COST FUNCTIONS FOR QUERY PROCESSING IN A DISTRIBUTED DATA BASE

P. Bodorik
Assistant Professor
School of Computer Science
Technical University of Nova Scotia
Halifax, Nova Scotia B3J 2X4

ABSTRACT

Various cost functions can be used in the formulation of a QPS in a DDB. The choice of a cost function depends on the application and the environment in which the DDB exists. An algorithm is described which creates a space of available strategies for processing of any given query. As the algorithm finds the cost of each strategy utilizing various measures/cost functions, it can be used as a tool in comparision and a choice of cost functions for any application of a DDB.

Les composantes du coût des recherches sur des bases de données réparties.

Différentes fonctions du coût peuvent être utilisées dans l'énoncé du QPS dans une DDB. Le choix d'une fonction de coût dépend de l'application et de l'environnement dans lequel la DDB existe. Un algorithme qui crée un espace de stratégies disponibles pour le traitement de n'importe quelle demande donnée est décrit. De même que l'algorithme trouve le coût de chaque stratégie en utilisant différentes fonctions mesures/coût, il peut aussi être utilisé comme outil dans la comparaison et le choix des fonctions de coût pour n'importe quelles utilisations de la DDB.

COST FUNCTIONS FOR QUERY PROCESSING IN A DDB

1. INTRODUCTION

Technological advances in computer networking stimulated a great interest in distributed processing for the purpose of sharing resources and data. Arguments of both a quantitative and qualitative nature were put forward in support of, and in opposition to, distribution; the problem became known as "Centralization vs. Decentralization" ((Reynolds, 1971), (Rockart, 1976)). A great interest has also been shown in the application of the concept of a Data Base Management System (DBMS) in a distributed environment supported by a network of geographically separate information processors. Users are provided with a unified data model, together with data description and manipulation languages, while the fact that data itself may is geographically distributed and supported by a network of cooperating information processors is hidden from users.

Advantages that can be gained by the use of a Distributed Data Base (DDB) in comparision to a centralized Data Base (DB) are ((Rothnie, 1977), (Toth, 1980)):

increased availablility;
improved response time;
modularity and upward scaling;
cost reduction.

A user specified query issued against a relational DDB is initially transformed either by a query processor or a pre-compiler, into relational algebra or relational calculus. In either case, the query is answered by execution of relational operators (selection, projection, join, union, etc.) on relations specified by the query. Since the ordering of operator execution is not fixed and relations are distributed over the network of information processors, before the query execution may begin the Query Processing Strategy (QPS) formulator must determine the sequence in which operators are executed, the location of their execution, and, of course, the movement of the data/relations among processors. Formulation of the QPS comprises the query "strategy formulation" phase.

Once the QPS is determined, instructions are sent to local DBMSs for operator exection and transfer of data. The cooperation of local DBMSs by information exchange over the network, is necessary to process the query. The strategy's execution comprises the "query execution" phase.

LITERATURE REVIEW OF QPS IN A DDB

Optimization of a QPS in a central DB ((Smith, 1976), (Yao, 1978, 1979) to list a few) has led to a generally accepted set of rules (Ullman, 1982) which apply to a DDB as well. The 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 two-variable operators.

(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, the query (R.a > 2) ^ (R.b < 4), where "^" is the "AND" boolean operator, 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 an additional negligible cost of a comparision per tuple. Simultaneaous execution of such one-variable terms is called a cascaded one-variable term.

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

One of the earliest works on QPS optimization was done by Wong and Yossefi (1976). 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 a 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. A modified version of this approach was adopted for a distributed version of the INGRES relational DB (Stonebraker, 1976) and was later evaluated on a test bed of queries (Youssefi, 1979).

In a network environment, the tuple substitution method to process two-variable terms may lead to unacceptable delays. To alleviate this problem, Epstein, Stonebraker, and Wong (1978) abandon tuple substitution and 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 in terms of volume transmitted or CPU processing time which is approximated by the maximum CPU time delay for all CPUs involved in processing of the query.

The aforementioned algorithm was later evaluated by an experiment (Epstein, 1980) in which several key issues were investigated. The cost funtion used was the number of bytes transferred during query processing. Two conclusions were offered: (i) good size estimates are crucial, and (ii) a limited search through a space of query processing strategies performs poorly, suggesting that an exhaustive search for the optimal strategy is desirable.

Cellery and Meyer (1980) consider query processing in a multi-query environment. Their strategy's objective was minimizing the mean response time of queries, the only such attempt that could be found. Optimization is performed by the scheduling of data transfer and a local work-load on processors. A heuristic is proposed which optimizes queries' response time at the expense of redundant processing and communication.

A QPS for a DDB which uses a "virtual ring" method for concurrency control is discussed by Toan (1979). The proposed optimization method is based on a four-fold static optimization combined 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 (Bernstein, 1979), a tactic borrowed from QPS optimization in a central DB (Smith, 1976). The method suggested by Henever and Yao (1979) is completely based on the use of semi-joins. They suggest that all joins should first be performed on appropriate projected "joining" attributes (ie., that semi-joins should be executed first), while whole relations are moved to the destination node concurrently. The results of semi-joins are also moved to the destination node where they are processed with the original relations to obtain the final result. They propose an algorithm which finds the optimal strategy for processing semi-joins for a restrictive set of "simple" queries. The algorithm is then extended to a heuristic which finds a sub-optimal strategy for processing of general semi-joins. The objective of the strategy is to minimize the total size of partial results, thus minimize the amount of transferred data.

Black and Luk (1982) propose a heuristic for generating a strategy for semi-joins. Its property is that for moderate size queries an exhaustive search may be performed.

Toth (1978, 1982) presents rules for the decomposition of queries. An optimal QPS is devised for a special class of sub-queries, Class A queries, which can be processed optimally. A Class A query is actually a superset of the simple query defined by Henever and Yao above. These sub-queries are processed first by an optimal strategy, thus reducing the size of intermediate results which are then processed by a

sub-optimal strategy. The objective of the strategy is the minimization of the total size of data transferred under a maximum time constraint.

