CORRECTING EXECUTION OF DISTRIBUTED QUERIES

P. BODORIK†, J. PYRA† and J.S. RIORDON††

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

††Dept. of Systems and Computer Engineering,
Carleton University,
Ottawa, Ontario, K1S 5B6, Canada

ABSTRACT

Algorithms for processing distributed queries require a priori estimates of the size of intermediate relations. Most such algorithms take a "static" approach in which the algorithm is completely determined before processing begins. If size estimates are found to be inaccurate at some intermediate stage, there is no opportunity to re-schedule, and the result may be far from optimal. Adaptive query execution may be used to alleviate the problem. Care is necessary, though, to ensure that the delay associated with re-scheduling does not exceed the time saved through the use of a more efficient strategy. This paper presents a low overhead delay method to decide when to correct a strategy. Sampling is used to estimate the size of relations, and alternative heuristic strategies prepared in a background mode are used to decide when to correct. Correction is made only if lower overall delay is achieved, including correction time. Evaluation using a model of a distributed data base indicates that the heuristic strategies are near optimal. Moreover, it also suggests that it is usually correct to abort creation of an intermediate relation which is much larger than predicted.

1. INTRODUCTION

Although researchers are currently turning their attention to multi-query optimization [Cellary, 1980; Ounegbe, 1983; Carey, 1985; Kim, 1985; Reuter, 1986; Su, 1986; Sellis, 1988] and adaptive query processing, most research on distributed query processing assumes a single query environment and static processing. Furthermore, most research concentrates on an important class of queries, Select-Project-Join (SPJ) [Ceri, 1984; Yang, 1987]. Use of semi-joins in optimizing SPJ type queries has received a great deal of attention [Hevner, 1979, 1980; Bernstein, 1981; Apers, 1983; Yu, 1982, 1983, 1985; Perrizo, 1984; Ceri, 1986; Masuyama, 1987; Otoo, 1987]. Although use of semi-joins has been widely accepted as a good processing tactic, it has also been recognized [Epstein, 1980; Lafortune, 1986] that semi-join and generalized join processing tactics must integrated. Epstein et al. [1978] proposed one of the earliest methods which uses joins as a processing tactic; it results from is an adaptation of INGRES to the distributed environment [Epstein, 1986]. A join processing heuristic algorithm for the ADD system [Mahmoud, 1979], first decomposes a query into "Class A" sub-queries which produce results processed by a sub-optimal heuristic. [Daniels, 1982; Selinger, 1980; Lohman, 1985; Mackert, 1986a, 1986b] describe query optimization in System R*.

After a user query is parsed and transformed into a canonical form [Ceri, 1984], an optimizing algorithm is used to form a strategy, i.e., a specification of the sequence and locations of the individual relational operations whose result satisfies the query. Optimization is carried out on the basis of a specified cost function, typically related to processing and/or data transfer delay [Bodorik, 1988a]. The strategy is distributed to *cohorts*, that is, to those information processors which participate in its execution.

In static processing a strategy is not modified once its execution begins. Static processing thus relies on accurate estimation of parameters characterizing the query and the distributed environment in which it is processed. In particular, it relies on estimates of the size of *intermediate* results [Epstein, 1980; Bernstein, 1981; Wong, 1982; Yu, 1982; Christodoulakis, 1983; Chao, 1986; Vander Zanden, 1986; Ijbema, 1986; Hwang, 1987; Bell, 1989], that is relations produced by executing relational operators such as joins. It also relies on estimates of the cost of processing and transmission over the network [Yu, 1986]. If estimates are inaccurate, the strategy may be far from optimal.

There are two possible approaches to dealing with this problem. One is to seek accurate estimates. Despite much effort in this direction, they are often unobtainable; moreover the attempt is expensive in terms of the size and maintenance of the required statistical data. Various types of adaptive (dynamic) query execution techniques comprise the second approach [Nguyen, 1981; Yu, 1983, 1986; Bodorik, 1988b, 1988c]. Execution of a strategy is monitored; if at some intermediate stage a priori estimates used in its optimization prove to be inaccurate, corrections are made with updated information.

Two general decision making methods have been proposed thus far to decide when to correct a strategy [Bodorik, 1988b]. In reformulation the unexecuted portion of the strategy is reformulated at every intermediate stage using available updated information. If the new strategy is estimated to reduce cost then correction is appropriate. In the second method reformulation occurs only when intermediate results exceed a predetermined threshold or lie outside a specified band of values. [Nguyen, 1981] uses the average size of intermediate results as one single dynamically updated threshold for the whole query. [Bodorik, 1988b] refines this approach by the use of one threshold value per intermediate result; however, these values are not updated dynamically. [Bodorik, 1989] addresses both the monitoring

This work has been supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada.

and execution phases of strategy execution. Available methods are identified for each phase; these are analyzed in terms of overhead, complexity and accuracy of information used in correcting.

This paper presents a low overhead delay method to decide when to correct a faltering strategy. During execution, alternative strategies are prepared. These may be invoked if they reduce execution time, including overhead delay. A key problem is the preparation of an adequate alternative strategy when the estimation error is unknown. It is proposed that sampling methods [Vitter, 1985; Olken, 1986] be used to predict the size of a join result before it is complete. After a fraction of the join is completed, sampling is used to predict the size of the result. If it is higher than the initial estimate and the alternative strategy is expected to reduce delay, then the original strategy is discarded and the alternative one introduced. The join which produces the larger-than-expected relation is also aborted. The proposed method is evaluated using a model of a Distributed Data Base (DDB).

2. ASSUMPTIONS

A query is assumed to be a tree query in a conjunctive normal form such that each term has at most two relational variables. This is a typical approach since it simplifies query representation but does not greatly reduce their expressiveness [Ceri, 1984]. The two-variable terms are processed by joins. As in [Selinger, 1980; Kumar, 1987], two relations are considered for joining only if they have a term present in the query. It is assumed that the values of attributes are distributed uniformly and independently of each other. The query's terms are also assumed to be independent in that no predicate is implied through others via transitivity. For each two-variable term R_i AND R_j, there exists a selectivity factor which determines the expected fraction of tuple pairs from R_i and R_j which satisfy it [Epstein, 1980; Bernstein, 1981; Wong, 1982]. These assumptions imply that an intermediate result of evaluating a set of predicates has a cardinality which is the product of the predicates' selectivities and the cardinalities of relations referred to by these predicates. Restrictions and projections are assumed to be locally optimized and executed as quickly as possible in order to reduce the sizes of relations involved in the query [Ceri, 1984; Lafortune, 1986].

In this paper the optimization objective is minimization of query response time. The result of a relational operator execution is transferred to another network location only when it is completely formed. Consequently pipelining, which may improve the query response time, is not considered. Although the paper does not explicitly consider semi-joins as a processing tactic, the proposed method is applicable to the environment in which the join and semi-join tactics are integrated [Lafortune, 1986].

