1978

Optimization of Data Access in Distributed Systems

Alan R. Hevner

S. Bing Yao

Report Number:
78-281

(DRAFT: July 1978)

OPTIMIZATION OF DATA ACCESS IN DISTRIBUTED SYSTEMS +

Alan R. Hevner and S. Bing Yao*
Computer Science Department
Purdue University
West LaFayette, Indiana 47907

TR-281

Abstract

The application of computer network technology to database systems has produced much interest in distributed database systems. Query processing on a distributed system is seen to be quite a different problem from query processing on a centralized system. A query requiring data from two or more distinct nodes in the network must be solved by a distribution strategy that consists of a schedule of local data processing and data transmissions. Two cost measures, total time and response time, are used to judge the quality of a given distribution strategy. Methods that find efficient distribution strategies for queries are proposed and analyzed. Algorithms that embody simple distribution tactics are shown to be optimal in the sense of minimizing response time and total time for a special class of queries. A method is proposed to extend the optimal algorithms to derive efficient distribution strategies for general query processing in distributed database systems.

Key Words and Phrases: Query Processing, Distributed Database Systems, Computer Network, Relational Data Model.

CR Categories: 3.50, 3.70, 3.74, 4.33.

---

* Present Address: New York University, 600 Tisch Hall, 40 W. 4th Street, New York, New York 10003.

+ This research is supported by the National Science Foundation under Grant Number MCS76-16604.

1. Introduction

Distributed database systems are emerging as a natural application of the rapidly advancing field of computer network technology to data processing systems. Distributed systems offer several attractive performance advantages over conventional centralized systems. These advantages include increased system reliability, faster, easier access to distributed data, and the potential for modular implementation and upgrading of the system by adding new network stations [10].

From the viewpoint of efficient data management, organizations are finding distributed systems attractive. As organizations grow in size and complexity, the geographic locations of the origin and use of data have become increasingly dispersed. Distributed end users have become more sophisticated in their information needs. Distributing organizational data and the data management capabilities to the dispersed end user locations answers an organization's important data availability needs. Rapid and reliable data availability is critical to an organization's decision making responsibilities [4].

The relative inexpense of mini- and micro- computers to serve as network nodes has made the idea of a distributed database system a practical consideration to many organizations [3]. Still, however, many problems remain to be solved before distributed database systems receive widespread use. Several recent papers have surveyed the current and future research effort in distributed data management [10],[7]. Some of the problem areas for research include efficient distributed data access, consistent data update synchronization, network failure resiliency, and effective decentralized control of distributed database functions.

This paper concentrates on the problem of efficiently accessing data

in a distributed system through the use of non-procedural queries. Accessing data that is stored at separate nodes in a distributed system differs in two important ways from accessing data on a centralized system: 1. Required data transmission via communication lines between nodes introduces substantial time delays in the system; and 2. An advantage of a distributed system is the potential for parallel data processing and data transmissions. An efficient strategy for accessing data must take into account these differences. We will see in Appendix A that data access strategies for centralized systems may perform poorly in a distributed system because of these differences.

In order to process a non-procedural query on a distributed database, a wide range of feasible data access strategies exists. The time costs of these feasible strategies vary greatly. The objective of finding the most efficient, or least costly, data access strategy can be viewed as an optimization problem over an extremely large search space of feasible solutions. Previous research in this area has utilized classical optimization search techniques to find efficient data access strategies. Wong [12] has proposed an algorithm that is currently being implemented in the SDD-1 system [6]. An initial feasible strategy is selected. Then a standard 'hill-climbing' optimization technique recursively finds lower cost strategies until no more cost improvements can be discovered. The resultant data access strategy is a local minimum cost solution in the search space.

Other data access methods extend centralized query tactics to a distributed environment in order to find feasible data access strategies on the network [8], [11]. Once an initial data access strategy is found, no further optimization is done.

In contrast to using search techniques, we develop algorithms that directly produce provably optimal data access strategies for a class of queries. These optimal algorithms can be extended to find efficient data access strategies for general queries. In general algorithms the straightforward tactics found optimal for certain queries can be used in place of the sometimes costly 'hill-climbing' optimization technique.

2. Distributed Systems

We consider a network of inter-connected computers. Each computer, known as a node in the network, has a processing capability and a data storage capacity. Each node can transmit data to other nodes by means of communication links. We assume that the data transfer rates between nodes are significantly slower than the data transfer rates between each computer and its data storage.

A distributed system exists on a computer network when the data stored at multiple nodes are interrelated or if a transaction at one node requires access to data stored at another node [2]. We assume that each node contains a distributed database management system (DDBMS) and a possibly redundant portion of the database. The unit of data distribution is a relation - a two dimensional table in which each row is a record, or tuple, of the same type [5]. Each column of the table consists of a set of values that is called a domain of an attribute of the relation. We assume that the physical distribution of relations among nodes is given. The DDBMS will maintain system directories so that each transaction will receive a non-redundant, consistent mapping of its required data. The problems of maintaining consistent copies of redundant data and up-to-date system directories are discussed in [1] and [9].

The distributed system described above can be characterized by the following parameters.

N - Number of nodes in the system.

M - Number of unique relations in the database.

For each relation R_i, i = 1, ..., M:

n_i - Number of records.

a_i - Number of domains.

For each domain d_{ij}, j = 1, ..., a_i of relation R_i:

v_{ij} - Number of possible values the attribute of the domain can have.

u_{ij} - Number of values the domain currently holds.

p_{ij} - Selectivity. $p_{ij} = u_{ij}/v_{ij}$, where $0 < p_{ij} \leq 1$.

w_{ij} - Size of each data item.

For each relation R_i, the size is defined as $s_i = n_i * \sum_{j=1}^{a_i} w_{ij}$.

A relational query performs the operations RESTRICTION, PROJECTION, and JOIN in order to retrieve data [5]. An update (i.e. a deletion, insertion or modification of data) may be viewed as a data retrieval followed by the writing of the update to the database. For simplicity, we will use a graphic notation to represent a query. Consider the following example.

Example 1: Given the relations

PRODUCT (PROD#, PNAME, QOH)

ORDER (ORD#, DATE, TOTCOST)

PROD-ORD (PROD#, ORD#, UNITS)

Consider the query represented by Figure 1.

This query retrieves all product number and order number pairs, where the product has less than 1000 units on hand and the order was submitted before 1 September 1978. An equivalent QUEL expression [16] for this query is:

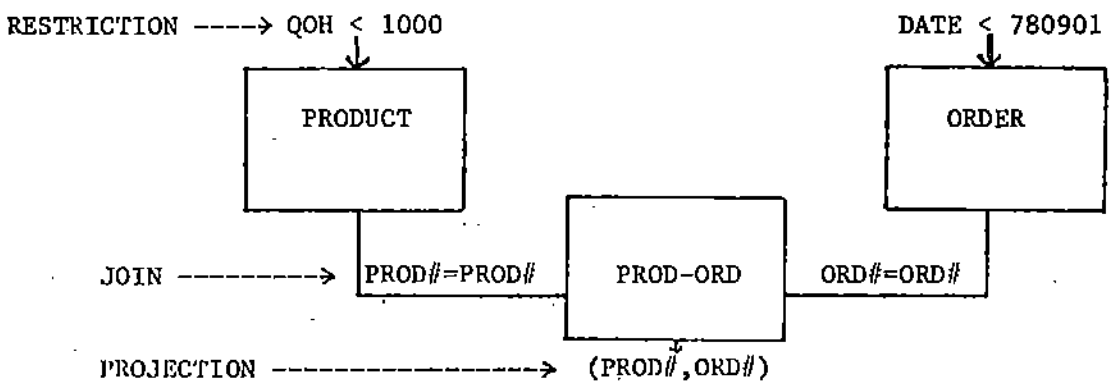


Figure 1 Graphic Representation of a Query

```
RANGE OF P IS PRODUCT
RANGE OF O IS ORDER
RANGE OF X IS PROD-ORD
RETRIEVE INTO RESULT (X.PROD#, X.ORD#)
WHERE (O.DATE < 780901) AND (O.ORD# = X.ORD#)
AND (P.QOH < 1000) AND (P.PROD# = X.PROD#)
```

The operations in a query may be performed in many orders. The objective of a query processing algorithm is to find the most beneficial order in which to perform the query operations [14], [15]. A query on a distributed system may require the access of data that is located at separate nodes in the network (i.e. if each of the relations in Example 1 were located at separate nodes). To process such a distributed query, the following information must be gathered by the DDBMS during query analysis.