A combined static and dynamic decomposition of user queries for the MICROBE DDB is suggested by Toan (1981). Semi-joins are considered for query processing and a threshold mechanism with dynamic threshold policies is utilized for the correction of the strategy in case the cost of execution deviates from its estimate. Decentralized query management and a broadcast network is assumed. The cost measure used is the size of partial results.

The problems of RA and QPS were attacked simultaneously by 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.

Query optimization in a star computer network is investigated by Kerscheberg, Ting and Yao (1982). The queries considered are assumed to be in a disjunctive normal form where each disjunction is a "sub-query" in a conjunctive normal form with each term having at most one two-variable operator. The QPS optimizes the query response time by minimizing secondary storage data access performed by information processors. The effect of communications speeds on the strategy's performance is investigated as well.

Bernstein et al. describe a QPS which was adopted for a System for Distributed Databases - SDD-1 (Bernstein, 1981). Queries are first translated into a relational calculus form called "envelope". There are two phases in the processing of a query. First, a query is reduced into several sub-queries by a decomposition process which was adopted from Youssefi (1979). Data for a particular sub-query is transferred to one chosen site where it is processed by use of semi-joins (reduction phase) and then assembled at one site for final processing. The cost function used is the size of operands (relations). Fragmented relations are considered only in the context of increased parallelism.

Relational algebra query processing optimization is considered by Chu and Hurlay (1979, 1982). A tree representing the original query's algebraic operations is constructed. Operations in this tree are then permutated to investigate alternate strategies. For that purpose, rules based on the permutability properties of relational algebra for identifying available/correct permutations are given. The objective of the optimization is the minimization of the volume of transferred data.

COST FUNCTIONS FOR QUERY PROCESSING IN A DDB

Fragmented relations are treated in a mathematical model by Pelagatti and Schreiber (1979). The model is targetted for application design stage, thus the speed at which a QPS for any query is arrived at is unimportant. The cost measure of the model is the data transfer cost. A framework for the quantitative analysis of performance based on various correlations between joining attributes, and between attributes and horizontal fragmentation, is provided. The model is capable of incorporation of redundant (multicopy) relations as well.

A special type of a query, a "set query", applied to relations, of which one may be horizontally distributed, is considered by Gavish and Segev (1982). They formulate the QP problem into a mathematical integer programming model and compare results of its solution by two heuristics. The cost function under consideration is the volume of transferred data.

OBJECTIVES

It is interesting to note that although different cost functions are claimed to be optimized by authors of various QPS formulators, in most cases the cost that is actually minimized is the sum of the size of partial results. This is assumed to minimize either the total volume of data processed by information processors or transferred over the network, depending on whether communication or CPU processing is considered to be the critical factor.

For example, in a distributed environment comprised of lower capacity CPUs connected by a high throughput local network, the total volume of processed data may be considered to be the critical parameter and hence it is a good candidate for adoption as a cost metric. It may be argued that by minimization of the size of partial results the total volume of processed data will be minimized as well, which in turn implies optimization of the response time.

Similarly, in an environment of high capacity processors which communicate by use of standard services provided by a public network, the communication delay may be considered to be the most critical factor. Minimization of the size of partial results will decrease the total volume of data transfer and hence decrease the query response time.

The aforementioned arguments ignore the potential for parallelism. A higher volume of processed or transferred data does not necessarily result in an increase in the response time, if the data is processed and/or transferred in parallel.

Matters are further complicated by the fact that trade-offs exist which depend on the choice of a cost function/measure. If the objective of the QPS formulation is the minimization of the query's total time delay, a higher volume of data transferred over the network and/or processed by CPUs, may be used to attain a higher parallelism in operations and thus a decrease in the query's time delay. In other words, a higher dollar cost is traded for a decrease in the time delay. Similarly, if the cost measure, used by the QPS formulator, is the dollar cost of communication, the optimal strategy may have a high cost in terms of the total time delay and the volume of data processed. In addition, various applications of a DDB imply different network and CPU processing unit costs.

It becomes apparent from the above discussion, that a particular cost measure may be suitable for one application but unsuitable for others. In an interactive environment, the time delay is considered to be the most critical factor and hence should be adopted as a measure of the query's cost, while in a batch environment a measure of throughput is more appropriate. Perhaps, the minimization of the size of query's partial results is a good compromise. To determine what cost function is to be used in the formulation of a QPS, a method/tool must exist that enables their comparision.

A relatively simple way to compare various cost functions for a particular application of a DDB, is to collect the set of expected queries and examine the cost of their execution using various cost measures while taking into consideration the frequency of their execution. This method, however, assumes that the optimal strategy and its cost can be determined for each query and cost function/measure under consideration. Optimal strategies can be determined from a space of strategies available for processing of a query, provided that each strategy has its cost calculated using various cost functions under consideration.

An algorithm, which creates the space of available strategies for any given query, is presented in Section 3. The space of strategies is represented by a graph organized in levels. Each graph node corresponds to a partial result obtained by a unique sub-strategy. The nodes of the final level represent unique strategy whose cost in various cost measures, is known. What cost functions are included is discussed in Section 4, which is followed by a summary.

2. ASSUMPTIONS

Assumptions on the query processing environment together with the terminology definition are now presented.

COST FUNCTIONS FOR QUERY PROCESSING IN A DDB

A. DDB Organization/Structure

Query processing is assumed to exist in the environment of a relational DDB based on, or similar to, ADD - Architecture for Distributed Databases - which utilizes a layered approach ((Mahmoud, 1979), (Toth, 1980, 1982)). Layers of interest are the Global view, Partitioned view and Distributed view layers.

In a relational DDB, relations may be fragmented vertically or horizontally (Mahmoud, 1979). Briefly, a horizontal fragmentation partitions a relation into subrelations according to a value of one partitioning key. All subrelations have mutually exclusive keys and identical attributes. Vertical fragmentation, on the other hand, is such that the key attribute(s) are replicated in all subrelations but the remaining non-key attributes are mutually exclusive. The cardinality of all sub-relations is therefore identical.

