of available strategies and therefore the cost of the QPS formulation phase will be increased. Typically, the cost of QPS formulation is much lower than the cost of query execution phase and therefore implied 2-V terms should be included in the formulation of a QPS. It will also be shown, that the complexity of the algorithm for their inclusion is of $O(n^2)$, where n is the number of joining attributes. In other words, it is a simple algorithm in comparison to the algorithm formulating a QPS.

<u>Definition 1</u>: Let * to denote one of the following arithmetic operators {=, >, <, >=, <=}. Let relations R_1, R_2, R_3 referenced by the query have, among others, joining attributes a, b, and c, respectively, such that a conjunctive normal form query has the following 2-V terms: $(R_1.a * R_2.b)$ ^ $(R_2.b * R_3.c)$. Then $(R_1.a * R_3.c)$ is called an implied 2-V term. Modify the query to include this 2-V term if it is not contained in the query already and apply the definition repetitively until no new implied 2-V terms are added to the query. Note, $(R_1.a < R_2.b)$ is equivalent to $(R_2.b > R_1.a)$.

<u>Theorem 1</u>: Query Q', which is the original query Q modified to contain implied 2-V terms, is equivalent to the original query Q (i.e., the answer to the query Q is the same as the answer to Q').

<u>Proof</u>: The proof of this theorem can be found in Bodorik (1983). The theorem is intuitively obvious and is proven by showing that $Q \subseteq Q'$ and $Q' \supseteq Q$.

Definition 1 can be extended in an obvious way to include pairs of operators that have the transitivity property. For example, $(R_1.a \geq R_2.b)$ ^ $(R_2.b > R_3.c)$ imply $(R_1.a \geq R_3.c)$.

<u>Theorem 2</u>: If the answer to the conjunctive normal form query is obtained by execution of successive joins, the inclusion of implied 2-V terms does not increase the number of joins to be executed if cascaded joins are available and a cascaded join is considered to be one join.

<u>Proof</u>: Formal proof can be found in Bodorik (1983). It is based on the fact that processing of a modified query containing implied 2-V terms will always result in cascaded joins.

<u>Theorem 3</u>: Assuming that (i) processing of cascaded 2-V terms is available, (ii) perfect information about the environment is available, and (iii) the QPS formatter determines optimal QPS, then the inclusion of implied 2-V terms does not increase the cost of the query execution phase, but it does increase the cost of QPS formulation phase.

<u>Proof</u>: If implied 2-V terms are included in a given query, the QPS formatter must consider increased number of available strategies and therefore the cost of QPS formulation is increased. What remains

to be shown is that the strategy formulated does not have a higher execution cost than without the implied 2-V terms. Assume that the QPS formatter formulates optimal strategy S with a cost of query execution c(S) for a query Q without implied 2-V terms. If the formatter creates optimal strategy S' for the same query with implied 2-V terms whose cost is c(S'), then it must be that $c(S') \leq c(S)$. Either the increased space of strategies created by inclusion of 2-V terms does contain optimal strategy S' whose cost c(S') is smaller than c(S), or it does not, in which case the optimal strategy for the modified query is S with a cost c(S). QED.

Algorithm to Determine Implied 2-V Terms for a Given Query

Input: Query in a conjunctive normal form.

Output: Query in a conjunctive normal form modified to include all implied 2-V terms.

Method

(1) Create a square matrix whose indices in each dimension correspond to all attributes of a cartesian product of referenced relations. Entries to the matrix are arithmetic relational operators present in 2-V terms in the original query. Entry in the row $R_1.a$ and a column $R_2.b$ is operator "*" if the given query has a 2-V term $(R_1.a * R_2.b)$, it is nill (empty) otherwise. Note that a 2-V term $(R_1.a > R_2.b)$ is equivalent to $(R_2.b < R_1.a)$ due to commutativity and therefore there will be two entries in the matrix for every 2-V term of the query.

(2) For each row of the matrix do step (3).