1. The locations of the relations accessed by the query.

2. The required domains of these relations (designated as joining domains, restricting domains, and output domains). In Example 1, P.QOH and O.DATE are restricting domains, O.ORD# and P.PROD# are joining domains, and X.ORD# and X.PROD# are joining and output domains. The presence of a join between two relations at separate nodes indicates that data transmission is required in order to derive the query response.

3. A result node for the query is determined. A constraint of all feasible data access strategies is that the query response must end up at

the result node.

Query processing involving data at a single node is termed local processing. The effect of local processing is to reduce the amount of data that needs further processing. PROJECT eliminates unneeded domains from relations. RESTRICT selects rows of a relation that satisfy specified data conditions. JOIN combines relations and in the process eliminates rows whose domain values do not match between relations. This operation is also known as a JOIN RESTRICTION since, in essence, each relation is being restricted by the other. Assume a query has a selectivity of q on domain $d_{k\ell}$. After the local restriction is performed the parameters of relation R_k are changed as follows.

i) $n_k \leftarrow n_k * q$,

ii) $s_k \leftarrow s_k * q$,

iii) $p_{k\ell} \leftarrow p_{k\ell} * q$.

A join between domains $d_{k\ell}$ and d_{ij} produces a pair of join restrictions. These restrictions result in the same parameter changes on both relations where the selectivity parameter q becomes p_{ij} for the restriction on relation R_k and $p_{k\ell}$ for the restriction on R_i.

For distributed queries data transmission must be used to bring dispersed data together at single nodes in order to allow local processing. The distributed query problem then, is one of transforming a distributed query into a series of data transmissions and local data processing. This process is called distribution [12].

In any distribution strategy all possible initial local processing should be done first. This step processes all restrictions and intra-nodal join restrictions in the query. The inter-nodal joining domains and the output domains for each relation are projected. After this

initial local processing the state of the system is defined by the following parameters:

m - Number of relations in the remaining query

α_i - Number of domains in relation R_i, i = 1, ..., m.

β_i - Number of inter-nodal joining domains in relation R_i.

The reduced size of each relation R_i is $s_i = n_i * \sum_{j=1}^{\alpha_i} w_{ij}$, where the projected domains can be renumbered to be the first α_i domains. Between the initial local processing and the final local processing at the result node, the intermediate sequence of data transmissions and local processing is optimized by minimizing an appropriate cost function.

3. Criteria for Optimization

We define our cost measure in units of time. The data transmission cost between any two nodes is defined as a linear function $C(X) = c_0 + c_1X$, where X is the amount of data transmitted. The constant c_0 represents an initial start-up cost for each separate transmission. An important property of C(X) is that if $X \leq Y$, then $C(X) \leq C(Y)$.

In light of this cost assumption, it is clear that minimizing the cost of a data transmission is equivalent to minimizing the amount of data transmitted. This assumption not only simplifies analysis but also can be justified by recognizing that advances in communication technology are rapidly eliminating distance and even physical connection as essential cost considerations (e.g. satellite communications).

A detailed analysis of the cost of a distribution strategy would include consideration of data transmission costs and local processing costs. However, on almost all networks data transmissions between nodes

are significantly slower than the data movements and processing within local nodes. Therefore, our second cost assumption is that the data transmission costs so dominate the cost of a distribution strategy that local processing costs are considered to be insignificant.

In a distribution strategy each required relation has associated with it a <u>schedule</u>. A schedule is defined as the pattern of sequential and parallel data transmissions made in order to reduce the size of the relation before its data is transmitted to the result node. The complete strategy consists of m parallel schedules. In order to visualize clearly the structure and interaction of the schedules in a strategy we adopt a graphic notation in which each data transmission is represented by a horizontal line connecting local processiong steps (e.g. |—— C(X) ——|). The length of the line corresponds to the transmission time, C(X). This graphic notation allows us to recognize synchronization between different schedules.

To illustrate the graphic notation let us consider the cost graph of a distribution strategy shown in Figure 2. Consider the schedule for the relation R_2 in the cost graph. Data is transmitted in parallel from R_1 and R_3 to R_4. Relation R_4 is reduced in size and then data from it is transmitted to R_2. Relation R_2 is reduced in size and transmitted to the result node.

In a distribution strategy it is possible to combine two schedules into one schedule. This occurs when the entire data of one relation is transmitted to another relation on a joining path. When the relations are joined, the second relation acquires all of the required data of the first. Thus the first relation need not be transmitted to the result node and its schedule can be eliminated from the distribution strategy. For

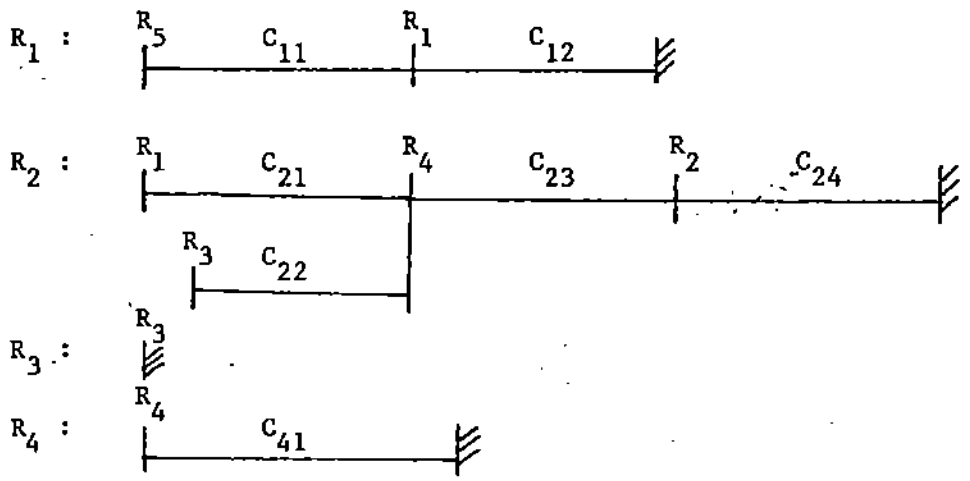


Figure 2. Example Cost Graph of a Distribution Strategy
Each relation is originally at a separate node.
R_3 is at the result node.

example, in Figure 2, relation R_5 is transmitted completely to relation R_1. After a join, relation R_1 contains the required data of relation R_5 and is transmitted to the result node. Relation R_5 need not be transmitted to the result node and its separate schedule is eliminated from the cost graph.

We define the schedule response time r_i for relation R_i as the time from the start of the schedule until relation R_i is received at the result node. For the R_2 schedule, $r_2 = c_{21} + c_{23} + c_{24}$. The time c_{22} is not included because it occurs in parallel with time c_{21}. We define schedule total time t_i for relation R_i as the sum of all times in the schedule. Thus, for relation R_2, $t_2 = c_{21} + c_{22} + c_{23} + c_{24}$. Many feasible schedules may exist for a relation R_i. We define the minimal response time of a schedule for relation R_i as $\hat{r}_i = \min(r_i)$, where the minimization ranges over all possible schedules.