The user-issued query is transformed into the relational calculus conjunctive normal form defined on the global view of the DDB. The query becomes a conjunction of terms

$$T_1 \wedge T_2 \wedge T_3 \wedge \cdots ,$$

in which each conjunction T_r is either a one-variable (restriction), or a two-variable (join) term. For example, the query $(R_i.a > 5) \wedge (R_j.b < R_k.c)$, where R_i, R_j and R_k are relations and a, b and c are attributes, contains two terms, the first is a one-variable restriction, while the second is a two-variable join. Relational arithmetic operators are drawn from a set of operators {=, >, <, >=, >=, =/=}.

The assumption that each term is at most two-variable is not restrictive since a term containing three or more relational variables can be transformed into a conjunction of two or more two-variable terms ((Ullman, 1980), (Epstein, 1979), (Mahmoud, 1979)). For example, a three-variable term $(R_i.a > R_j.b = R_k.c)$ can be transformed into an equivalent conjunction of two two-variable terms $(R_i.a > R_j.b) \wedge (R_j.b = R_k.c)$.

A partitioned view processor defines the partitioning of global relations into vertical and horizontal fragments ((Chang, 1975), (Mahmoud, 1979)), or elementary relations, and it transforms the global view query to reflect this partitioning (Toth, 1980). A distributed view processor simply identifies the network locations of elementary relations.

B. Network and Static/Dynamic Strategy

A DDB is assumed to operate in the environment of a point-to-point network (e.g., ARPANET, DATAPAC). A QPS formulator is assumed to generate a static strategy. Once a strategy is formulated, it remains unchanged throughout the query exection phase.

C. Single/Multi Query Optimization

Strategies considered optimize strategies for a single query only; multi-query environment as proposed aby Cellery and Mayer (1981) is not considered.

D. Accepted Rules for QPS Formulation

Several generally accepted rules for QPS formulation, which were discussed in the literature review section, are adopted. In particular:

(i) Restrictions and projections are performed as soon as possible.
(ii) Cascaded one-variable operators are executed simultaneously.
(iii) Cascaded two-variable operators are executed simultaneously.

3. SPACE OF STRATEGIES

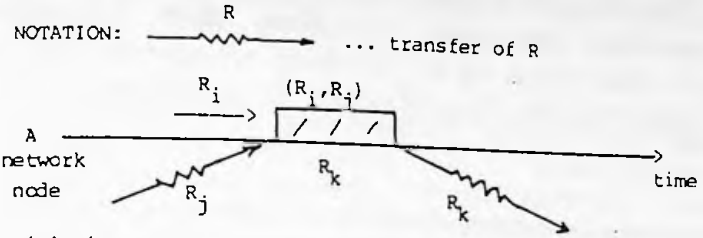
The problem of the creation of a space of strategies is addressed in this section. Before the algorithm for its creation is examined, examples of strategies which demonstrate that a QPS is defined not only by the sequence of joins to process two-variable terms but also by the locations where joins are executed, are given.

EXAMPLES OF QP

Assume that relations R_1, R_2, and R_3 are relations in network nodes A, B, and C respectively; a, b, ... are attributes; and θ_1, θ_2, ... are arithmetic relational operators. A simple query having only two two-variable terms $(R_1.a\ \theta_1\ R_2.b) \wedge (R_2.c\ \theta_2\ R_3.d)$ can be processed by various strategies. One of the strategies is:

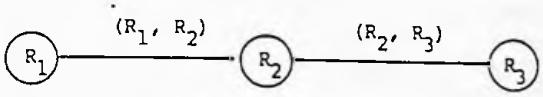
(i) join relations R_1 and R_2 in network node A to give relation R_{12},
(ii) join relations R_{12} and R_3 in network node C to give the query result.

The notation for timing diagrams is defined in Figure 1(a). The query and the timing diagram for the above strategy is shown in Figure 1(b) and (c) respectively. The timing diagram shows that relation R_1

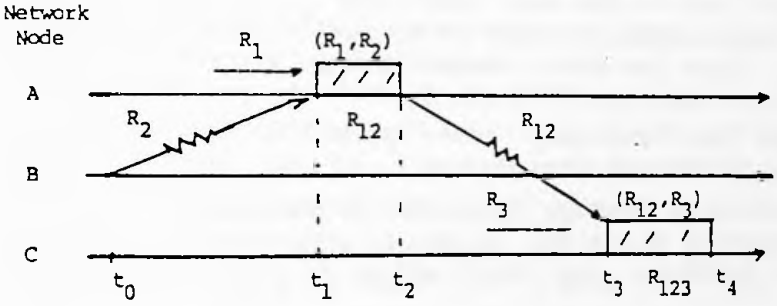


A join is executed to process a two-variable term $(R_i \; \theta \; R_j)$. Relation R_i is located in network node A already, while relation R_j arrives just prior to the join execution of R_i with R_j. The join results in relation R_k which is then immediately transferred to some other node.

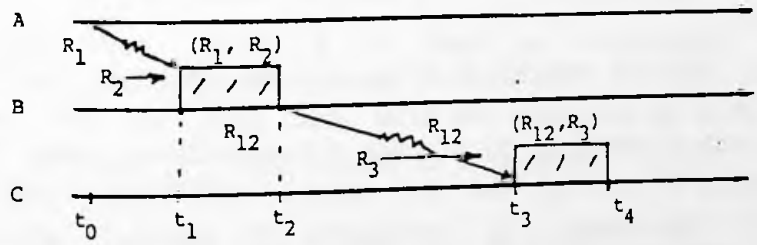
1(a) Notation for strategy timing diagrams



1(b) Query $(R_1.a \; \theta_1 \; R_2.b) \; \hat{} \; (R_2.c \; \theta_2 \; R_3)$



1(c) Strategy timing diagram



1(d) Strategy timing diagram

Figure 1 A query and two strategies for its processing

is transferred to node B and immediately upon its arrival at time t_1, the join of relation R_1 and R_2 commences to process the two-variable term ($R_1.a\ \theta_1\ R_2.b$). The join is completed at time t_2 giving relation R_{12}, which is then transferred to node C. When this transfer is completed at time t_3, the join of R_{12} with R_3 to process ($R_{12}.c\ \theta_2\ R_3.d$) commences. The query result, relation R_{123}, is available in node C at time t_4.