3. PROPOSED (AJL) METHOD

To minimize delay, the decision to correct should be computationally simple and, moreover, a corrective strategy should already exist when it is decided to correct. The proposed method to formulate alternative strategies, termed the Aborted Join Last (AJL) method, satisfies both of these requirements. An alternative strategy is prepared for each intermediate result/relation before it is actually formed by a relational operation. Whenever such a relation is formed, the delay of the strategy under execution is estimated using the information on its new size. If the estimated delay of the current strategy is higher than that of the alternative one, then it is aborted and the alternative strategy is instituted. The difficulty here, of course, is the formulation of an alternative strategy. It is formulated concurrently with the execution of the current strategy and before the size of the intermediate result in question is known. The proposed solution avoids the necessity of computing an intermediate result by using sampling [Vitter, 1985; Olken, 1986]. It proceeds as follows:

1. Given a query, a formulator/optimizer is used to derive a processing strategy. The latter is distributed to cohorts which then cooperate in transferring relations and executing relational operations according to the strategy's instructions.

2. Concurrently with the strategy's execution, an alternative strategy is formed for each intermediate result.

3. During the course of a join execution sampling methods are used to estimate the size of the result. This estimate is then used to update the estimated delay of the current strategy and compare it with the delay of the alternative strategy. If the alternative strategy has a lower expected delay, correction takes place; the current strategy is aborted and the alternative strategy is adopted. Otherwise the original strategy is allowed to continue.

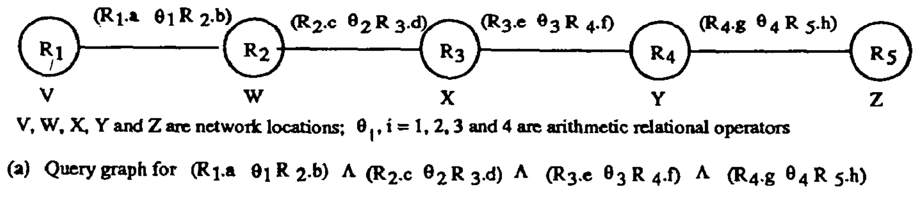
Note the difference between the size estimation performed in the initial Query Processing Strategy (QPS) phase and the proposed sampling when a join is being performed. The former uses simplifying assumptions, such as a uniform distribution of attribute values, independence of joining attributes and some simple statistical information such as a selectivity factor. The latter is based on samples of actual values from currently retrieved tuples, which should provide more accurate estimates. Increasing the sample size increases the confidence in accurate size prediction, but this is at the expense of higher overhead delay.

ALTERNATIVE STRATEGY

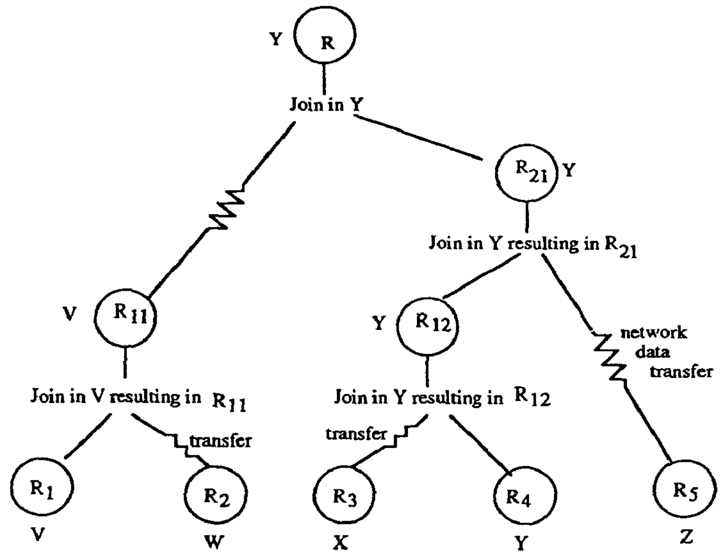
The proposed method is applicable to SPJ tree queries (sub-queries) which are processed by a join processing tactic. It is a simple method based on the following observation. Consider the situation in which sampling indicates that an intermediate result will be larger than expected. If, as a consequence, this results in a correction, an optimal corrective strategy tends to postpone the manipulation of the relation in question. The proposed method postpones a join which produces much larger than expected relation to the very end of the strategy.

The method is presented using query graphs. Consider a query in graph form in which vertices represent relations and edges represent the two-variable terms (Figure 1(a)). A strategy for this query is shown in a tree form in Figure 1(b). Under the stated assumptions, a join is identified by the two-variable term which it "processes". For example, a join of R_3 with R_4 "processes" the two-variable term ($R_3.e\ \theta_3\ R_4.f$).

Postponing a join to the end of a strategy implies, of course, that its operands are created first. Because of the assumption that the query is of a tree type, each operand is a result of a separate sub-query, i.e., each sub-query is represented by a sub-graph, such that the two sub-graphs are disconnected. In reference to Figure 1, for instance, postponement of the join to process the term "$R_3.e\ \theta_3\ R_4.f$" to the end of the strategy implies that its operands are produced by sub-queries shown in Figure 2(a) within the context of the original query. Clearly, for a strategy to process a tree query, the postponement of a join to

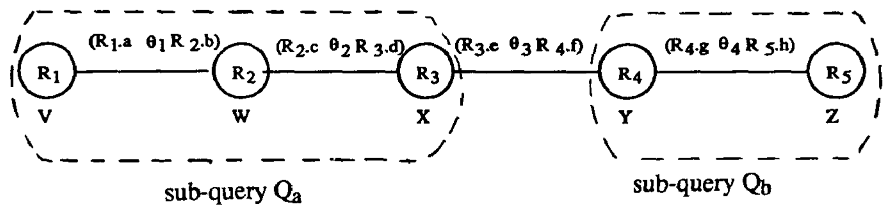

$(R_1.a\ \theta_1 R_2.b)$
$(R_2.c\ \theta_2 R_3.d)$
$(R_3.e\ \theta_3 R_4.f)$
$(R_4.g\ \theta_4 R_5.h)$
R_1
R_2
R_3
R_4
R_5
V
W
X
Y
Z
V, W, X, Y and Z are network locations; θ_i, i = 1, 2, 3 and 4 are arithmetic relational operators

(a) Query graph for $(R_1.a\ \theta_1 R_2.b) \wedge (R_2.c\ \theta_2 R_3.d) \wedge (R_3.e\ \theta_3 R_4.f) \wedge (R_4.g\ \theta_4 R_5.h)$