(3) For each row corresponding to attribute $R_1.a$ examine every non-empty entry. Assume that the entry is the operator * and that it is in a column corresponding to the attribute $R_2.b$. The entry identifies 2-V term $(R_1.a * R_2.b)$. Scan the row corresponding to $R_2.b$ for every entry * (i.e. the same operator) because it identifies implied 2-V term $(R_1.a * R_3.c)$. Make two entries in the matrix corresponding to this term.

(4) Repeat steps (2) to (3) until no new entries are made in step (3).

Correctness: Correctness of the algorithm is shown in Bodorik (1983).

The algorithm can be improved by use of a data structure which will eliminate a search through empty elements of the matrix. In addition, after the first iteration through steps (2) to (3), the "old" elements of the matrix do not have to be re-examined as they will not contribute to the addition of new implied 2-V terms. Therefore, on a new iteration through steps (2) and (3), only those entries of the matrix have to be examined that were added on the previous pass.

The improved algorithm and its proof of correctness can be found in Bodorik (1983). Let n be the number of joining attributes in the resulting query containing implied 2-V terms. For each of the 2-V terms, maximum n terms are searched in the improved algorithm and hence, the algorithm is of $O(n^2)$.

Discussion

Considering that the algorithm is fairly simple, it is a good candidate for implementation. It was shown that inclusion of implied 2-V terms increases the cost of QPS formulation but that it may lead to a decrease in the cost of query execution. If the cost of query execution is much larger in comparison to QPS formulation cost then the implied 2-V terms should be included in the query submitted to the QPS formatter.

QPS FORMULATION - PROBLEM AS DIFFICULT AS AN NP-HARD PROBLEM

The problem of finding the optimal QPS under assumptions adopted at the beginning of this paper (QPS problem) is at least as difficult as an NP-hard problem. This is proven by showing that a subset of the QPS solution, QPS", is transformable to the problem TS' which, in turn, is transformable to the "Travelling Salesperson Problem" for which it is known that it is NP-hard (Reingold (1977), Horowitz (1978)).

In the Travelling Salesperson (TS) problem, the salesperson must visit n cities. Each city, except the city of departure, is visited exactly once, and then the salesperson must return to the city of departure. The salesperson is required to minimize the total cost of the trip given travel costs between any two cities.

TS' Problem: Original TS problem calls for a tour to start and end in the same, given city. Modify this requirement to allow the salesperson's tour to start in any city and also to finish in any city, i.e., the cities of departure and tour's termination are not necessarily the same.

Lemma 1: TS problem is transformable to the TS' problem.

Proof. The proof can be found in Bodorik (1983).

QPS' Problem: Let (R_1, R_2) represent a 2-V term $(R_i.a * R_j.b)$, where * denotes any arithmetic relational operator, with the exception of $\neq$. Consider a query whose graph is shown in Figure 2(a). One of the strategies may be to process a 2-V term (R_1, R_2) resulting in relation R_{12}, followed by processing of (R_{12}, R_3) resulting in R_{123}. As well, relations R_4 and R_5 may be joined by processing of a 2-V term (R_4, R_5), resulting in relation R_{45}. The modified query graph to reflect this partial result is shown in Figure 2(b). Next, relations R_{123} and R_{45} may be joined by processing of a 2-V term (R_{123}, R_{45}). In other words, several joins may be performed to form partial results R_a, R_b by processing one or more 2-V terms to create each. Relations R_a and R_b may then be joined by processing the 2-V term (R_a, R_b). This is not allowed in the modified QPS problem, QPS'. Only those 2-V terms (R_a, R_b) are allowed for joining, where at least one of the operands, R_a or R_b (or both) are original relations referenced by the query (i.e. are not partial results of processed joins). Clearly, the QPS' problem is a subset of the QPS problem.