The second strategy, whose timing diagram is shown in Figure 1(d), may be:

(i) join relations R_1 and R_2 in network node B to give relation R_{12},
(ii) join relations R_{12} and R_3 in network node C to give the query result.

Both strategies are identical in the sequence of join execution. They both specify that first R_1 is to be joined with R_2 giving R_{12}, followed by the join of R_{12} with R_3 in node C to give the query result, while their only difference is in the location of the first join is execution. One specifies node A, while the other node B.

If it is assumed that the processors in nodes A and B have an identical load and are of the same capacities, ie., they have the same processing power and incure the same delay due to access to secondary storage devices, then the delay caused by the processing of a join of R_1 with R_2 in node A will be identical to that of the same join but executed in node B. However, even under this assumption, the two strategies produce different time delays. If R_1 and R_2 are of a different size, which is likely, then the transfer cost of R_1 from node A to B, as specified by the first strategy, will differ from the cost of transfer of R_2 from node B to A which is specified by the second strategy.

In addition to assumptions of Section 2, the following assumptions are made by the algorithm which builds a space of strategies.

a) The first assumption is made on a priori cost and size estimation of partial resutls. Given two relations R_i and R_j, an algorithm is able to estimate the size and cost of the partial result of joining R_i with R_j to process a two-variable term ($R_i.a\ \theta\ R_j.b$). Relations R_i and R_j can be either elementary relations referenced by the query, or estimates of partial results of processing of some two-variable terms. Included in the estimation is the removal of joining attributes (attributes 'a' and 'b' for the above relations R_i and R_j) by a projection. Projection is included only in the case, of course, when these attributes are not 'target' attributes and are no longer necessary for execution of

some other joins necessary to process the query.

b) The second assumption refers to the destination of the query result. When a query is issued against the DDB, the desired destination, to which the query result should be delivered by the DDB, is specified as well. The destination node is typically, but not necessarily, that node in which the query is issued. The strategy for processing of a given query must be such that the query result is delivered to the specified destination node. It is assumed that when a query processing strategy under examination is such that the query result is not obtained in the destination node, then the strategy is augmented by the final transfer of the result to the destination node. The total cost of a strategy is increased by the cost of this final data transfer.

c) The third assumption specifies that a join of two relations R_i and R_j can be executed only in a network node where either R_i or R_j is located.

d) Finally, it is assumed that relations referenced by a query are not partitioned either horizontally or vertically. A query may therefore be considered to be expressed on the global view of the data base (Toth 1980, 1982).

CREATION OF A SPACE OF STRATEGIES

The development of a space of strategies for a given query can be demonstrated by the following example.

Consider a simple query $(R_1.a > R_2.b) \hat{} (R_2.c < R_3.d)$ referencing relations R_1, R_2, and R_3 located in network nodes A, B, and C respectively. The space of strategies must be such that it reflects a choice of the sequence of joins and their locations of execution. Options for the selection of the first join, together with the location of its execution, are:

(i) ${}_1R^A_{01}$ and ${}_2R^B_{02}$ are joined in A (ie., ${}_2R^B_{02}$ is moved to node A) to give relation ${}^{01,02}_{12}R^A_{11}$

(ii) ${}_1R^A_{01}$ and ${}_2R^B_{02}$ are joined in B to give ${}^{01,02}_{12}R^B_{12}$

(iii) ${}_2R^B_{02}$ and ${}_3R^C_{03}$ are joined in B to give ${}^{02,03}_{23}R^B_{13}$

(iv) ${}_2R^B_{02}$ and ${}_3R^C_{03}$ are joined in C to give ${}^{02,03}_{23}R^C_{14}$

The labelling scheme for relations, subscripts and superscripts, will be presented shortly.

The above options can be represented by a directed graph in which nodes represent either elementary relations, ie., relations referenced by the query, or relations which are results of some join executions to process two-variable terms. Edges outgoing from a graph node identify relations from which it was derived by an execution of a join.

To avoid the confusion between network nodes and nodes of a graph, the graph nodes will be referred to as graph elements, or as elements. Since the graph nodes, or elements, represent relations they will also be sometimes referred to simply as relations so that the repetition of the phrase "element representing a relation" is avoided.

Figure 2(a) shows the above options represented by a directed graph. As the elementary relations are not results of any join execution, the elements representing them do not have any outgoing edges. The graph elements representing results of the first join, on the other hand, have two outgoing edges each. Edges identify relations from which the graph element was derived by a join. The graph elements are labelled using the notation $\overset{lsp}{lsb}R\overset{rsp}{rsb}$, in which the left and right subscripts and superscripts have the following meaning:

(i) RIGHT SUPERSCRIPT identifies a network node in which the relation is located by a use of capitals A, B, C, ...

(ii) RIGHT SUBSCRIPT is the identifier, or ID. It consists of two parts. The first part is the number of joins executed to derive the relation, while the second one is a number which is unique for all elements derived by the particular number of joins. For instance, if two relations are derived by exactly two joins, the first part (which is actually the first digit of their right subscript) will be 2, while their second parts (consisting of the remaining digits) will be different. It is tacidly assumed that any query can be processed by at most 9 joins, but this notation will suffice for exposition purposes. If an element represents an elementary relation referenced by the query, then the first digit of its ID is 0, as the relation is not a result of any join. Note that a query definition may specify relations R_1, R_2, ..., up to R_9 only, but again, this is deemed to be sufficient. There is a direct correspondence between the query definition and the directed graph representing the space of strategies. An elementary relation, ie., relation originally referenced by the query, is denoted by R_i, i=1, 2, ..., 9 in the query specification, while in a directed graph it is represented by an element with a label $\overset{lsp}{lsb}R\overset{rsp}{0i}$, ie., its right subscript is the same, only it is preceeded by a "0".