In this paper we analyze the optimization of distribution strategies under two different cost objectives: the minimization of response time, and the minimization of total time.

i) Response time r. The cost objective is to minimize the time from the start of data transmission after initial processing until all required data is received at the result node in order to do final processing. The response time of a distribution strategy is given by the maximum length schedule. Therefore, $r = \max(r_i)$, $i = 1, \ldots, m$. The minimum response time $\hat{r}$ can be defined as $\hat{r} = \max(\hat{r}_i)$, $i = 1, \ldots, m$.

ii) Total time t. The cost objective is to minimize the sum of all schedule total times. The total time of a distribution strategy is $t = \sum_{i=1}^{m} t_i$ and the minimum total time $\hat{t}$ is $\hat{t} = \min(t)$ over all feasible distribution strategies.

For the cost graph of Figure 2, response time = $r = r_2 = c_{21} + c_{23} + c_{24}$ and total time = $t = t_1 + t_2 + t_4 = c_{11} + c_{12} + c_{21} + c_{22} + c_{23} + c_{24} + c_{41}$.

The choice of cost objective depends upon the distributed system. In a lightly loaded system with few queueing delays, response time minimization would be preferable. In a more heavily loaded system queuing delays may make total time minimization the better objective. Optimization of distribution strategies for both cost objectives is analyzed in the next section.

4. Optimization of Distributed Query Processing

A distribution algorithm is an algorithm that derives a distribution strategy for a given query. To design effective distribution algorithms for 'multi-variable' queries (queries involving multiple relations) on a distributed database, it seems natural to try to apply query processing strategies used on centralized databases. Wong and Youseffi [13] describe a general decomposition algorithm that processes multi-variable queries. The two basic tactics of the algorithm are: 1. Reduction - breaking off components of the query which are "joined" with other components by a single variable; and 2. Tuple Substitution - substituting for one of the variables a tuple (record) at a time. The authors demonstrate that these tactics can produce an efficient query processing strategy when good choices of reduction and substitution variables are made. The application of the tactics of reduction and tuple substitution to a distributed query, however,

will often lead to obviously inefficient distribution strategies. An example to illustrate the deficiencies of these tactics in a distributed system is given in Appendix A.

If tuple substitution and reduction are not appropriate for distributed query processing, what then are the basic tactics that can be used to find an efficient distribution strategy? We will show three simple tactics that take into consideration the unique aspects of a distributed system. It will be shown that these tactics provide a foundation for efficient distribution strategies.

An obvious distribution tactic is the transmission of all the required relations directly to the result node where the remaining local processing can be completed. This is called the initial feasible solution. Links which connect the nodes containing required relations with the result node are called destination paths. Most distribution algorithms use the initial feasible solution as a starting solution from which to find more efficient distribution strategies.

An inter-nodal join between two joining domains in the query defines a joining path in the network. In a distribution strategy the objective of transmitting data on joining paths is to transmit the least amount of data to cause the greatest processing advantage (i.e. size reduction) at the receiving node. As we search for cost beneficial data transmissions, the order in which the potential transmissions are checked and added to the distribution strategy is of critical importance. The distribution tactic that first checks data transmissions from small relations to larger relations is called the small to large tactic.

The third basic tactic is to make the most advantageous use of parallelism on the network. When a distribution algorithm minimizes

response time as its cost objective the inclusion of operations that occur in parallel with existing operations adds no additional response time to the strategy. Parallelism is emphasized when query response time is minimized.

The following example illustrates the use of these three distribution tactics to find a distribution strategy.

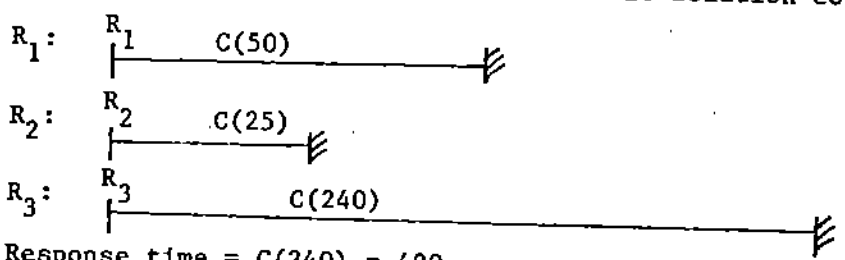
Example 2:

Assume a query requires data from three relations R_1, R_2, and R_3, located at separate nodes. After initial processing, the size and selectivity parameters are:

$(s_1,p_1) = (50,\frac{1}{2})$, $(s_2,p_2) = (25,\frac{1}{4})$, and

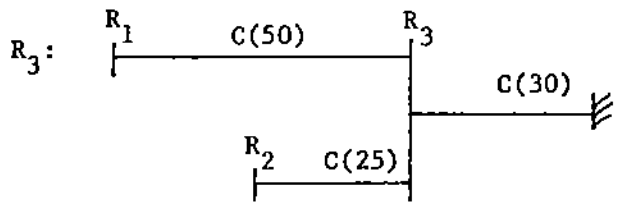
$(s_3,p_3) = (240,1)$.

Assume $C(X) = 10 + 2 * X$, and response time is to be minimized. Let the result node be a separate node. The initial feasible solution cost graph is:



Response time = C(240) = 490.
Total time = C(25) + C(50) + C(240) = 60 + 110 + 490 = 660.

Using the 'small to large' tactic, we test the cost benefit of transmitting the 'small' relations R_1 and R_2 to the 'large' relation R_3. We find that both transmissions are cost beneficial and can be done in parallel to reduce the response time of the strategy. The resulting distribution strategy can be seen to be optimal in terms of minimizing the query's response time.



Response time = C(50) + C(30) = 110 + 70 = 180.
Total time = C(25) + C(50) + C(30) = 60 + 110 + 70 = 240.

The rest of this section describes algorithms that derive distribution strategies for distributed queries. The three basic tactics are used in these distribution algorithms.

The optimization of distribution strategies is performed on a class of distributed queries called simple queries. A simple query is defined such that after initial local processing each relation in the query contains only one common joining domain. Thus, $\alpha_i = \beta_i = 1$ for all R_i, i=1,...,m.

A simple query introduces the following distribution considerations. The initial local processing forms the required data at each node into one relation. This is possible since all required relations have a common joining domain. Each required relation will be transmitted as a unit in a distribution strategy that will eventually transmit all required data to the result node.

4.1. Minimizing Response Time

To minimize the response time of distribution strategies for simple queries let us consider a distribution algorithm (Algorithm C) which is briefly described as follows. For cost comparison puposes, the starting distribution strategy is assumed to be the initial feasible solution. The algorithm searches for cost beneficial data transfers in the current system state. The state of the system is given by the size, s_i, selectivity, p_i, and schedule response time, r_i, of each relation R_i. A cost beneficial transmission to reduce response time is defined as any data transmission to relation R_i that reduces r_i in the current system state. Algorithm C makes use of a relation ordering, $R_1,\ldots,R_m$, and the 'small to large' tactic to look for cost beneficial data transmissions. All relations R_j, where $j < i$, are checked for potential data transmission to R_i. The data transmission that causes the greatest reduction in r_i is integrated into the distribution strategy. For the data transmission from relation R_j to R_i, Algorithm C transmits the selectivity of all relations R_k, $k < j$ to R_i (i.e. the accumulated selectivity of R_j is $\prod_{k=1}^{j} p_k$). This is an application of the 'parallel' tactic in which all relations R_k can be transmitted to R_i in parallel with R_j for little or no additional response time. The parallel data transfer from R_k to R_i is added to the distribution strategy if R_k does not already appear in the schedule of R_i and if $p_k \neq 1$.