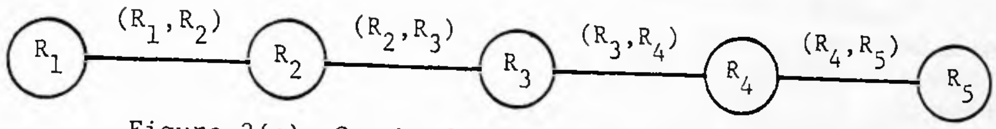


Figure 2(a) Graph of a Query with Four 2-V Terms

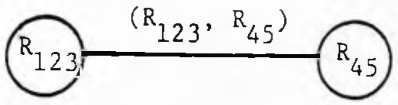


Figure 2(b) Graph of a Query Reflecting Processed 2-V Terms

QPS" Problem: Make an additional simplification to the QPS' problem which involves cost calculations. Assume that all information processors have the same capacity and loads. Let the cost of CPU processing of a 2-V term (R_i, R_j) be the same whether the processing occurs in the node where R_i or R_j is located. Construct a cost matrix C" such that rows and columns correspond to joining attributes of a cartesian product $R = R_1 \times R_2 \times \ldots \times R_n$, where $R_1, R_2, \ldots, R_n$ are relations referenced by the query. Calculate the cost of processing of each query's 2-V term and enter it to the cost matrix C" (i.e., the cost of data transfer is ignored altogether). If, in the original query, there is no 2-V term (R_i, R_j) specified, enter a cost $c''_{i,j} = \infty$. Whenever a join (R_i, R_j) is considered in formulation of a QPS, where R_i is the partial result of joining relations $R_a, R_b, \ldots, R_g, R_h$, and where the last join for this partial result was ($R_{ab\ldots g}$, R_h), than the cost of joining (R_i, R_j) is taken to be $c''_{h,j}$. Recall that due to restrictions introduced in QPS', R_j must be a relation that was not yet processed. Clearly, QPS" is a subset of QPS' which is a subset of QPS (QPS" $\subsetneq$ QPS' $\subsetneq$ QPS).

Lemma 2: QPS" problem is transformable to TP'.

Proof: The proof can be found in Bodorik (1983). It is based on the following observation. The TP' problem is defined as a problem of finding the least expensive tour to be made by a travelling salesperson. The salesperson has a choice (without cost) of starting in any city and finishing in any city, as long as no city is visited twice. The cost of travelling between cities is given by a square matrix C with n rows and n columns. Each element $c_{i,j} > 0$ represents a cost of travelling between cities i and j. Some of the costs may be infinite if there is no transportation available between the two cities. Diagonal elements $c_{i,i}$ are also set to ∞.

Consider the QPS" problem. It also has a cost matrix C" whose rows and columns correspond to joining attributes of the original query and the entries correspond to costs of processing joins (R_i, R_j), where R_i and R_j are relations referenced by the query. Finding the least expensive QPS strategy in the QPS" problem involves finding the minimum cost sequence of processing of 2-V terms under restrictions of QPS". Any 2-V term can be processed first, any 2-V term can be processed last, which corresponds directly to TS'. Suppose a strategy starts with a 2-V term (R_i, R_j). This corresponds to the salesperson starting in the city i and moving to city j with a cost of $c_{i,j}$. The cost of the corresponding 2-V term in a QPS" problem is $c''_{i,j}$. The next 2-V term considered must be (R_{ij}, R_k) such that the term (R_j, R_k) exists in the original query, i.e. $c''_{j,k} \neq \infty$. The total cost of processing is the cost of (R_i, R_j) + the cost of (R_j, R_k), i.e. $c''_{i,j} + c''_{j,k}$. In the TP' problem, this represents the choice of the salesperson moving from city j to city k with the total cost of the tour so far being $c_{i,j} + c_{j,k}$, and so on. The restriction that the salesperson does not visit any city twice corresponds directly to the restriction in the QPS" that no join be processed twice.

Theorem 4: The problem of finding the optimal QPS is at least as difficult as an NP-hard problem.

Proof: It follows from Lemma 1 that the TS problem is transformable to the TS' problem. Since the TS problem is NP-hard (Reingold (1977)), it follows that TP' is also NP-hard. By Lemma 2, QPS" is transformable to TP' and therefore QPS" is NP-hard. QPS" is a subset of the QPS' problem which, in turn, is a subset of the QPS problem and since QPS" is NP-hard it follows that the QPS problem is at least as difficult as an NP-hard problem. QED.

Discussion

Since there is no algorithm available for solving problems in NP in polynomial time, this also applies to the problem of QPS formulation. As a result, if queries issued against a DDB can be expected to have a large number of 2-V terms, heuristic approaches to QPS formulation are justified.