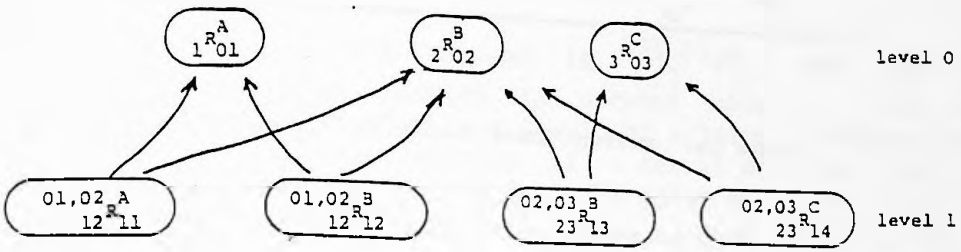


Figure 2(a) Graph Nodes Representing Elementary Relations (Level 0) and Results of One Join (Level 1)

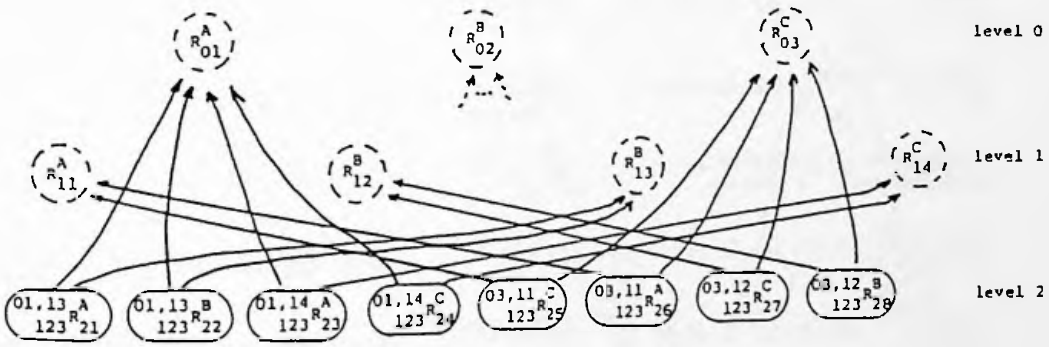


Figure 2(b) Elements Representing Relations which are Results of Two Joins (Level 2)

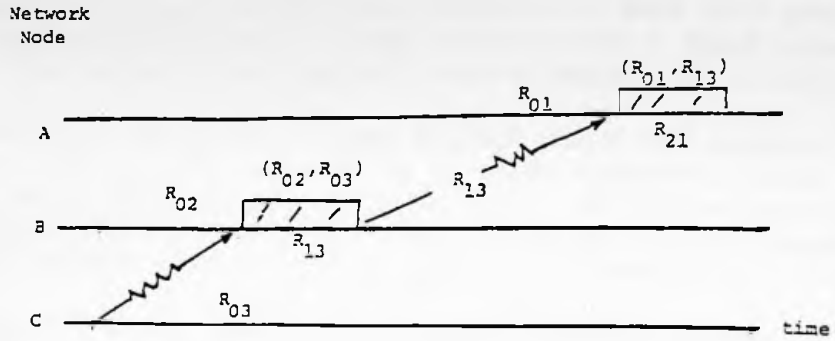


Figure 2(c) Strategy Timing Diagram for $^{01,13}_{123}R^{A}_{21}$

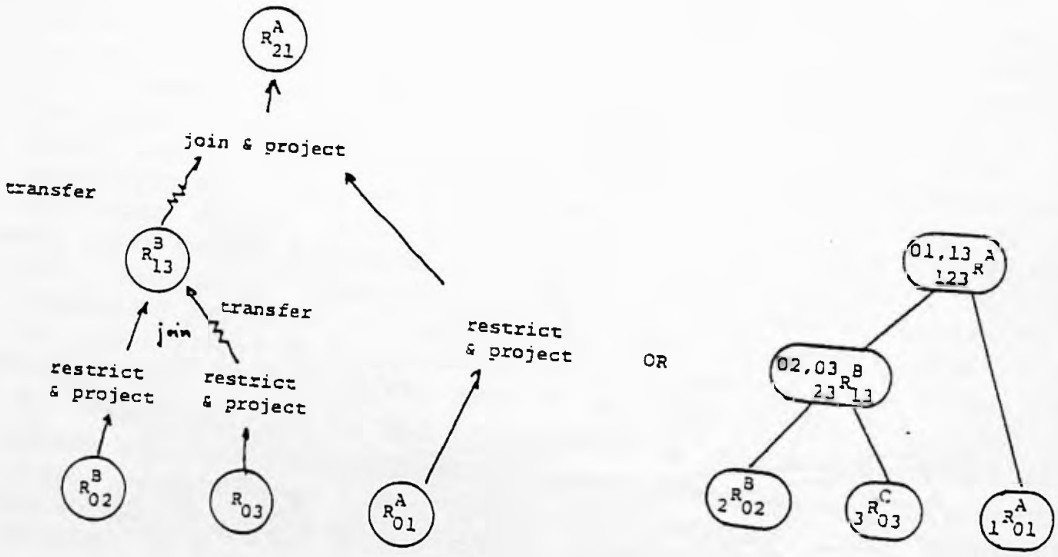


Figure 2(d) Tree of Elements Representing a Strategy for $^{01,13}_{123}R^{A}_{21}$

(iii) LEFT SUPERSCRIPT consists of a pair of ID's of relations processed by a join to create the element. For each of the two IDs in the left superscript there is an outgoing edge to the element with that ID, ie., to an element with a right subscript matching the ID.

(iv) LEFT SUBSCRIPT identifies relations processed by joins to derive the relation represented by the element. If the relation is a result of a join of relation R_1 with R_2, followed by a join of the result with R_3, then the left subscript will be 123.

For instance, if the element is ${}^{121,03}_{123}R^{A}_{21}$, then the element represents a relation located in network node A. Its ID is 21 signifying that the relation is a result of 2 joins. Because the left subscript is 123, the relation is a result of joins which process elementary relations R_1, R_2, and R_3. The relation is a result of a join of two relations represented by elements with ID's 121 and 03.