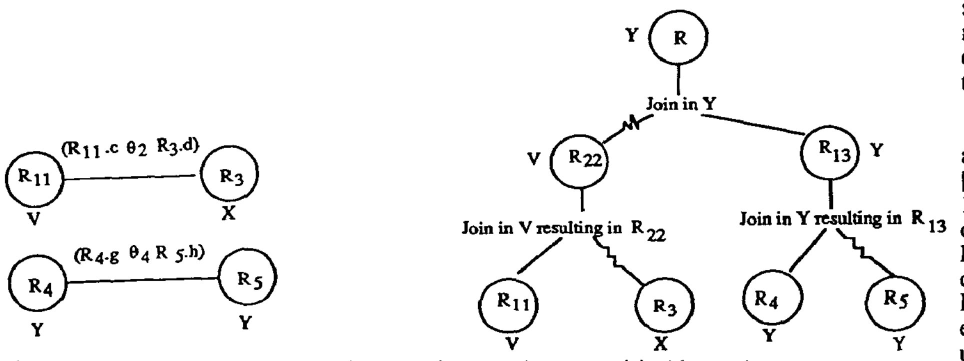


(b) Strategy represented by a tree

Figure 1 A query and a strategy to process it


$(R_1.a\ \theta_1 R_2.b)$
$(R_2.c\ \theta_2 R_3.d)$
$(R_3.e\ \theta_3 R_4.f)$
$(R_4.g\ \theta_4 R_5.h)$
R_1
R_2
R_3
R_4
R_5
V
W
X
Y
Z
sub-query Q_a
sub-query Q_b

(a) Distinct sub-queries to derive operands for the post-poned join


$(R_{11}.c\ \theta_2\ R_3.d)$
R_{11}
R_3
V
X
$(R_4.g\ \theta_4\ R_5.h)$
R_4
R_5
Y
Y
Y
R
Join in Y
V
R_{22}
R_{13}
Y
Join in V resulting in R_{22}
Join in Y resulting in R_{13}
R_{11}
R_3
R_4
R_5
V
X
Y
Y

(b) Sub-queries with completed intermediate results which were in progress at the time of decision

(c) Alternative strategy

Figure 2 Sub-queries and alternative strategies

the end of a strategy implies that its two operands are results of two distinct sub-queries which are mutually exclusive with reference to the relations of the original query. Moreover, the formulation of alternative strategies uses estimates of intermediate results which should already be formed when the decision to correct is made. With the exception of the sampled joins, operations in progress at the time of the decision are permitted to complete. Consequently, the sub-queries which produce the operands for the postponed join to process the term "$R_3.e\ \theta_3\ R_4.f$" are such that the join of R_1 with R_2 and the transfer of the relation R_5 from Z to Y are assumed to be completed as planned for the original strategy. The sub-queries and the alternative strategy are shown in figures 2(b) and 2(c), respectively.

Recall that an alternative strategy is formed for each of the intermediate results. The alternative strategy for the relation R_{12}, which is the result of the join of R_3 with R_4, consists of two sub-strategies to derive the operands of the final join, plus the final join. Each of the two sub-strategies is prepared in a background mode, that is, concurrently with the execution of the original strategy. In addition, an equation is prepared to determine the delay of the final join of the alternative strategy. It is used to quickly estimate the delay of the alternative strategy when the size of the result is estimated using sampling. A brief example in the sub-sequent section should clarify the above statements.

4. MODELED ENVIRONMENT

The alternative strategies described above are heuristic and hence sub-optimal. Modeling was used to compare the proposed method to other methods including those which use optimal strategies. This section describes the modeled distributed environment and determination of processing and data transfer delays.

Relations are assumed to be complete and unique as in modeling reported in [Sacco; 1984; Segev, 1984; Lafortune, 1986]; that is, there is no fragmentation or replication of data. Relations are located in distinct network locations. A query is assumed to be a tree Select-Project-Join (**SPJ**) query. Recall that each term has a selectivity factor which is used to determine the size of a join result.

In order to provide a broad base of varying types of queries (within the assumptions adopted of SPJ queries), all possible configurations of join graphs for queries involving between two and five joins (three and six relations) were determined. Only those queries with unique join graph shapes were considered. Fifty two queries were modeled such that the cardinalities of relations, ranging from ten tuples to one hundred thousand tuples, and the selectivities, ranging from 0.1 to 0.0001, were randomly determined. All relations were assumed to have equivalent average size per tuple (as in [Kerschberg, 1982]) of fifty bytes .

The network delays are directly proportional to the volume of transferred data. The unit cost of a network transfer is represented by a matrix in which each element represents the cost/delay to transfer a unit of data (byte) between two network locations. Two types of network environment comparable to those used in [Mackert, 1986a] were modeled; they differ in the data transfer rates: one corresponds to a long haul network with data transfer rates ranging from 2400 to 19200 bps, while the other corresponds to a local area network with an average data transfer rate of between 1 and 4 Mbps.

Join delays, which are assumed to be directly proportional to the volume of data accessed through secondary storage devices, are determined as in [Mackert, 1986a] on a 'per tuple' basis. The average join involves some combination of the following essential operations depending on the exact join method used:

(i) disk access;
(ii) internal transfer; and
(iii) comparison.

Different join processing methods vary in the amount of disk access and processing time they require. For simplicity, it is assumed that each tuple of the result and some combination of the tuples of the operands, as determined by the algorithmic complexity of the join method used, contribute a constant factor to the overall delay of executing the join. The constant factor is the time required to perform the essential operations and a disk access. Five machine instructions are assumed to suffice to join two tuples and execution proceeds at a rate of of 0.0004 msec/instructions (comparable to IBM 4381 CPU processing). The secondary storage I/O time is estimated as a sum of the average seek, latency and transfer times per page of data, such that each page is 2 Kbytes. The secondary storage device statistics were derived from an IBM 3380 disk with I/O time of 0.02348 seconds per page of data. A join is assumed to be executed using one of the following methods: Nested-loop with algorithmic complexity of $O(mn+r)$, (perfect) Hashing without misses with a complexity of $O(m+n+r)$ and Indexed method with a computational complexity of $O(n \log n + m \log n + r)$; m and n are cardinalities of the operand relations, and r is the cardinality of the join result. Hashing without misses, or perfect hashing, implies that there is no overflow of buckets. One relation is hashed into buckets. For each tuple of the other relation, hashing is used to find the bucket containing tuples for joining. For the index method, an index is first built on one (the smaller) of the operand relations with a cardinality of n. This index is then used to find appropriate tuples to be joined with each tuple of the other relation with the cardinality of m.

Due to the space limitation, the following report will concentrate on reporting the results of modeling the type of environment in which joins are executed using the indexed join method and data transfer delays are for a long-hauled network. Note that the indexed method is "representative" under the adopted assumptions: it is better than the nested-loop method, but worse than the hashing method without misses. Results of modeling the other join and data transfer methods will also be high-lighted.

5. MODELING METHOD AND NOTATION

Recall that as the proposed method prepares alternative strategies in which aborted joins are last; it is called Aborted Join Last (AJL) method. It shall be compared with the following methods:

Abort Join Optimal (AJO): This method is theoretically "optimal", but not realizable. It is similar to the proposed AJL method, but assumes perfect a priori knowledge of the intermediate result size just before the join execution commences. Without overhead delay, a new corrective strategy is formulated and executed for the remaining unprocessed portion of the strategy. The new strategy may, of course, be the same as the original one.

Complete Join Optimal (CJO): This is identical to the AJO method with the exception that the join is not aborted. Only when the join is completed is a new strategy formulated and instituted (without delay) for the remaining unprocessed portion of the strategy.

Static Optimal (SO): The strategy execution is static.