RELATIONAL DDB - MODEL FOR QPS

In this section a simple model of relational DDB for QPS formulation and its evaluation is described. It is based on parameters describing the DDB while adopting assumptions on query processing stated earlier. Based on this model, a space of available strategies and their costs can be created for any given query, and it may be used for QPS formulation and evaluation.

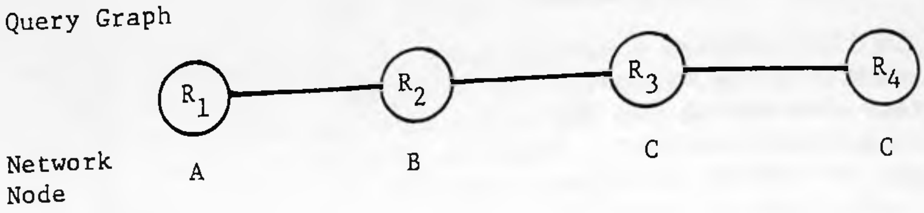
Since the model's parameters identify all information necessary for QPS formulation and its cost estimation, they are exactly the same as parameters that must be made available to a QPS formatter. Specifically, they convey information about:

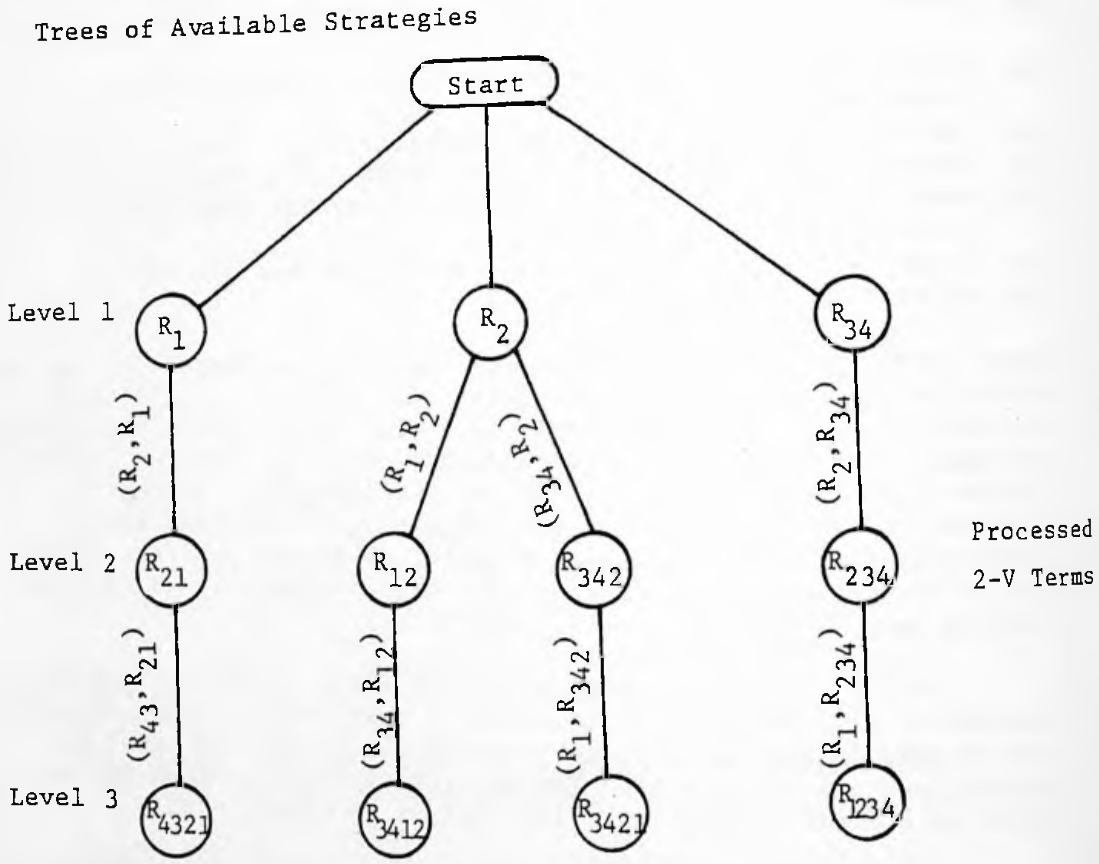
(a) Relations (syntactical details, sizes, availability of access paths).
(b) Information Processors (time delays, monetary costs).
(c) Distribution of Relations on Information Processors.
(d) Network Nodes and Unit Costs (time delay and monetary costs of data transfer).
(e) Distribution of Information Processors on Network Nodes.
(f) Selectivity Function.