Continuing with the example, consider relations which are results of exactly two joins:

(i) ${}_{1}R^{A}_{01}$ is joined with ${}^{02,03}_{23}R^{B}_{13}$ in node A to give ${}^{01,13}_{123}R^{A}_{21}$

(ii) ${}_{1}R^{A}_{01}$ is joined with ${}^{02,03}_{23}R^{B}_{13}$ in node B to give ${}^{01,13}_{123}R^{B}_{22}$

(iii) ${}_{1}R^{A}_{01}$ is joined with ${}^{02,03}_{23}R^{C}_{14}$ in node A to give ${}^{01,14}_{123}R^{A}_{23}$

(iv) ${}_{1}R^{A}_{01}$ is joined with ${}^{02,03}_{23}R^{C}_{14}$ in node C to give ${}^{01,14}_{123}R^{C}_{24}$

(v) ${}_{3}R^{C}_{03}$ is joined with ${}^{01,02}_{12}R^{A}_{11}$ in node C to give ${}^{03,11}_{123}R^{C}_{25}$

(vi) ${}_{3}R^{C}_{03}$ is joined with ${}^{01,02}_{12}R^{A}_{11}$ in node A to give ${}^{03,11}_{123}R^{A}_{26}$

(vii) ${}_{3}R^{C}_{03}$ is joined with ${}^{01,02}_{12}R^{A}_{12}$ in node C to give ${}^{03,12}_{123}R^{C}_{27}$

(viii) ${}_{3}R^{C}_{03}$ is joined with ${}^{01,02}_{12}R^{A}_{12}$ in node B to give ${}^{03,12}_{123}R^{B}_{28}$

The graph representation for the above options is derived from a graph shown in Figure 2(a). For each relation which is a result of two joins, a new element with two outgoing edges is added to the graph. Edges of any new element terminate at relations processed by a join to derive the element. The directed graph is shown in Figure 2(b).

DIRECTED GRAPH

Assume the usual definition of a "sink" node - it is a node with no outgoing edges. In a directed graph representing a space of strategies, a sink node always represents one of the elementary relations. Elements which represent the query result obtained by different strategies have no incoming edges, call such elements "final". Given a final element, its strategy can be obtained simply by following its outgoing edges. For the example of Figure 2(b), the element $^{01,13}_{123}R^{A}_{21}$ identifies a strategy shown in Figure 2(c), which is determined by following all the paths from the final node to the sinks.

Obviously, the graph is connencted, there is at least one incoming or outgoing edge for every element. A graph element/node is either (i) a sink representing an elementary relation referenced by the query, in which case there must be at least one incoming edge as the relation must be processed by a join, or (ii) an element representing a relation which is a result of some processing and therefore it must have outgoing edges.

Definition: Level p of the directed graph representing the space of available strategies is a set of nodes which represent relations processed by exactly p joins.

Certain observations, actually theorems presented without their obvious proofs, that can be made about the graph properties follow.

OBSERVATIONS:

Observation 1: Levels are mutually exclusive sets of graph nodes.

Observation 2: Every node of the graph is in exactly one level.

Observation 3: The graph has levels 0, 1, 2, ..., n_j, where n_j is the number of joins necessary to process a query, ie., the graph has $n_j + 1$ levels.

Observation 4: A node of level $p > 0$ has exactly two outgoing edges to nodes of level p_x and p_y, such that $p_x, p_y < p$ and $p=p_x+p_y+1$.

Observation 5: Every node, which is not a final node, must have at least one incoming edge.

Observation 6: Every final node, ie., a node representing the query result obtained by a particular strategy, is a root of a binary tree which defines the strategy to derive it. The tree representing the strategy of Figure 2(c) is shown in Figure 2(d). Note that this is the usual representation of a QPS appearing in the scientific literature on QPSs.

NUMBER OF LEVELS / NUMBER OF NECESSARY JOINS

Each element of the directed graph at level p represents a relation which is a result of processing of p joins. The number of levels in a graph, therefore, depends on the number of joins which are necessary to process a given query, while it, in turn, depends on the number of two-variable terms in the query definition and the number of query's cascaded joins. However, a much simpler relationship exists for a global view query (Bodorik, 1984) which implies that if the number of relations refernced by a query is n_r then the number of joins, some of which may be cascaded, to process the query is $n_j = n_r - 1$.

The number of joins which is necessary to process a query is not only of general interest, but it is also used by the following algorithm which creates the space of available strategies.

ALGORITHM

Observation 6 infers that a directed graph representing the space of available strategies can be viewed as a collection of binary trees, each representing one QPS, such that the trees may "share" some of their nodes. The problem is how to construct the graph of strategies for a given query in an efficient way and without any redundancies. In other words, the graph should be such that each of its elements represents a partial result derived by a unique strategy. If such a directed graph is constructed for a query and the cost of relations represented by the graph elements is known, then it is a simple task to examine all final nodes to find the one with the minimum cost. It identifies the query's minimum cost strategy, while the strategy itself is defined by a tree of which the minimum cost node is a root.

The example of Figure 2 demonstrates how the algorithm constructs the directed graph level by level. Initially, level 0 is built from the elementary relations. The cost of a relation of level 0 includes the cost of one-variable processing, ie., the cost of restrictions and a projection to remove attributes which are not necessary for further processing. Representation of possible ways to create a partial result of exactly two joins is created from the graph of Figure 2(a) by an addition of another level as shown in Figure 2(b). An element of a new level p is created from two elements of the previous levels p_x and p_y such that $p = p_x + p_y + 1$. In addition, elements of levels p_x and p_y must be such that they represent relations that can be joined. For instance, in Figure 2(a), R_{12} and R_{01} can not be joined, neither can R_{01} and R_{03}.

Paraphrasing, a relation R, which is a result of p joins and is therefore an element of level p, can be created by a join of two relations R_i and R_j of levels p_x and p_y respectively, where R_i is a result of p_x joins, while R_j of p_y joins. If R_i and R_j are joined to give R then R will be a result of $p_x + p_y + 1$ joins. As a result, relationship $p = p_x + p_y + 1$ must be satisfied.