It should be noted that although a join which produces a larger-than-expected relation is aborted in the AJL and AJO methods, all other operations which are in progress at the time of this abort and correction are permitted to complete. This applies to all adaptive processing methods. [Bodorik, 1988c] has found that such an approach leads to lower delays in comparison to aborting all operations which are in progress at the time of the correction.

An optimizing algorithm which formulates a QPS estimates two parameters: size (cardinality) of relations produced by relational operations, and average delays due to relational processing and data transfer. Since it was shown in [Bodorik, 1988c] that the former are far more critical than the latter, only inaccuracies in size estimation are modeled. To find the effects of inaccuracies in estimating the size of intermediate results on the query response time, the size of intermediate results, one at a time, is increased by a certain factor. How the strategy's execution delay (query response time) is affected is determined under the assumption that all parameters, with the exception of the size of the one intermediate result under consideration, remain unchanged. The modeled strategy execution may be either SO (static) or adaptive with one of the AJL, AJO or CJO methods.

Consider a query i processed by the minimal response time strategy which forms M_i intermediate results. For each intermediate result, one at a time, its size is increased by a factor of y (multiplied by y). The strategy execution delay is then found assuming static processing and also adaptive processing with the previously described methods. The average increase in delays is found for each query and over all of the modeled queries. This is performed using various values for the factor y. In order to remove the effects of queries with high response time, what is actually measured is the factor by which the query response time is increased. Let

$T_{\alpha,\beta}\{i,j,y\}$...delay of query i such that the size of the j^{th} intermediate result of its strategy is increased by a factor y. Depending on the subscript α, the strategy execution is assumed to be either static or adaptive. The subscript β,

$0<\beta\leq 1$, which applies only to the proposed AJL method, denotes the fraction of the join used by the sampling method to determine the size of the join result:

α = "SO" ...the strategy execution is static
= "AJL"...the strategy execution is adaptive using the AJL method
= "AJO"...the strategy execution is adaptive using the AJO method
= "CJO"...the strategy execution is adaptive using the CJO method

β ... if α ="AJL", then β, $0 < \beta \leq 1$, is the fraction of the join used for sampling;
... NA for Not Applicable if $\alpha \neq$ "AJL"

$F_{\alpha,\beta}\{i,y\}$...the average increase (expressed as a factor) in the execution delay of the strategy for the query i, such that intermediate results have their sizes increased by a factor y. The type of processing depends on the subscripts α and β:

$$F_{\alpha,\beta}\{i,y\} = \sum_{j=1}^{M_i} (T_{\alpha,\beta}\{i,j,y\} \;/\; T_{\alpha,\beta}\{i,j,1\})\;/\;M_i$$

Note that $T_{\alpha,\beta}\{i,j,1\}$ is the strategy execution delay under the assumption that all estimates on which the strategy is based are correct and it therefore does not depend on whether the strategy execution is static or adaptive. Finally, average increases over all queries are defined:

$E_{\alpha,\beta}\{y\}$ ...denotes the average increase, expressed as a factor, in the execution delay over all queries i = 1, 2,...,N, where N is the number of queries:

$$E_{\alpha,\beta}\{y\} = \sum_{i=1}^{N} F_{\alpha,\beta}\{i,y\} \;/\; N$$

EXAMPLE

Consider the example of Figure 1. The query is shown in Figure 1(a), while its initially formulated, optimal strategy is shown in Figure 1(b) in a "tree" format. Since for the purposes of the example the timing considerations are important, the strategy is represented by a timing diagram in Figure 3(a). Consider now the join of relations R_3 with R_4 and assume that its result, the relation R_{12}, has cardinality (and therefore size) which is higher than estimated in the initial formulation. In the evaluation technique, this is simulated by multiplying the size of the relation R_{12} by a factor y. The query's delay is then determined depending on the method of processing as discussed below.

SO method: If static processing is assumed then the strategy execution delay is calculated using the "increased" size of the intermediate result under consideration (relation R_{12}). This implies that the execution delays of joins of R_3 with R_4, R_{12} with R_5, and R_{21} with R_{11} are also affected by this size increase. Had the relations R_{12} or R_{21} been transferred then these transfer delays would also have been affected. The join of R_3 with R_4 is affected because the join execution delay depends not only the size of its operands and the execution method, but also on the size of the relation it produces. The strategy's timing diagram as affected by this size increase is shown in Figure 3(b).

AJO method: **Before** the join of R_3 with R_4 commences, shown as time t_{AJO} in Figure 3(b), it is assumed that the actual size of the relation R_{12} becomes known. An optimal strategy is formulated and instituted for the unprocessed portion of the query without any overhead delay.

AJL method: The actual size of the relation R_{12} is predicted using sampling after completing the fraction β of the join of R_3 with R_4. Assume that this occurs at time t_{AJL} as shown in Figure 3(b). Using this predicted size of R_{12}, the execution delays of the current and the alternative strategies are estimated for the unexecuted portion of the query. If the alternative strategy is estimated to lead to a lower delay than the current strategy, correction takes place: the current strategy is aborted and the alternative one is instituted. Modeling is such that the alternative strategy delay includes the overhead sampling delay (the time period between the start of the join of R_3 with R_4, which happens to be t_{AJO}, and t_{AJL}). How the overhead delay to abort the current strategy and institute the alternative one affects the response time will be examined in a later section.

CJO method: **After** the completion of the join which produces the intermediate result R_{12}, an optimal strategy is formulated and instituted without any overhead delay for the unprocessed portion of the query. In Figure 3 this occurs at time t_{CJO}.

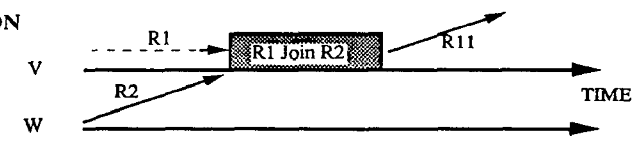
The original strategy may complete in any of the considered adaptive processing methods. If the corrective strategy is instituted, it applies to the "unexecuted/unprocessed portion of the query". Recall that for the AJO and CJO methods operations in progress at the time of the decision to correct are permitted to complete. This implies that since under the AJO method the decision to correct is made at time t_{AJO}, which is just before the commencement of the join of R_3 with R_4, the corrective strategy is formulated for the query as shown in Figure 4(a): the join of R_1 with R_2 is permitted to complete. Similarly, as the decision to correct is made by the CJO method at time t_{CJO}, the corrective strategy is formulated for the query as shown in Figure 4(b). Finally, consider the AJL method. With the exception of the aborted "sampled" join in question, all operations which are in progress at the time t_{AJL} when it is decided to correct are permitted to complete. Consequently, the alternative strategy is for the "unprocessed portion of the query", which in this example happens to be identical to the one obtained under the AJO method (Figure 4(a)).