Most of the above parameters can be accurately estimated for any DDB. Since the formulation of a QPS considered assumes a single query environment, the parameters expressing time delays for CPU processing and data transfer include average delays due to processing of other queries within the data base. The crucial paramter, and most difficult one to estimate, is the "Selectivity Function" which determines the sizes of partial results, relations that are formed due to processing of joins, restrictions, and projections (Epstein (1980), Toth (1980), Demolombe (1980)).

Once the above information is available, either in the environment of a real DDB or its model, a relatively simple algorithm can be used to develop a space of available strategies under the assumptions stated earlier. The details of the algorithm can be found in Bodorik (1983); only its outline is given here.

The algorithm develops a tree of available strategies consisting of a number of levels corresponding to the number of 2-V terms in the given query. The tree's first level corresponds to relations obtained after processing of all one variable terms and those joins for which both operands (relations) are located in the same network node. The nodes of the tree's second level represent relations that are partial results of execution of the first join (could be a cascaded join) and projections that follow to remove no longer needed joining attributes. Nodes of the third level represent relations that are partial results of processing exactly 2 joins followed by projections to remove joining attributes, and so on. The nodes of the last level represent the answer to the query, each obtained by a different strategy (see Figure 3).





Note, (R_a, R_b) and (R_b, R_a) represent the same 2-V term, (R_a, R_b) implies that the 2-V term is processed in the network node where R_b is located, while (R_b, R_a) implies that it is processed in the node where R_a is located.

Figure 3. Simple Query and Its Tree of Available Strategies

The cost of a strategy can be in any one of the following cost metrics:

(i) cost of total CPU processing (time delay or monetary)
(ii) cost of data transfer (time delay or monetary)
(iii) combined cost of CPU processing and data transfer (time delay or monetary)
(iv) combined monetary and time delay cost

The first three are standard cost metrics used in optimization of a query processing strategy (Mahmoud (1979)), while the fourth one uses a transformation function which may be user-supplied. It is defined as $f(t) = ct$, where t is in seconds and c is a constant of proportionality. It represents the user's cost of waiting and allows optimization of combined monetary and time delay costs.

The cost for a partial result represented by a tree node can be calculated from the costs of partial results of previous levels using parameters expressing costs of data transfer and CPU processing. The nodes of the last level which represent the answer to the query also contain the cost of transfer to the destination node.

Once the space of strategies for a given query is developed, it is a simple task to search the last level for a node with the smallest cost to determine the optimal QPS based on perfect information. As a matter of fact, optimal strategy for each one of the aforementioned cost metric can be found and therefore the space of strategies can be used to study trade-offs existing in a choice of a cost metric. For example, if time delay of query execution is chosen as a cost metric, the monetary cost of the optimal strategy may be high. Using the space of strategies, its monetary cost can be compared with the cost of the optimal strategy based on a monetary metric cost.

The cost of strategies is developed prior to a query's execution. Due to the variation of parameters (costs, delays) in a real environment and imperfection in size estimation of partial results, the real cost of the strategy's execution will differ from the estimated cost obtained from the space of strategies. To investigate the dependence of the query's cost on the dynamic environment, a simple simulation can be used. A parameter of interest (e.g. CPU processing delays) can be varied randomly while holding the remaining parameters constant, and the new cost of the strategy can be calculated for each random variation. In addition, the cost of the optimal strategy for each random variation can be found again to be used for comparison purposes.