To summarize, the graph construction is started by the creation of level 0, whose nodes represent original relations referenced by the query. Then level 1, 2, ..., n_j are built in that order, where n_j is the number of joins necessary to process a given query. Nodes of level p represent relations which are results of exactly p join executions. An element of level p is constructed from elements of levels p_x and p_y such that $p = 1 + p_x + p_y$. The number of graph levels is equal to the number of joins necessary to process a query. Any time a new element representing relation R is constructed from elements representing relations R_i and R_j, its cost and size is estimated. This comforms to the assumption (a) of this section. In addition, pointers to nodes representing R_i and R_j are added, they represent graph edges to relations from which the new node is created.

An element of the last level represents the query result obtained by a particular strategy. If this result is not located in the destination node, the element's cost is increased by the cost of transfer to the destination node and the strategy is augmented by this transfer as well. This comforms to the assumption (b).

Since each element of the last level represent the query result obtained by a different strategy, to find the optimal strategy, the last level is searched for the node with the minimum cost. The optimal strategy is found by following the pointers from the minimum cost element to the sinks representing query's original relations.

An algorithm, which constructs a graph of available strategies for a given query, is shown in Figure 3. It creates a graph representing all available strategies and, in addition, the graph is such that it does not contain any redundant elements. The redundancy is avoided by taking into consideration all of the available information any time a new graph element, which represents a sub-strategy, is created.

PARTITIONED RELATIONS

The algorithm of Figure 3 assumes that the query is expressed on the glogal view of the DB. This assumption implies that a query does not reference relations which are partitioned either horizontally or vertically. Since realistic applications of a DDB necessitate the use of partitioned relations, this assumption must be removed in order to allow a query to be expressed on the distributed view of the DB, thus

COST FUNCTIONS FOR QUERY PROCESSING IN A DDB

INPUT: Query Q defined by $\{(R, L, INFO)_i\}$ and $\{2\text{-}V_u\}$, where
(i) $\{2\text{-}V_u\}$, u = 1, 2, ... is a set of two-variable terms for query Q, and
(ii) $\{(R, L, INFO)_i\}$, i = 1, 2, ... is a set of quadruples conveying information about relations referred to by the two-variable terms, where
- R_i is a relation referred to by a two-variable term in $\{(2\text{-}V)_u\}$
-
- L_i is the network location of R_i
- $INFO_i$ is information about relation R_i which is necessary for size and cost estimation of join processing.

OUTPUT: The graph of QP strategies and their costs. QPS are

Method: The graph of strategies is created by levels. Each graph element represents either the original relation referenced by the query (elementary relation) or a relation which is a partial result of processing of one or more joins. Level 0 represents elementary relations with processed restrictions and projections that remove unnecessary attributes (ie., attributes that are neither target or joining attributes), level 1 represents relations which are partial results of processing exactly one join, and level p represents relations which are partial results of exactly p joins. Each graph node contains:

- two pointers to its predecessors from which it is created
- pointer to the next element of the same level
- quadruple conveying information about the relation the node represents, ie., if the relation is R_i then the quadruple is $(C, L, \{R\}, INFO)_i$, where
  - C_i is the cost of relation R_i
  - L_i is the network location of R_i
  - $INFO_i$ is information about relation R_i which is necessary for size and cost estimation of join processing.
  - $\{R\}_i$ is a list of elementary relations processed by joins to derive R_i. For example, if R_i is a result of a join of R_1 with R_2, followed by the join of the result with R_3, then $\{R\}_i = \{R_1, R_2, R_3\}$.

Note the correspondence of the information stored in each graph node and the labelling scheme for graph elements in pictorial representation representation of the directed graph. For relation R_i at level p and its quadruple $(C, L, \{R\}, INFO)_i$, the right subscript of its label is "i" preceded by "p", ie., "pi". The right superscript is L_i, the location of the relation. The left superscript corresponds to the two graph edges - the graph node pointers. The left subscript corresponds to $\{R\}_i$. If L_i=A, $\{R\}_i = \{R_1, R_2\}$, i=7, and the relation is at level 1, then in the pictorial representation of the directed graph, the graph node will be labelled ${}^{01,02}_{12}R^{A}_{17}$.

STEPS

1. Create level zero of the graph of strategies directly from relations referenced by the query and set the cost of each element to the cost of its restrictions and projections.

2. Do steps (2) to (4) for p = 1, 2, ..., n_j, to create levels 1, 2, ..., n_j, where n_j is the number of necessary joins to process the query.

3. Create level p from elements of previous levels by execution of step (4) for p_x = p-1, p-2, ... !p/2!.

4. For each tree element R_i of level p_x, search level p_y, such that p = p_x + p_y + 1, for those elements which represent a relation which can be joined with R_i. For each such found element R_j, create a new element which represents relation R_k arrived at by a join of R_i with R_j in the network node where R_i is located. Set the quadruple $(C, L, \{R\}, INFO)_k$ of the new element representing relation R_k to:
   - C_k is set to the cost of R_k
   - L_k is set to L_i
   - $\{R\}_k$ is set to the union of $\{R\}_i$ and $\{R\}_j$
   - $INFO_k$ is derived from $INFO_i$ and $INFO_j$

   The pointers of the new nodes are set to identify the elements representing R_i and R_j. Finally, the new node is inserted to level p. If p_x =/= p_y and L_i =/= L_j (ie., R_i and R_j are not located in the same network node) then create an additional node which represents the join of R_i with R_j in network node L_j where R_j is located and insert it to level p.

Figure 3 Algorithm to develop a graph of available strategies

allow partitioned relations. The algorithm was extended to process relations which are partitioned horizontally and/or vertically and it can be found in (Bodorik, 1984) - it is not discussed here any further.