6. EVALUATION

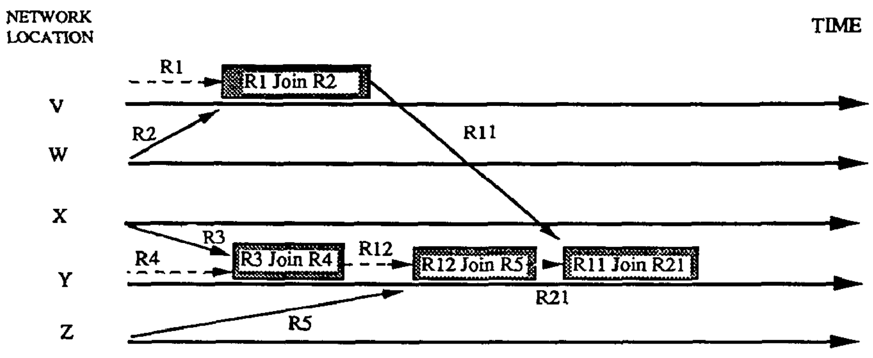
Figure 5 shows $E_{\alpha,\beta}\{y\}$ for various values of y and for each of the methods under consideration, i.e., for α being SO, AJL, AJO and CJO. For instance, if α = AJO, then $E_{\alpha,\beta}\{y\}$ shows the increase, expressed as a factor, in the average delay of strategies executed dynamically with the AJO method, when the actual size of intermediate results, one at a time, is y times higher than estimated. In Figure 5, β=0.2 when α = AJL; that is, 20% of a join is completed before sampling is used to determine the size of the join result and to decide whether or not to abort the join and institute the alternative strategy. The delay to execute the β = 0.2 fraction of the subsequently aborted join[1] is included as overhead in modeling the proposed AJL method.

[1]This delay (to complete β = 0.2 fraction of an aborted join) is directly proportional to the β fraction of the size of the join's operands and the resulting relation.

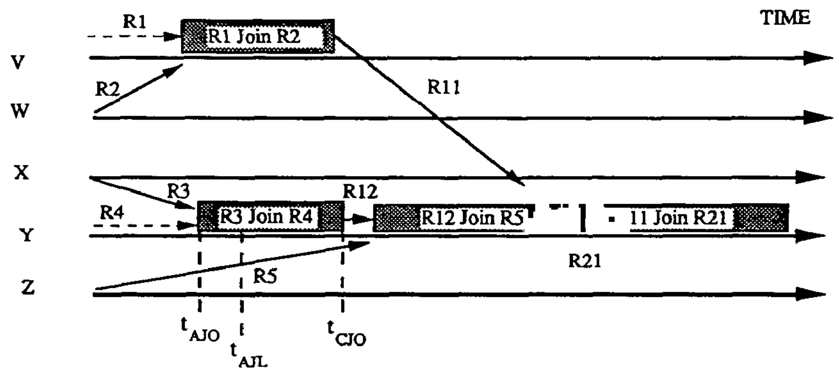
NOTATION



R2 is transferred from W to where it is joined with R1 (already located in V). The result of the join, the relation R11, is transferred immediately to another network location.

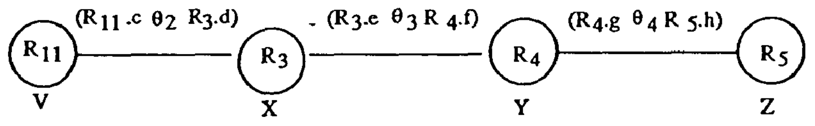


(a) Strategy timing diagram

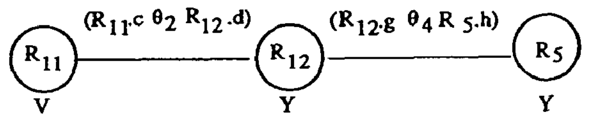


(b) Strategy timimg diagram with an increased size of the intermediate relation R12

Figure 3 The effect of an increase in the size of a join result on the strategy


$(R_{11}.c\ \theta_2\ R_3.d)$
$(R_3.e\ \theta_3\ R_4.f)$
$(R_4.g\ \theta_4\ R_5.h)$
R_{11}
R_3
R_4
R_5
V
X
Y
Z

(a) A query with completed join of R_1 with R_2


$(R_{11}.c\ \theta_2\ R_{12}.d)$
$(R_{12}.g\ \theta_4\ R_5.h)$
R_{11}
R_{12}
R_5
V
Y
Y

(b) A query with completed joins of R_1 with R_2 and R_3 with R_4

Figure 4 "Unexecuted/unprocessed" portions of the query

As expected, increase in delays of static strategies appears to be linear with the increases in inaccuracies in estimation which are represented by the values of y. This dependance follows directly from the assumptions made on the size of relations produced by joins. Secondly, the proposed AJL method is close to the AJO method. In other words, the alternative heuristic strategies lead to execution delays which are close to those of optimal corrective strategies produced by the AJO method. This is in spite of the fact that the sampling delays of the AJL method are included in overhead of correcting, while the overhead when deciding to correct is completely neglected in the AJO method. In the AJL method, sampling occurs first, that is, a β fraction of a join is completed before the size of the join result is determined through sampling and it is decided whether or not to correct and initiate the alternative strategy prepared in a background mode. If correction does take place, the overhead sampling delay is included in the delay of the alternative strategy.

Finally, for high inaccuracies, i.e., for large values of y, aborting the join which produces the much larger than estimated relation becomes crucial. Consider the CJO method. When a join produces a larger result than estimated, it is not aborted. A corrective strategy is assumed to be formulated and instituted without any overhead delay after this join is complete. Although this strategy is optimal for the remainder of the query, its average delay is greater for higher values of y than those of the AJO and AJL methods which do permit abort of joins producing larger than expected relation. If such a join is not aborted, the corrective strategy is forced to use the join's result which incurs higher delays than those of the AJO and AJL methods which abort the join in question and thus can postpone the creation of a large relation to the end of a corrective strategy. This is also confirmed by the AJL method when the β values are varied as is shown in Figure 6. It shows $E_{\alpha,\beta}(y)$ with α = AJL and several values of β in the range between zero and one. Increases in delays under the proposed AJL method are shown for various fractions of sampling as determined by β. Obviously, increasing the value of β also increases the confidence in estimating the size of the join's result and also the delay in correcting. Conversely, decreasing its value also decreases the confidence in correctly predicting the size of the join result [Vitter, 1985; Olken, 1986]. If $\beta = 1$, the join is permitted to complete and hence the cost of the alternative strategy is correctly determined; however, if correction does take place, the join execution time is wasted. If $\beta = 0.5$, the sampling method to determine the

y	4	8	12	16	20	24	28	32
$E_{SO,NA}\{y\}$	1.73	2.29	2.70	3.18	3.58	3.96	4.33	4.70
$E_{CJO,NA}\{y\}$	1.61	2.01	2.27	2.45	2.60	2.74	2.86	2.98
$E_{AJL,0.2}\{y\}$	1.63	2.12	2.36	2.47	2.53	2.60	2.66	2.71
$E_{AJO,NA}\{y\}$	1.51	1.82	1.98	2.05	2.10	2.14	2.17	2.20