Algorithm C is detailed in Appendix B. The following example provides a numerical illustration of its use.

Example 3:

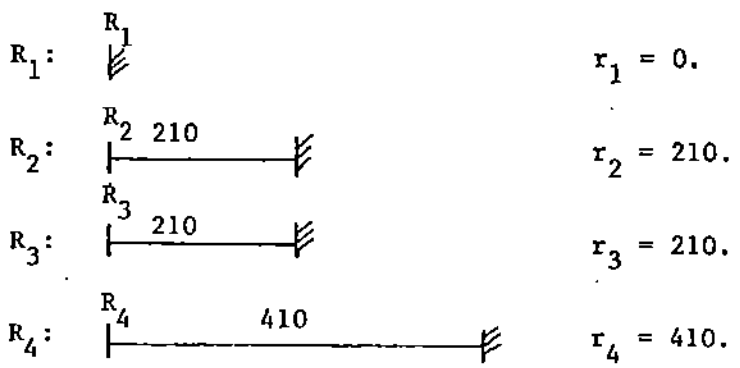
Assume a given query requires data from four relations at four separate

nodes. After initial processing the size and selectivity values are:

	size(s_i)	selectivity(p_i)
R_1	100	¼
R_2	200	½
R_3	200	½
R_4	400	1

Assume R_1 is at the result node and let transmission time be $C(X) = 10 + X$.

The cost graph of the initial feasible solution is:



Response time = r = 410.
Total time = t = 830.

Algorithm C finds $\hat{r}_i$, the minimum data transmission time for relation R_i, in the relation order R_2, R_3, and R_4.

<u>R_2 response time reduction</u>:

Transmit R_1 to R_2: $r_2' = 110 + C(¼ * 200) = 110 + 60 = 170$.

Since $170 < 210 = r_2$, the transmission from R_1 to R_2 is integrated into the strategy. Let $\hat{r}_2 = 170$.

<u>R_3 response time reduction</u>:

Transmit R_2 to R_3: $r_3' = 170 + C(1/8 * 200) = 170 + 35 = 205$.

Transmit R_1 to R_3: $r_3' = 110 + C(¼ * 200) = 110 + 60 = 170$.

Since $170 < 205 < 210 = r_3$, the data transmission R_1 to R_3 is integrated

into the strategy. Let $\hat{r}_3 = 170$.

R_4 response time reduction:

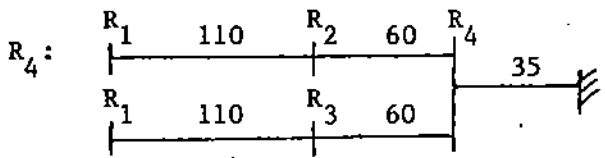
Transmit R_3 to R_4: $r_4' = 170 + C(1/16 * 400) = 170 + 35 = 205.$

Transmit R_2 to R_4: $r_4' = 170 + C(1/8 * 400) = 170 + 60 = 230.$

Transmit R_1 to R_4: $r_4' = 110 + C(\frac{1}{4} * 400) = 110 + 110 = 220.$

Since $205 < 220 < 230 < 410 = r_4$, the data transmission R_3 to R_4 is integrated into the strategy. The cost benefit of transmitting R_3 to R_4 includes the size reduction of R_4 by the selectivity $p_2 = \frac{1}{2}$. The parallel transmission of relation R_2 to R_4 is thus added to the strategy.

The final cost graph is:



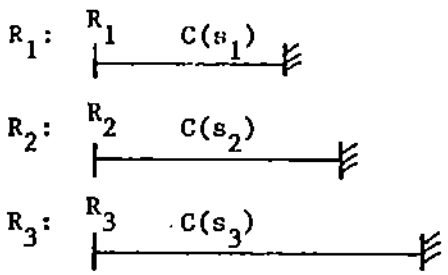
$\hat{r}_4 = 205.$

Response time = $\hat{r} = 205.$
Total time = $t = 375.$

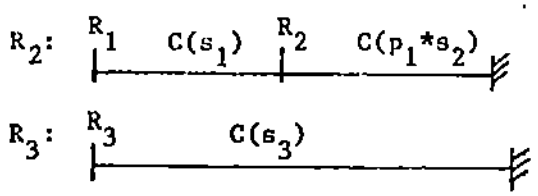
Algorithm C derives a distribution strategy for a simple query efficiently. The heart of the algorithm contains a double loop in which for each relation R_i, i=1 to m, all relations R_j, j=1 to i-1, are checked for cost beneficial transmission. Thus, the algorithm requires

$\sum_{i=1}^{m}(i-1) = \frac{m(m-1)}{2}$ cost calculations and comparisons. The complexity of Algorithm C is of order $O(\frac{m^2}{2})$ where m is the number of required relations in the query.

For any distributed query there may exist many distribution strategies that have an equivalent minimum response time. For example the following cost graphs demonstrate different feasible strategies for a simple query requiring data from three relations.



Response time = $C(s_3)$.
Total time = $C(s_1) + C(s_2) + C(s_3)$.



Response time = $C(s_3)$.
Total time = $C(s_1) + C(p_1*s_2) + C(s_3)$.

If $C(s_3)$ was the minimum response time, both strategies would have minimum response time for the given query.

We will show that Algorithm C finds a distribution strategy that is optimal in the sense of having a minimum response time. We make no claim on the uniqueness of the optimal strategy. Consider first the following Lemma.

Lemma 1:

For a simple query if, after initial processing, the required relations $R_1,\ldots,R_m$ are ordered so that $s_1 \leq s_2 \leq \ldots \leq s_m$ then $\hat{r}_1 \leq \hat{r}_2 \leq \ldots \leq \hat{r}_m$.

Proof:

The proof is by induction on relation order i=1, to i=m.

For i = 1:

In the initial feasible solution $r_1 = C(s_1)$. Since relation R_1 has the smallest size, transmitting any other relation R_j to R_1 would necessarily increase r_1 since r_1 would equal $C(s_j) + C(p_j * s_1) > C(s_1)$. Therefore, $\hat{r}_1 = C(s_1)$. By the size ordering, for any relation R_j we have $\hat{r}_1 = C(s_1) \leq C(s_j)$. For a size reduction on R_j some relation must be transmitted at the beginning of the schedule for R_j. Since $s_1 \leq s_k$ for all k = 1,..., m this initial transmission cost is at least $C(s_1) = \hat{r}_1$. Thus we have $\hat{r}_1 \leq \hat{r}_j$ for all j = 1,..., m.

For an arbitrary i:

By the definition of a cost beneficial data transmission to minimize response time, the integration of data transmissions into a distributed strategy never increases r_j for any j=1,...,m. Since initially $r_{i-1} = C(s_{i-1})$ for any distribution strategy $r_{i-1} \leq C(s_{i-1}) \leq C(s_i)$. Consider two cases for $\hat{r}_i$.

Case 1: Relation R_{i-1} is transmitted to R_i as part of the minimum response time schedule of R_i. Thus r_{i-1} contributes to the value of $\hat{r}_i$. Therefore $\hat{r}_{i-1} \leq r_{i-1} \leq \hat{r}_i$.

Case 2: Relation R_{i-1} is not transmitted to R_i as part of the minimum response time schedule of R_i. Apply the minimum response time schedule of R_i to the relation R_{i-1}. Since initially $s_{i-1} \leq s_i$ and the same size reductions are performed on both relations clearly $s'_{i-1} \leq s'_i$ where s'_{i-1} and s'_i are the reduced sizes. The time cost of the applied schedule is the same for both relations and $C(s'_{i-1}) \leq C(s'_i)$, therefore $r_{i-1} \leq \hat{r}_i$. Since $\hat{r}_{i-1} \leq r_{i-1}$, we have $\hat{r}_{i-1} \leq \hat{r}_i$.