4. COST FUNCTIONS

The algorithm of Figure 3 does not specify what cost function is utilized to determine the cost of partial results and strategies. Its only assumption is that given two relations and information about them, ie., their cost, cardinalities, locations and other information necessary for estimation of the size of partial result, there is a cost function which evaluates the cost of the join operation and the cost of the data transfer. The cost of CPU processing and data transfer can be obtained with reasonable accuracy provided that the size of relations is known; it will depend on the unit costs for CPU processing and data transfer.

Futhermore, each partial result and strategy can have its cost determined in more than one cost measure simultaneously. In fact, the algorithm was implemented (Bodorik, 1984) to calculate the cost of each partial result using the following measures:

total time delay;
time delay due to CPU processing only;
time delay due to communication only;
total dollar cost (depends on the volume of data processed and transferred);
dollar cost due to the CPU processing only (depends on the volume of data processed);
dollar cost due to the data transfer only (depends on the volume of data transferred);
sum of the sizes of partial results.

Since the algorithm develops the space of strategies in which the cost of each strategy is given in various cost measures, optimal strategy for each of the measures can be found by a search through the final level of the graph. Once a minimum cost strategy is found for one cost measure, for example for the total time delay, the strategy's cost using other measures, for instance, the total dollar cost, is found as well. As a result, the algorithm can be used as a tool for investigation on the choice of a cost function for processing of queries.

5. CONCLUSION

Various cost functions can be used in the formulation of a QPS in a DDB. The choice of a cost function depends on the application and the environment in which the DDB exists. The query's cost depends not only on the delays and dollar unit costs due to CPU processing and data transfer, but also on the strategy formulation which minimizes the query's cost using a particular cost function.

An algorithm was described which creates a space of available strategies for processing of any given query. As the algorithm finds the cost of each strategy utilizing various measures/cost functions, it can be used as a tool in comparision and a choice of cost functions for any application of a DDB.

The algorithm was implemented and will be tested on several applications of a DDB in the near future.

REFERENCES

Bernstein, P., A., "Approaches to Concurrency Control in Distributed Data Base Systems", AFIPS, 1979, 813-820.

Bernstein, P. A., et. al., "Query Processing in a System for Distributed Databases (SDD-1)", ACM TDS, Vol. 6, No. 4, Dec. 1981, 602-625.

Black, P. and Luk, W. S., "A New Heuristic for Generating Semi-Join Programs for Distributed Query Processing", Proc. of COMPSAC 82, IEEE Computing Society's 6th International Conf., 1982, 581-588

Bodorik, P., "Query Processing Strategies in a Distributed Data Base", Ph. D. dissertation (in preperation), Carleton University, Ottawa, 1984 (expected).

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

Chang, S-K. and Cheng, W-H., "A Methodology for Structured Database Decomposition", IEEE TOSE, March 1980, Vol. SE-6, NO. 2, 205-218.

Chu, W. W. and Hurley, P., "A Model for Optimal Query Processing for Distributed Data Bases", CG1393-8/79, 1979 IEEE, 116-122.

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

Epstein, R. and Stonebraker, M., "Analysis of Distributed Data Base Processing Strategies", 6th International Conf. on Very Large Data Bases, Montreal, Quebec, 1980, 92-101.

Gavish, B. and Seger, A., "Query Optimization in Distributed Computer Systems", Management of Distributed Data Processing, North Holland Publ. Comp., 1982, 233-252.

Henever, A. R. and Yao, S. B., "Query Processing in Distributed Data Base Systems", IEEE TOSE, vol. SE-5., No. 3, May 1979, 177-187.

Kerschberg, L., Ting, P. D. and Yao, B., "Query Optimization in Star Computer Networks"., ACM TDS, Vol. 7, No. 4, Dec. 1982, 678-711.

Mahmoud, S. A., Riordon, J. S. and Toth, K. C., "Distributed Database Partitioning and Query Processing", IFIP-TC-2, Venice, Italy, 1979, 32-51.

Paik, I-S and Delobel, C., "A Strategy for Optimization of Distributed Query Processing", Proc. 1st Int. Conf. on Distributed Computing Systems, Huntsville, Alabama, Oct. 1979, 686-698.

Pelagatti, G. and Schreiber, F. A., "Evaluation of Transmission Requirements in Distributed Database Access", ACM-SIGMOD 1979, Int. Conf. on Mamagement of Data, May 1979, Boston, Mass., 102-108.

Reynolds, C. H., "Issues in Centralization", Datamation, March 1977.

Rockart, J. F. and Leventer, J. S., "Centralization vs. Decentralization", Working Paper WP-845-76, April 1976, MIT Sloan School of Management, Cambridge, Mass., USA.

Smith, J. M. and Chang, P. Y., "Optimizing the Performance of a Relational Algebra Database Interface", CACM, Vol. 18, No. 10, Oct. 1975, 568-579.

Stonebraker, M. and Neuhold, E., "A Distributed Database Version of Ingres", Memorandum No. ERL-M612, Sept. 1976, Eng. Research Lab., University of California, Berkeley, California.

Toan, N. G., "A Unified Method for Query Decomposition and Shared Information Updating in Distributed Systems", First Int. Conf. on Distributed Computer Systems, Huntsville, Alabama, Oct. 1979, 679-685.

Toan, N. G., "Distributed Query Management for a Local Network", Proc. 2nd Int. Conf. on Distributed Computing Systems, Paris, France, April 1981, 188-196. Toth, K. C., "Distributed Database Architecture and Query Processing", Ph. D. Dissertation, Carleton University, Ottawa, Ontario, 1980.

Toth, K.C., et. al., "Query Processing Strategies in a Distributed Database Architecture", Distributed Data Sharing Systems, North Holland Publ. Comp., 1982, 117-134.

Ullman, J. D., "Principles of Database Systems", Computer Science Press, 1982, 2nd edition.

Wong, E. and Youssefi, K., "Decomposition - A Strategy for Query Processing", ACM TDS, Vol. 1., No. 3., Sept. 1976, 223-241.

Youssefi, K. and Wong E., "Query Processing in a Relational Database Management System", IEEE, CH1406-8/79, 407-417.

Yao, S. B., "Optimization of Query Evalution Algorithms", ACM TDS, Vol. 4, No. 2, June 1979, 133-155.