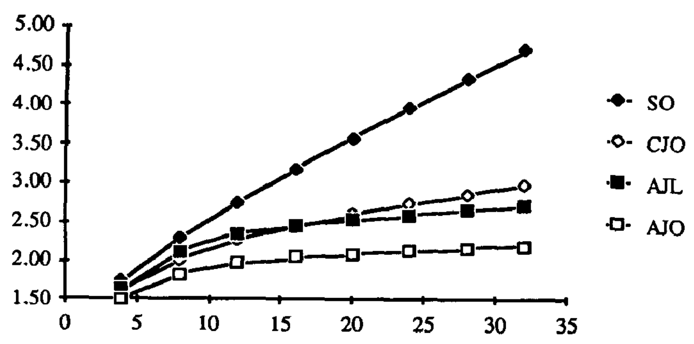


Figure 5 $E_{\alpha,\beta}\{y\}$ shown for various values of y; α = SO, CJO, AJL and AJO

y	4	8	12	16	20	24	28	32
$E_{AJO,NA}\{y\}$	1.51	1.82	1.98	2.05	2.10	2.14	2.17	2.20
$E_{AJL,0.2}\{y\}$	1.63	2.12	2.36	2.47	2.53	2.60	2.66	2.71
$E_{AJL,0.5}\{y\}$	1.63	2.12	2.37	2.49	2.56	2.62	2.69	2.74
$E_{AJL,0.75}\{y\}$	1.63	2.12	2.38	2.51	2.58	2.65	2.72	2.77
$E_{AJL,1.0}\{y\}$	1.63	2.12	2.40	2.52	2.61	2.68	2.74	2.80
$E_{SO,NA}\{y\}$	1.73	2.29	2.70	3.18	3.58	3.96	4.33	4.70

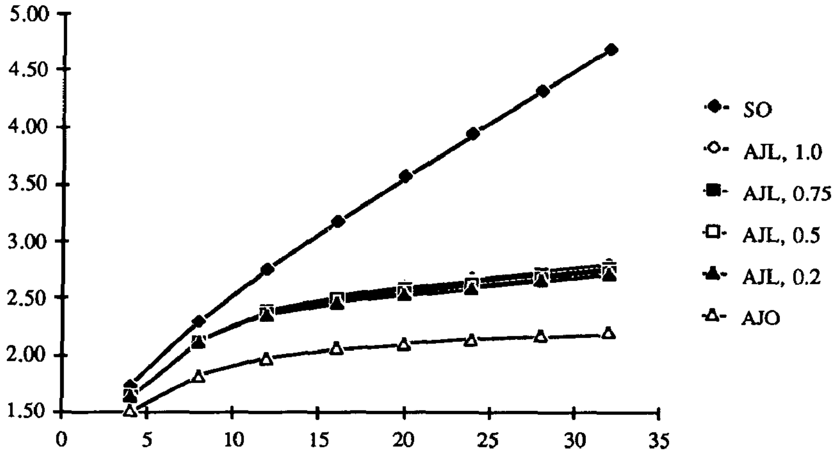


Figure 6 $E_{AJL,\beta}\{y\}$ shown for β = 0.2, 0.5, 0.7 and 1.0

size of the join result uses half the resulting relation and, consequently, should be highly accurate. However, again, if a correction does take place the join execution time (or β = 0.5 fraction there of) is wasted. As β gets smaller, wasted overhead delay in case of correction decreases, but the confidence in accurately predicting the size of the join result also decreases. Consider now the case when β = 1. Join execution completes and its size is used to determine whether the alternative strategy should be instituted. If correction takes place, the alternative strategy "aborts" this completed join, that is, it ignores the relation the join produced produced. Even if this is the case, the average delays are still lower than those of the CJO method. Figure 6 shows that sampling overhead delays are more than offset by benefits of correction even for high values of β.

The reported results are for the distributed environment characterized by parameters of a long-hauled network type and an index join processing method. Modeling of other types of environment, not reported herein due to the space limitation, produced similar results in that the AJL method lead to delays which were close to those of the optimal AJO method. For a local area network, shorter delays were observed due to the faster communication. In addition to the index join processing method, the nested-loop and perfect hashing without misses join execution methods were also modeled. As expected, a "fast" hashed join execution method gave lower delays than slower join methods. The nested-loop join execution method incurred delays which "dwarfed" all other delays [Pyra, 1988]. In such an environment the adaptive methods appeared to have been particularly crucial in reducing high execution delays due to inaccurate size estimation.

MONITORING AND CORRECTING OVERHEAD

With the exception of sampling delays which are inherent in the AJL method, overhead delays were neglected for the sake of simplicity. Their effect on the performance of the AJL method is now examined. For this purpose some assumptions on the monitoring and correcting delays are made. Delays due to the initial QPS formulation equally apply to all of the methods under consideration and are therefore neglected. The strategy execution and its potential correction are assumed to proceed according to the following simplified method adopted from [Bodorik, 1989]. Once the QPS is initially formulated and distributed by a master processor, its execution commences. Concurrently with its execution, the master processor creates and distributes alternative strategies to cohorts. Since formulation of alternative strategies is assumed to proceed in a background mode, that is concurrently with the execution of the original strategy, it does not contribute to overhead delay. It would be preferred for cohorts themselves to prepare these alternative strategies in order to distribute the load created by the strategy formulation/optimization process. For that purpose, however, they would have to have appropriate schemas and the strategy formulator.

When a processor executes a join, it uses sampling to predict the size of the join result and (re)calculates the delay of the current and alternative strategies. This is a relatively simple process as it does not involve optimization, but only a (relatively) simple calculation. The delay of an alternative strategy includes the overhead delay due to monitoring and correcting. If the alternative strategy has a lower delay, the current strategy is aborted and the alternative strategy is instituted for the "unprocessed portion of the query". It is assumed that this will induce a delay which is equivalent to transferring one fixed length, 2000 byte long message to all other processor. The message is assumed to contain the command to abort and also the alternative strategy itself. CPU processing associated with correction is assumed to be equivalent to the delay caused by accessing 100 data units (1024 byte long pages) in the secondary data storage devices (about 2.3 seconds).

The above assumptions are reasonable since an error free network is assumed and only one estimation inaccuracy is considered at a time. In a real system, however, correcting would be far more complex because processors would first have to validate the corrective strategy to ensure that it is consistent with the state of the strategy execution, including the case when more than one estimating errors are detected. One such method is proposed in [Pyra, 1988]. Alternatively, an additional step may be introduced in which the state of the execution is ascertained by the master processor which would then either validate the alternative strategy or formulate a new one [Bodorik, 1989]. It is for these reasons that generously high overhead delays were assumed for the purposes of modeling.

Let

$E_{\alpha=AJLov,\beta}\{y\}$ ... the average increase, expressed as a factor, in the execution delay over all queries i = 1, 2,...,N. Strategies are processed using the AJL method. The overhead corrective delays, represented by a transfer of a fixed length message to all processors and a fixed duration CPU processing delay, are included in the execution delays.

The effect of corrective overhead on the execution delays ($E_{\alpha=AJLov,\beta}\{y\}$) is shown in Figure 7. The static (SO), optimal (AJO) and the proposed (AJL) methods without overhead are also shown for the purposes of comparison. The sampling overhead is $\beta = 0.2$. As expected, overhead delays showed to be insignificant in an environment in which they are relatively small in comparison to the average strategy execution delays which was 40 seconds. They appeared to be significant in the environment which included both fast communication and a fast join processing method, that is, in an environment which included a LAN and a hashed join execution method. This was due to the fact that overhead delays were not negligible in comparison to the strategy execution delays [Pyra, 1988].