SUMMARY

Several aspects of QPS formulation in a relational DDB were discussed. Although inclusion of the query's implied 2-V terms seems to increase the number of joins necessary to process a query, it was shown that this is not the case if cascaded joins are available. Inclusion of 2-V terms leads to an increase in the number of available strategies, some of which may have a lower execution cost, but it also increases the cost of QPS formulation. A simple algorithm for their inclusion in a query was outlined and shown to be of $O(n^2)$, where n is the number of joining attributes. If the cost of the QPS formulation is small in comparison to the query's execution cost, the inclusion of 2-V terms is a good candidate for implementation.

However, the problem of QPS formulation under the assumptions stated was shown to be a problem which is at least as difficult as an NP-hard problem. Implied 2-V terms should therefore be included in a query only if the potential decrease in the query's execution phase outweighs the increase in the query's formulation phase, a problem requiring further study. Due to the complexity of the QPS formulation, use of heuristic QPS formatters is justified for queries with a large number of 2-V terms.

A simple model of a relational DDB was described together with an algorithm which formulates the query's space of available strategies. This algorithm was implemented and is currently used for evaluation of trade-offs existing in a choice query's cost metric and QPS performance. In the near future, it will be modified to remove some of the restrictions inferred by the assumptions stated, specifically, its load balancing ability will be improved and the query will be permitted to contain union operators.

REFERENCES

BODORIK, P., "Query Processing Strategy in a Distributed Data Base", School of Computer Science Report, Technical University of Nova Scotia, Halifax, N. S., 1983.

CELLERY, W. and MEYER, D., "A Multi-Query Approach to Distributed Processing in Relational DBMS", Distributed Data Bases, North Holland Publishing Company, 1980, pp. 99-119.

CODD, E. F., "A Database Sublanguage Founded on the Relational Calculus", Proc. ACM SIGFIDET Workshop on Data Description, Access and Control, New York, 1971.

DEMOLOMBE, R., "Estimation of the Number of Tuples Satisfying a Query Expressed in a Predicate Calculus Language", Sixth Conference on Very Large Databases, Montreal, October 1980, pp. 55-63.

EPSTEIN, R. et al, "Distributed Query Processing in a Relational Data Base System", ACM-SIGMOD, Proc. International Conference on Management of Data, Austin, Texas, 1978, pp. 169-180.

EPSTEIN, R. and STONEBRAKER, M., "Analysis of Distributed Data Base Processing Strategies", Sixth International Conference on Very Large Data Bases, Montreal, Canada, 1980, pp. 92-101.

HENEVER, A. R. and YAO, S. B., "Query Processing in Distributed Data Base Systems", IEEE TSE, Vol. SE-5, No. 3, May 1979, pp. 177-187.

HOROWITZ, E. and SAHNI, S., "Fundamentals of Computer Algorithms", Computer Science Press, 1978.

MAHMOUD, S. A., RIORDON, J. S. and TOTH, K. C., "Distributed Database Partitioning and Query Processing", IFIP-TC-2, Venice, Italy, 1979, pp. 32-51.

NGUYEN GIA TOAN, "A Unified Method for Query Decomposition and Shared Information Updating in Distributed Systems", First Int. Conf. on Distr. Comp. Systems, Huntsville, October 1979, pp. 679-685.

NGUYEN GIA TOAN, "Distributed Query Management for a Local Network", Proc. 2nd International Conference on Distributed Computing Systems, Paris, France, April 1981, pp. 188-196.

PAIK IN-SUP and DELOBEL, C., "A Strategy for Optimization of Distributed Query Processing", Proc. First International Conference on Distributed Computing Systems, Huntsville, Alabama, October 1979, pp. 686-698.

REINGOLD, M. E., NIEVERGELT, J. and DEO, N., "Combinatorial Algorithms", Prentice-Hall, 1977.

SMITH, J. M. and CHANG, P. Y., "Optimizing the Performance of a Relational Algebra Database Interface", CACM, Vol. 18, No. 10, October 1975, pp. 568-579.

TOTH, K. C., "Distributed Database Architecture and Query Processing", Ph.D. Dissertation, Carleton University, Ottawa, Ontario, 1980.

ULLMAN, J. D., "Principles of Database Systems", Computer Science Press, 1980.