From cases 1 and 2, $\hat{r}_{i-1} \leq \hat{r}_i$ for all i. ■

Theorem 1:

For a simple query, if, after initial processing, the required relations $R_1,...,R_m$ are ordered so that $s_1 \leq s_2 \leq ... \leq s_m$ then Algorithm C will derive a minimum response time distribution strategy.

Proof:

We first show that Algorithm C derives minimal schedule response time for each relation, i.e. $r_i = \hat{r}_i$ for all i=1,...,m.

For i=1: From Lemma 1, $\hat{r}_1 = C(s_1)$.

For an arbitrary i:

Assume that Algorithm C derives minimum response time schedules for R_j, j=1,...,i-1.

From Lemma 1, since $\hat{r}_i \leq \hat{r}_k$ for k = i+1,...,m, no data transmission from relations R_k, k =i+1,...,m can be included in the minimum response time schedule of R_i. Thus only data transmissions from relations R_j, j=1,...,i-1 need to be considered. The data transmission time of any of the relations R_j, j=1,...,i-1, to R_i, is minimum if $r_j = \hat{r}_j$.

To derive a schedule that finds $\hat{r}_i$, a search must be performed for the minimum r_i over all possible schedules that are composed of data transmissions from relations R_j, j=1,...,i-1. Since from Lemma 1, $\hat{r}_1 \leq \hat{r}_2 \leq \ldots \leq \hat{r}_{i-1}$, by transmitting one of the relations R_j, j=1,...,i-1, to R_i the selectivities of all relations R_k, k=1,...,j, may be used to reduce the size of R_i. This is because the relations R_k can be transmitted in parallel with R_j and will add no additional response time. Thus any arbitrary schedule of data transmissions to R_i can be replaced by the transmission of the largest relation in the schedule along with the parallel transmission of all smaller relations R_k, k=1,...,j-1. The response time of this transmission is $\hat{r}_j$ and the accumulated selectivity is $\prod_{k=1}^{j} p_k$ because of the parallel transmissions of relations R_k.

The data transmission of R_j to R_i has the resulting schedule response time for R_i, $r_i = \hat{r}_j + C(s_i * \prod_{k=1}^{j} p_k)$. This value is calculated for every R_j, j=1,...,i-1, transmission to relation R_i. The minimum of these values along with the original $r_i = C(s_i)$ is the minimum response time, $\hat{r}_i$, for relation R_i.

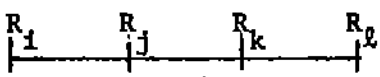
Since Algorithm C implements the checking procedure described above the schedule for relation R_i has the optimal response time $r_i = \hat{r}_i$.

The response time for the distribution strategy derived by Algorithm C is $r = \max_{1 \le i \le m} \{\hat{r}_i\} = \hat{r}_m$. Since the response time of any distribution strategy must be at least $\hat{r}_m$, we have $r = \hat{r}$ as required. ∎

4.2 Minimizing Total Time

Given an ordering on the required relations of a simple query an ordered serial strategy consists of transmitting each relation, starting with R_1, to the next relation in a serial order. The strategy is represented by $R_1 \rightarrow R_2 \rightarrow \ldots \rightarrow R_m \rightarrow R_r$, where R_r is the relation at the result node. There are two cases of the ordered serial strategy. In Case 1 R_r is included in its proper order in the transmission pattern, $R_1 \rightarrow R_2 \rightarrow \ldots \rightarrow R_r \rightarrow \ldots \rightarrow R_m \rightarrow R_r$. In Case 2 R_r is not included in its proper order, $R_1 \rightarrow \ldots \rightarrow R_{r-1} \rightarrow R_{r+1} \rightarrow \ldots \rightarrow R_m \rightarrow R_r$. We will show in this section that ordered serial strategies have minimum total times.

We define a serial schedule to be a schedule in which there are no parallel data transmissions. We denote it by the sequence of relations in the schedule. For example the serial schedule



is denoted by the sequence $\bar{Q} = R_i, R_j, R_k, R_\ell$. The total time (and response time) of a serial schedule is represented by COST $(\bar{Q})$. Let $\bar{Q} = R_{i_1}, \ldots, R_{i_\ell}$. Let P(j) represent the accumulated selectivity of the relations transmitted to relation R_{i_j}. The selectivity of a particular relation can be included only once in the selectivity product by definition. Thus, $P(j) = \prod_{k=1}^{j-1} p'_{i_k}$

where $p'_{i_k} = \begin{cases} 1 \text{ if } i_k = i_\ell \text{ for some } \ell < k \\ p_{i_k} \text{ otherwise.} \end{cases}$

Therefore, $COST(\overline{Q}) = \sum_{j=1}^{\ell} C(s_{i_j} * P(j))$

$= \ell c_o + \Psi(\overline{Q})$ where $\Psi(\overline{Q}) = C_1 * \sum_{j=1}^{\ell} (s_{i_j} * P(j))$.

Two schedules for a relation R_i are equivalent if the relations transmitted in one schedule are a subset of the relations transmitted in the other schedule and the size and selectivity reductions of relation R_i are identical. We now proceed to show that the ordered serial strategy finds the minimum total time for a simple query.

Lemma 2:

Given a serial schedule $\overline{Q} = R_{i_1},\ldots,R_{i_\ell}$, if $i_j = i_k$ for some $j<k$ then the schedules $\overline{Q}$ and $\overline{Q}' = R_{i_1},\ldots,R_{i_j},\ldots,R_{i_{k-1}}, R_{i_{k+1}},\ldots,R_{i_\ell}$ are equivalent and $COST(\overline{Q}') < COST(Q)$.

Proof:

By definition once a relation's selectivity is included in the accumulated selectivity of a serial schedule, another transmission of the same relation can cause no further selectivity reduction. The elimination of relation R_{i_k} in the serial schedule $\overline{Q}'$ results in no change in the size and selectivity reductions of the relation that receives the serial schedule. Thus the serial schedules $\overline{Q}$ and $\overline{Q}'$ are equivalent.

Also

$$\text{COST}(\bar{Q}) - \text{COST}(\bar{Q}) = \sum_{j=1}^{\ell} C(s_{i_j} * P(j)) - \sum_{\substack{j=1 \\ j\neq k}}^{\ell} C(s_{i_j} * P(j'))$$

$$= \ell c_0 + c_1 \sum_{j=1}^{\ell} (s_{i_j} * P(j)) - (\ell-1)c_0 - c_1 \sum_{\substack{j=1 \\ j\neq k}}^{\ell} (s_{i_{j'}} * P(j')).$$

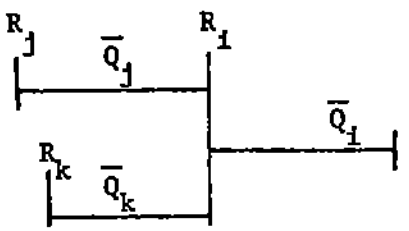
Since R_{i_k} does not contribute any selectivity reduction to P(j) in the first summation, then for all values of j, P(j)= P(j') when j=j'. Thus, $\text{COST}(\bar{Q}) - \text{COST}(\bar{Q}') = c_0 + c_1 (s_{i_k} * p_{(k)}) = C(s_{i_k} * p_{(k)}) > 0$. Therefore, $\text{COST}(\bar{Q}) > \text{COST}(\bar{Q}')$. ■

Lemma 3:

Given a non-serial schedule containing at least one instance of parallel data transmission, an equivalent serial schedule $\bar{Q}$ exists such that $\text{COST}(\bar{Q})$ is less than or equal to the total time cost of the non-serial schedule.