DISCUSSION

Modeling suggests that the proposed adaptive processing technique should be beneficial in dealing with the problems arising due to inaccuracies in size estimation. They may, however, become beneficial only when inaccuracies are relatively high. This depends primarily on the relative magnitudes of overhead and strategy execution delays which, in turn, depend on the parameters characterizing the distributed environment, in particular the join processing and data transfer methods. That inaccuracies in size estimation affect optimal strategies confirms results of modeling reported in [Epstein, 1980], but are in direct contradiction with the results reported in [Kumar, 1987] which reported that optimal strategies are very insensitive to inaccuracies in estimation. Since the report in [Kumar, 1987] is on experiments for a centralized DB while the report in [Epstein, 1980] and on this work is is for a DDB, it appears that the problems of inaccuracies in size estimations are negligible in a centralized DB but critical in a DDB. Further investigation is required to confirm whether or not this indeed is the case.

y	4	8	12	16	20	24	28	32
$E_{SO,NA}\{y\}$	1.73	2.29	2.70	3.18	3.58	3.96	4.33	4.70
$E_{AJLov,0.2}\{y\}$	1.65	2.13	2.43	2.54	2.61	2.68	2.74	2.80
$E_{AJL,0.2}\{y\}$	1.63	2.12	2.36	2.47	2.53	2.60	2.66	2.71
$E_{AJO,NA}\{y\}$	1.51	1.82	1.98	2.05	2.10	2.14	2.17	2.20

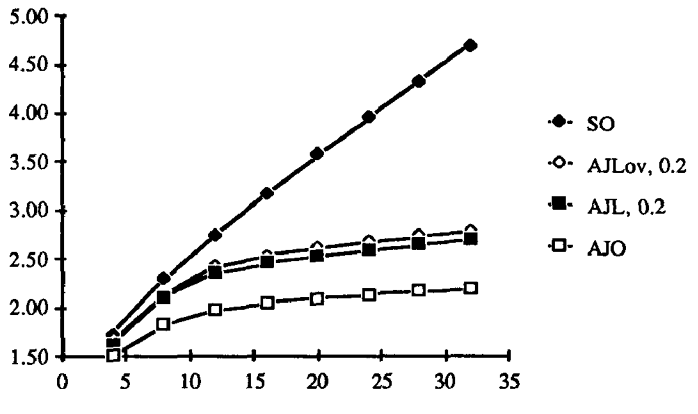


Figure 7 $E_{AJLov,0.2}\{y\}$ is shown for various values of y

7. SUMMARY AND CONCLUSIONS

This paper has proposed and evaluated a new method for deciding when to correct a distributed QPS. Once a strategy is formed and initiated, alternative strategies are prepared concurrently with the QPS execution. In addition, sampling methods are used to avoid the complete processing of relations whose results are much larger than initially estimated. Modelling has been used to demonstrate that alternative strategies, although heuristic, lead to delays which are close to those of optimal strategies. Furthermore, it has been shown that it is usually beneficial to abort an intermediate result/relation which is much larger than estimated. Although some expended work is wasted, this is more than offset by reducing delays by operating on smaller relations. The fixed overhead delays due to monitoring and correcting proved to be insignificant for the modeled environment.

Integration of the proposed method and the methods which can be utilized in monitoring and instituting a corrective strategy need to be investigated. Modelling was such that only one estimation error was considered at any one time. It remains to be investigated how multiple errors affect the query response time. Further problems, such as the stability of the methods employed and effects of pipelining, must be addressed before adaptive processing of queries can be shown to provide sufficient benefits to justify its widespread development and implementation.

REFERENCES

Apers P., Hevner A., Yao S.B., 1983; "Algorithms for Distributed Queries", IEEE TOSE, Vol. SE-9, No. 1, Jan. 1983, 57-68.

Bell D.A., Ling D.H.O. and McClean S., "Pragmatic Estimation of Join Sizes and Attribute Correlations", Proc. Fifth IEEE Data Engineering Conference, Los Angeles, CA, February 6-10, 1989, 76-84.

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

Bodorik P. and Riordon J.S., 1988a; "Distributed Query Processing Optimization Objectives", Proc. Fourth IEEE Data Engineering Conference, Los Angeles, CA, February 2-4, 1988, 320-329.

Bodorik P. and Riordon J.S., 1988b; "A Threshold Mechanism for Distributed Processing of Queries", Proc. of the ACM CSC '88 Conference, Atlanta, GA, Feb. 23-25, 1988, 616-625.

Bodorik P. and Riordon J.S., 1988c; "Evaluating Dynamic Processing of Distributed Queries", Proc. of the IEEE Int. Conference on Distributed Computing Systems, San Jose, California, June 13-17, 1988, 510-519.

Bodorik P. and Riordon J.S., 1988d; "Heuristic Algorithms for Distributed Query Processing", Proc. of the Int. Conf. on Databases in Parallel and Distributed Systems, Austin, Texas, Dec. 5-8, 144-153.

Bodorik P., Riordon J.S. and Jacob C., 1989; "Dynamic Distributed Query Processing Techniques", Proc. of the ACM CSC '89 Conference, Lousville, KY, February 21-23, 348-357.

Carey M., Livny M., Lu Hongjun, 1985; "Dynamic Task Allocation in a Distributed Database System", Proc. of the 1985 IEEE Conf. on Distributed Comp. Systems, 282-291.

Cellary W., Meyer D., 1980; "A Multi-query approach to Distributed Processing in a Relational Distributed Database Management System", Distributed Data Bases: Proc. Inter. Symp. on Distributed Data Bases, Edited by Delobel C., Litwin W., North Holland Publ. Co. March 1980.

Ceri S., Pelagati G., 1984; Distributed Databases: Principles and Systems, McGraw Hill, 1984.

Ceri S., Gottlob G., 1986; "Optimizing Joins between Two Partitioned Relations in Distributed Databases", Journal of Parallel and Distributed Computing, Vol. 3, 1986, 183-205.

Chao T., Egyhazy C.J., 1986; "Estimating Temporary File Sizes in Distributed Relational Database Systems:, Proc. IEEE Data Eng. Conf., 1986, 4-12.

Christodoulakis S., 1983; "Implications of Certain Assumptions in Database Performance Evaluation", ACM TODS, Vol. 9, No. 2, June 1984, pp. 173-186.

Chu W., Hurley P., 1982; "Optimal Query Processing for Distributed DB Systems", IEEE TOC, Vol. C-31, No. 9, Sept. 1982, 835-850.

Daniels et al., 1982; "An Introduction to Distributed Query Compilation in System R", In Distributed Data Bases, Schneider H.J., editor, North Holland, 1982, 247-290.

Egyhazy C., Triantis K., 1988; "A Query Processing Algorithm for Distributed Relational Database System", Computer Journal, Vol. 31, No. 1, 1988, 34-40.