Proof:

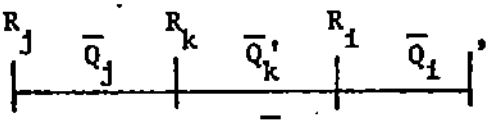
An arbitrary non-serial schedule can be viewed as being composed of parallel components of the form



, where each $\bar{Q}$ represents a serial schedule, possibly empty. Without loss of generality, it is sufficient to show that any such component can be transformed into a serial schedule with less or equal total time. An arbitrary non-serial schedule can be transformed into a serial schedule by a sequence of component transformations.

From Lemma 2 we assume that no relation is contained in $\bar{Q}_i$, $\bar{Q}_j$, or

$\overline{Q}_k$ more than once. Let the sizes and selectivities of the relations before the execution of the schedule component be (s_j,p_j), (s_k,p_k). The size and selectivity of R_i after the data transmissions of $\overline{Q}_j$ and $\overline{Q}_k$ are (sel * s_i, sel * p_i) where sel = $\prod_{R_i \in (\overline{Q}_j \cup \overline{Q}_k)} p_i$. Let $\overline{Q}_k = R_{k_1},\ldots,R_{k_\ell}$. The total time of the parallel component is $COST(\overline{Q}_j) + COST(\overline{Q}_k) + COST(\overline{Q}_i)$ where $COST(\overline{Q}_k) = \ell c_0 + \Psi(\overline{Q}_k)$.

Now consider the serial schedule 

, where $\overline{Q}'_k = R_{k_1},\ldots,R_{k_{\ell'}}$ includes only the relations in $\overline{Q}_k$ that are not in $\overline{Q}_j$. The same size and selectivity reductions are achieved at R_i. Thus the serial schedule is equivalent with the non-serial schedule.

In the serial schedule the size and selectivity of R_k are reduced to (sel' * s_k, sel' * p_k) where sel' = $\prod_{R_\ell \in \overline{Q}_j} p_\ell$. The total time of the serial schedule is $COST(\overline{Q}_j) + COST(\overline{Q}'_k) + COST(\overline{Q}_i)$ where $COST(\overline{Q}'_k) = \ell' c_0 + sel' * \Psi(\overline{Q}'_k)$. Since $\ell' \leq \ell$, sel' ≤ 1, and $\Psi(\overline{Q}'_k) \leq \Psi(\overline{Q}_k)$ clearly $COST(\overline{Q}'_k) \leq COST(\overline{Q}_k)$. Thus the serial schedule has a total time less than or equal to the total time of the non-serial schedule. ∎

The application of Lemmas 2 and 3 to any feasible distribution strategy would transform it uniquely into a serial schedule in which each required relation is transmitted exactly once (except, perhaps, the relation at the result node). Lemma 4 now proves that performing the serial data transmissions of the schedule in a specified order has the least total cost.

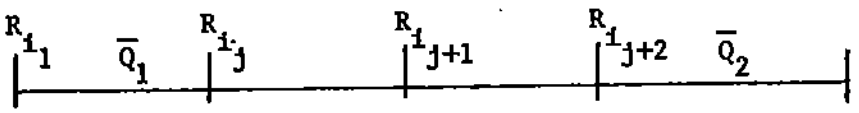
Lemma 4:

For a simple query if, after initial processing, the required relations $R_1,\ldots,R_m$ are ordered so that $s_1 \leq s_2 \leq \ldots \leq s_m$ then the

serial schedule $\bar{Q} = R_{i_1},\ldots,R_{i_\ell}$, where $i_j < i_{j+1}$ for all $j=1,\ldots,\ell-1$, has the minimum total cost among all equivalent schedules.

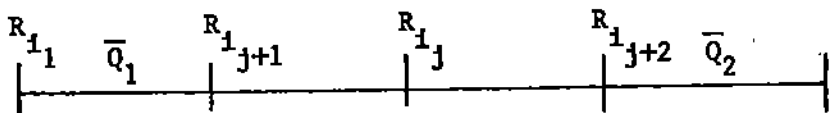
Proof:

Assume that $i_j > i_{j+1}$ for some $j=1,\ldots,\ell-1$ in the serial schedule $\bar{Q}=R_{i_1},\ldots,R_{i_\ell}$. Consider the serial schedule $\bar{Q}' = R_{i_1},\ldots,R_{i_{j+1}}, R_{i_j},\ldots,R_{i_\ell}$. Since the order of relations R_{i_j} and $R_{i_{j+1}}$ are simply reversed in the schedule no change would occur in the final size and selectivity reductions. Thus $\bar{Q}$ and $\bar{Q}'$ are equivalent. $\bar{Q}$ can be broken into the serial components



where $\bar{Q}_1 = R_{i_1},\ldots,R_{i_{j-1}}$ and $\bar{Q}_2 = R_{i_{j+2}},\ldots,R_{i_\ell}$. Thus, $COST(\bar{Q}) = COST(\bar{Q}_1) + C(q\, s_{i_j}) + C(qp_{i_j} s_{i_{j+1}}) + COST(\bar{Q}_2)$ where $q = \prod_{R_\ell \in \bar{Q}_j} p_\ell$.

Now consider $\bar{Q}'$.



We have $COST(\bar{Q}') = COST(\bar{Q}_1) + C(qs_{i_{j+1}}) + C(qp_{i_{j+1}} s_{i_j}) + COST(\bar{Q}_2)$.

By the definition of the relation ordering $s_{i_{j+1}} \leq s_{i_j}$ where $i_{j+1} < i_j$.

Since we are dealing with only one common joining domain queries, this implies that $p_{i_{j+1}} \leq p_{i_j}$ and $(1- p_{i_j}) \leq (1- p_{i_{j+1}})$.

Thus, $s_{i_{j+1}}(1 - p_{i_j}) \leq s_{i_j}(1 - p_{i_{j+1}})$ or $s_{i_{j+1}} + p_{i_{j+1}} s_{i_j} \leq s_{i_j} + p_{i_j} s_{i_{j+1}}$.

$$\text{Now } C(qs_{i_{j+1}}) + C(q\, p_{i_{j+1}}\, s_{i_j}) = 2c_0 + c_1\, q\, (s_{i_{j+1}} + p_{i_{j+1}}\, s_{i_j})$$

$$\leq 2c_0 + c_1\, q\, (s_{i_j} + p_{i_j}\, s_{i_{j+1}})$$

$$= C(q\, s_{i_j}) + C(q\, p_{i_j}\, s_{i_{j+1}})$$

Therefore COST $(\bar{Q}') <$ COST$(\bar{Q})$. ∎

Theorem 2 can now be proved by use of Lemmas 2, 3, and 4.

Theorem 2:

For a simple query if, after initial processing, the required relations $R_1,\ldots,R_m$ are ordered so that $s_1 \leq s_2 \leq \ldots \leq s_m$ then an ordered serial strategy has the minimum total time. Case 1 of the ordered serial strategy is optimal if

$$(1 - p_r) > \frac{\frac{c_0}{c_1} + s_r \prod_{j=1}^{r=1} p_j}{\sum_{i=r+1}^{m} s_i \prod_{\substack{j=1 \\ j\neq r}}^{i-1} p_j}$$

Otherwise Case 2 of the ordered serial strategy is optimal.

Proof:

Any feasible distribution strategy must include the transmission of all required relations to the result node. Among all feasible distribution strategies there must exist at least one strategy with minimum total time. By the use of Lemmas 2, 3, and 4 any such optimal distribtuion strategy has an equivalent ordered serial strategy with less or equal total time. Thus the ordered serial strategy must have minimum total time.

There are two possible cases of the ordered serial strategy. Which case has the minimum total time must be tested.

From the definition of the ordered serial strategy, the total time

for Case 1, where R_r is included in the ordered serial strategy, is:

$\sum_{i=1}^{m} C(s_i \prod_{j=1}^{i-1} p_j)$. For Case 2, where R_r is not included in the ordered serial strategy, the total time is: $\sum_{\substack{i=1 \\ i\neq r}}^{m} C(s_i \prod_{\substack{j=1 \\ j\neq r}}^{i-1} p_j)$.

Case 1 of the ordered serial strategy is optimal if:

$$\sum_{i=1}^{m} C(s_i \prod_{j=1}^{i-1} p_j) < \sum_{\substack{i=1 \\ i\neq r}}^{m} C(s_i \prod_{\substack{j=1 \\ j\neq r}}^{i-1} p_j) .$$

Eliminating common terms:

$$\sum_{i=r}^{m} C(s_i \prod_{j=1}^{i-1} p_j) < \sum_{i=r+1}^{m} C(s_i \prod_{\substack{j=1 \\ j\neq r}}^{i-1} p_j) .$$

Making the summation range equivalent:

$$C(s_r \prod_{j=1}^{r-1} p_j) < \sum_{i=r+1}^{m} [C(s_i \prod_{\substack{j=1 \\ j\neq r}}^{i-1} p_j) - C(s_i \prod_{j=1}^{i-1} p_j)]$$

$$c_0 + c_1 s_r \prod_{j=1}^{r-1} p_j < \sum_{i=r+1}^{m} [c_0 + c_1 s_i \prod_{\substack{j=1 \\ j\neq r}}^{i-1} p_j - c_0 + c_1 s_i \prod_{j=1}^{i-1} p_j]$$

$$\frac{c_0}{c_1} + s_r \prod_{j=1}^{r-1} p_j < (1 - p_r) \sum_{i=r+1}^{m} s_i \prod_{\substack{j=1 \\ j\neq r}}^{i-1} p_j$$

$$\frac{\frac{c_0}{c_1} + s_r \prod_{j=1}^{r-1} p_j}{\sum_{i=r+1}^{m} s_i \prod_{\substack{j=1 \\ j\neq r}}^{i-1} p_j} < (1 - p_r) .$$

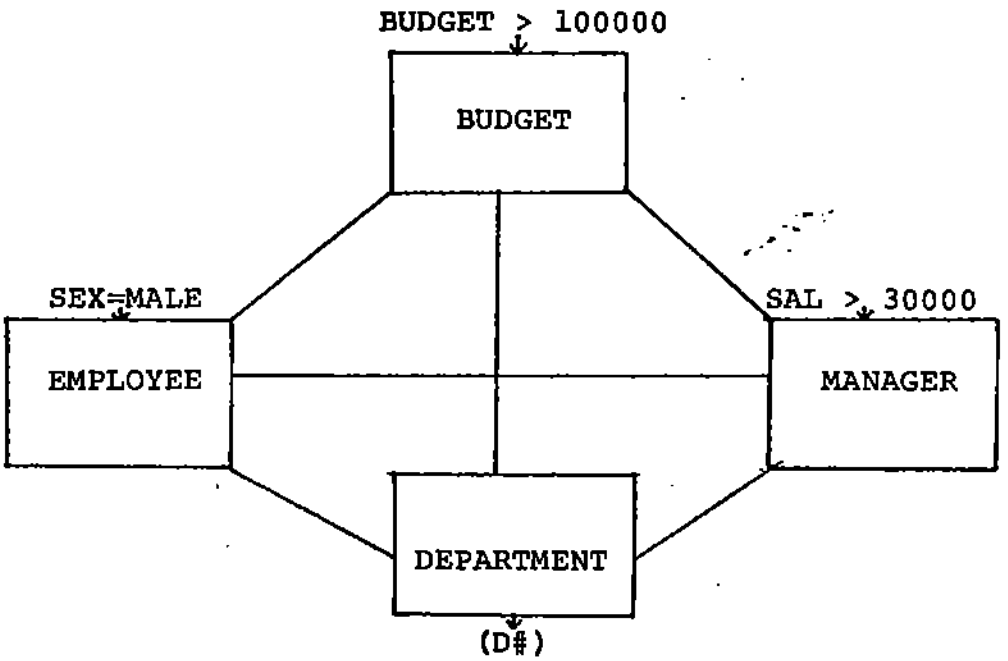
4.3. A Simple Query Example

We present a simple example for which Algorithm C and Wong's Algorithm [12] will be used to find distribution strategies. Response times and total times will be compared for the initial feasible solution; the Algorithm C strategy; the Wong's Algorithm Strategy; and the optimal total time serial strategy.

Example 4:

A distributed database has the following four relations at separate network nodes.

Relation	Variable
DEPARTMENT (D#,LOC)	D
EMPLOYEE (E#,NAME,D#,SEX)	E
MANAGER (E#,D#,SAL)	M
BUDGET (D#,BUDGET)	B



The joining domain is D#.

Figure 3: Query for Example 4

The query illustrated in Figure 3 is entered into the system. Assume that the result node is distinct from the nodes having required data.

The first step of all distribution strategies is to do local processing. After the restrictions are performed on the domains E.SEX, M.SAL, and B.BUDG the joining domain D// is projected for all relations. Let the size and selectivities be:

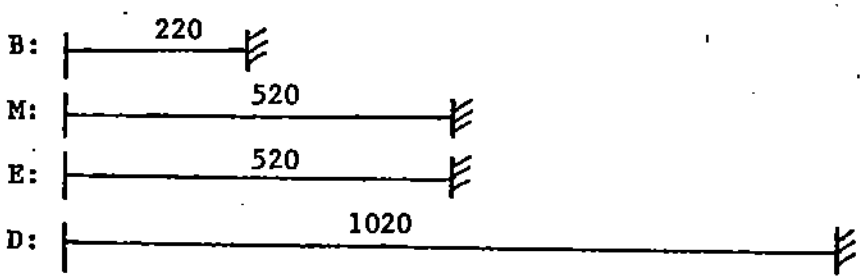
Relation	Size(s_i)	Selectivity(p_i)
B	200	1/5
M	500	1/2
E	500	1/2
D	1000	1

With the small to large relation ordering (B,M,E,D), Theorems 1 and 2 guarantee that the minimum response time and the minimum total time distribution strategies can be derived for this query.

Let $C(X) = 20 + X$.

1. Initial feasible solution:

The cost graph is:



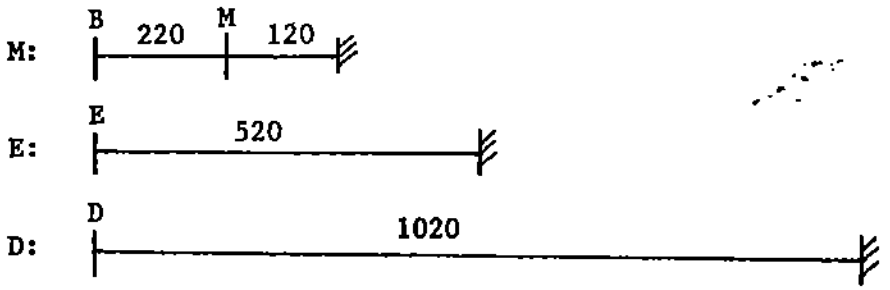
Response time = 1020. Total time = 2280.

2. Algorithm C:

Algorithm C finds $\hat{r}_i$ for each relation from smallest to largest.