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

Epstein R., Stonebraker M., 1980; "Analysis of Distributed DB Processing Strategies", 6th VLDB Conf., Montreal, QUE, Canada, 1980, 92-101.

Epstein R., et. al., 1986; "Distr. Query Processing in a Relational DB System", in The INGRES Papers: Anatomy of a Relational Database System, Stonebraker M., ed., Addison-Wesley, 1986, 197-214.

Gavish B., Segev A., 1986; "Set Query Optimization on Distr. Data Database Systems:, ACM TODS, Vol. 11, No. 3, 1986, 265-293.

Hevner A., Yao S.B., 1979; "Query Processing in Distributed Data Base Systems", IEEE TOSE, Vol. SE-5, No. 3, May 1979, 177-187.

Hevner A., 1980; "Query Processing in Distr. Data Base Systems", Ph.D. thesis, Univ. of Minnesota, 1980.

Hwang H-Y, Yu Y-T, 1987; "An Analytical Method for Estimating and Interpreting Query Time", Proc. VLDB, 1987, 347-358.

Ibaraki T., Kameda T., 1984; "On the Optimal Nesting Order for Computing N-relational Joins", ACM TODS, Vol. 9, No. 3, Sept. 1984, pp. 482-502.

Ijbema a., Blanken H., 1986; "Estimating Bucket Accesses: A Practical Approach", Proc. IEEE Int. Conf. on Data Engineering, 1986, 30-37.

Jhingran A., 1988; "A Performance Study of Query Optimization Algorithms on a Database System Supporting Procedures", Proc. of the Int. Conf. on VLDB, 1988, 88-99.

Kambayashi Y., 1985; "Processing Cyclic Queries:, in Query Processing in Distributed Data Base Systems, Edited by Kim, Reiner and Batory, Springer-Verlag, 1985, pp. 62-78.

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

Kim W., 1985; "Global Optimization of Relational Queries: A First Step", in Query Processing in Distributed Data Base Systems, Edited by Kim, Reiner and Batory, Springer-Verlag, 1985, pp. 207-216.

Kumar A., Stonebraker M., 1987; "The Effect of Join Selectivities on Optimal Nesting Order", SIGMOD RECORD, Vol. 16, No. 1, March 1987.

Lafortune S., Wong E., 1986; "A State Transition Model for Distributed Query Processing", ACM TODS, Vol. 11, No. 3, Sept. 86, 294-322.

Lamport L., 1978;
"Time, Clocks and Ordering of Events in a Distributed System", CACM, Vol. 21, No. 7, 1978.

Lohman G.M., et. al., 1985;
"Query Processing in R*", in Query Processing in Database Systems, Edited by Kim W., Reiner D., Batory D., Springer Verlag, 1985, pp. 31-47.

Mackert L.F., Lohman G.M., 1986a;
"R* Optimizer Validation and Performance Evaluation for Distributed Queries", Proc. VLDB, 1986, 149-159.

Mackert L.F., Lohman G.M., 1986b;
"R* Optimizer Validation and Performance Evaluation for Local Queries", Proc. ACM SIGMOD, 1986, 84-95.

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

Masuyama S., et al., 1987;
"Shortest Semi-join Schedule for a Local Area Distributed Database System", IEEE TOSE, Vol. SE-13, No. 5, May 1987, 602-606.

Nguyen, N.G., 1981;
"Distributed Query Management for a Local Network", Proc. 2nd Int. Conf. on Distributed Computing Systems, Paris, France, April 1981, 188-196.

Olken G., Rotem D., 1986;
"Simple Random Sampling from Relational Databases", Proc. 12th VLDB, Kyoto, August 1986, 160-169.

Otoo E.J., et al., 1987;
"Improving Semi-Join Evaluation in Distributed Query Processing", Proc. of the IEEE Conf. on Distr. Comp. Systems, 1987, 554-561.

Ounegbe E., Rahimi S., Hevner A., 1983;
"Local Query Translation and Optimization in a Distributed System", Proc. NCC, July 1983, pp. 229-239.

Perizzo W., 1984;
"A Method for Processing Distr. Database Queries", IEEE TOSE, Vol. SE-10, No. 4, July 1984, 466-471.

Pyra J., 1988;
"Dynamic Query Processing in Distributed DB Systems", M.Comp.Sci. Thesis, Technical University of Nova Scotia, Halifax, Nova Scotia, Canada, December 1988.

Reuter A., 1986;
"Load Control and Load Balancing in a Shared Database Management System", Proc. of the Int. Conf. on Data Engineering, Los Angeles, CA, USA, Feb. 1986, pp. 188-197.

Sacco G.M., 1984;
"Distributed Query Evaluation in Local Area Networks", Proc. of the Int. Conf. on Data Engineering, 1984, 510-516.

Segev A., 1984;
"Optimizing Fragmented 2-Way Joins", Proc. of the Int. Conf. on Distributed Computing Systems, 1984, 378-388.

Selinger P.G., Adiba M., 1980;
"Access Path Selection in Distributed Data Base Management Systems", Proc. of the First Int. Conf. on Data Bases, Aberdeen, 1980.

Sellis T.K., 1988;
"Multiple-Query Optimization", ACM TODS, Vol. 13, No. 1, March 1988, 23-52.

Su S.Y.W., et al., 1986;
"A Distributed Query Processing Strategy Using Decomposition, Pipelining and Intermediate Results Sharing Techniques", Proc. of the Int. Conf. on Data Engineering, Los Angeles, CA, USA, Feb. 1986, pp. 94-102.

Vander Zanden B.T., et al., 1986;
"Estimating Block Access when Attributes are Correlated", Proc of the Int. Conf. on VLDB, 1986, 119-127.

Vitter J.S., 1985;
"Random Sampling with a Reservoir", ACM TMS, Vol. 11, No. 1, March 1985, 37-57.

Wong E., 1982;
"A Statistical Approach to Incomplete Information in Database Systems", ACM TODS, Vol. 7, No. 3, Sept. 1982, 470-488.

Yang H.Z., Larsen P.-A., 1987;
"Query Transformation for PSJ-queries", Proc. Int. Conf. on VLDB, 1987, 245-254.

Yu C., Lin Y.C., 1982;
"Some Estimation Problems in Distributed Query Processing", Proc. IEEE Data Eng. Conf., 1982, 13-19.

Yu C., et. al., 1983;
"On the design of a Distr. Query Processing Strategy", Proc. ACM SIGMOD Conf., May 1983, pp. 30-39.

Yu C.T., 1985;
"Distributed Database Query Processing", in Query Processing in Database Systems, Edited by Kim W., Reiner D., Batory D., Springer Verlag, 1985, pp. 48-61.

Yu C., et. al., 1986;
"Adaptive Techniques for Distributed Query Optimization", Proc. of the Int. Conf. on Data Engineering, Los Angeles, CA, USA, Feb. 1986, pp. 86-93.

Yu C., et al., 1987;
"Algorithms to Process Distributed Queries in Fast Local Networks", IEEE TOC, Vol. C-36, No. 10, Oct. 1987, 1153-1163.