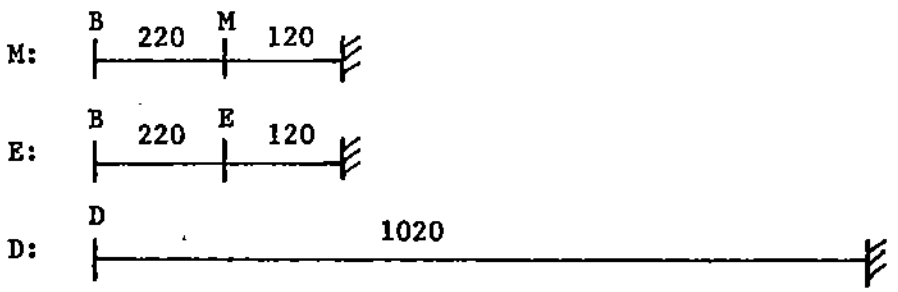
i) Find $\hat{r}_B$. Since B is the smallest relation, no data transmission can reduce r_B. Thus $\hat{r}_B$ = C(200) = 220.

ii) Find $\hat{r}_M$. The data transmission from relation B is beneficial. Thus, $\hat{r}_M$ = C(200) + C(1/5 * 500) = 220 + 120 = 340. Eliminating the schedule of relation B the new cost graph is:



iii) Find $\hat{r}_E$. The data transmission from relation B is the most cost beneficial. Thus, $\hat{r}_E$ = C(200) + C(1/5 * 500) = 220 + 120 = 340.

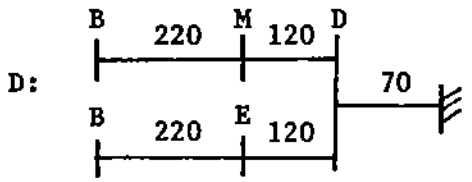
The new cost graph is:



iv) Find $\hat{r}_D$. The parallel data transmission of relations M and E is the most cost beneficial. Thus, $\hat{r}_D$ = C(200) + C(100) + C(1/20 * 1000)

= 220 + 120 + 70 = 410.

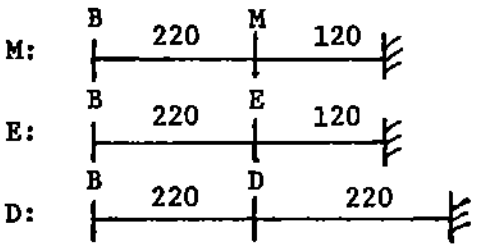
The final cost graph is:



For the distribution strategy derived by Algorithm C the minimum response time is $\hat{r} = \hat{r}_D = 410$; the total time is $t = 750$.

Wong's Algorithm

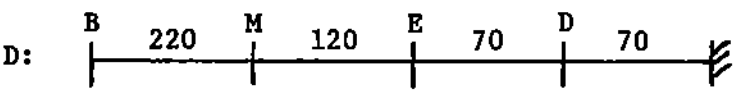
Using Wong's algorithm we find all data transmissions such that the amount of data transmitted is less than the amount of data reduction at the receiving relation. With this testing method we use the algorithm as described in reference [12] . The final cost graph is:



Response time = 440. Total time = 1120.

4. Serial Strategy

Since there is no relation at the result node both cases of the serial strategy are identical. The final cost graph is:



Response time = 480. Minimum total time t = 480.

Table 1 shows the times for the four distribution strategies of this query.

	RESPONSE TIME	TOTAL TIME
INITIAL FEASIBLE SOLUTION	1020	2280
ALGORITHM C STRATEGY	410	750
WONG'S ALGORITHM STRATEGY	440	1120
SERIAL STRATEGY	480	480

TABLE 1

The most striking observation on Table 1 is the large time differences between the initial feasible solution and the other three distribution strategies. We can conclude that the use of an algorithm to derive efficient distribution strategies is an important part of a distributed database management system. The Algorithm C strategy and the serial strategy are both less costly than the Wong's Algorithm strategy. This is to be expected since Algorithm C and the serial strategy definitions are designed to function optimally with a relation ordering in a simple query environment. Wong's Algorithm and the initial feasible solution are applicable to a general query environment.

A total of ten simple queries similar to the query in Figure 3 were formulated on the distributed database of Example 5. Table 2 lists the differing sizes and selectivites of the ten queries. The four distribution strategies of Table 1 were derived for all ten queries. The response times for the strategies are compared in Graph 1 and the total times are compared in Graph 2.

5. Conclusions

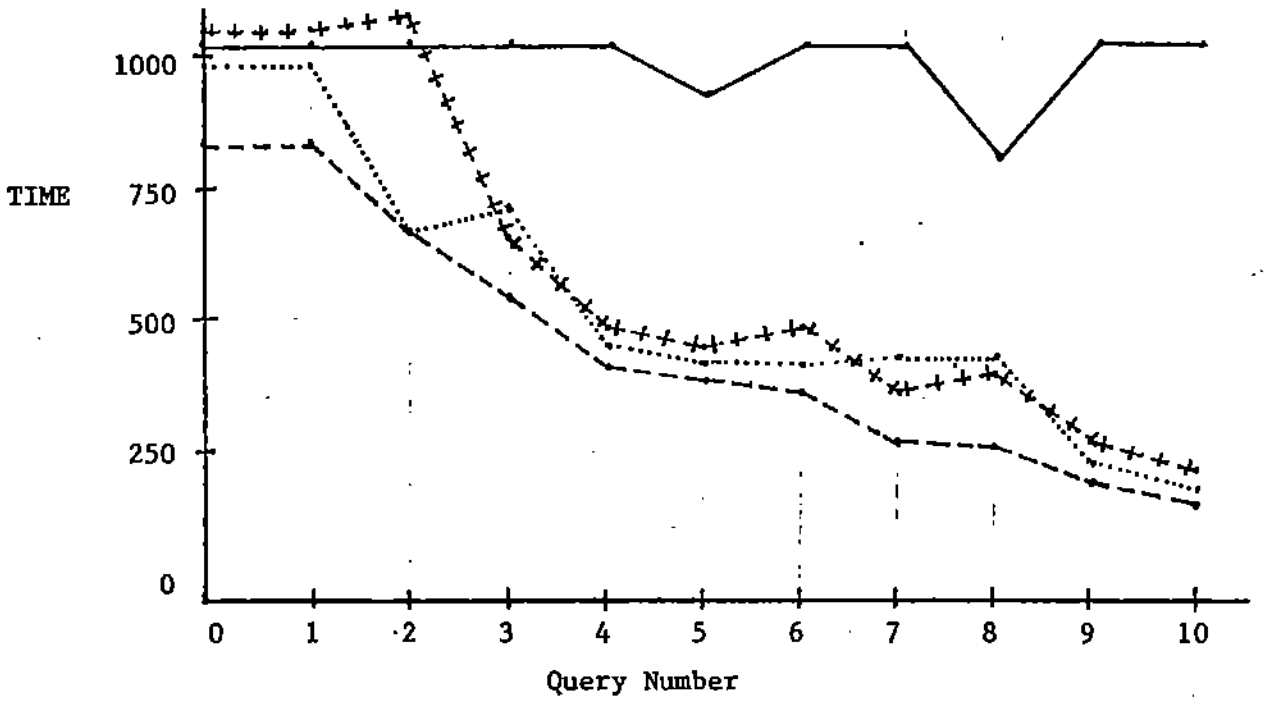
Data access via queries in a distributed system requires a synchronized pattern of data transmissions and local processing known as a distribution strategy. In this paper we have presented methods to design efficient distribution strategies by employing several straight forward distribution tactics. For a simplified query environment these tactics are used to derive distribution strategies which are shown to be optimal in terms of minimizing response time (Algorithm C) and total time (the ordered serial strategy). These results are simple to use and computationally efficient. Only a small set of relation statistics (size and selectively) are required. Comparative examples are given to show the advantages of our distribution strategies over conventional heuristic strategies.

Relation sizes and selectivities

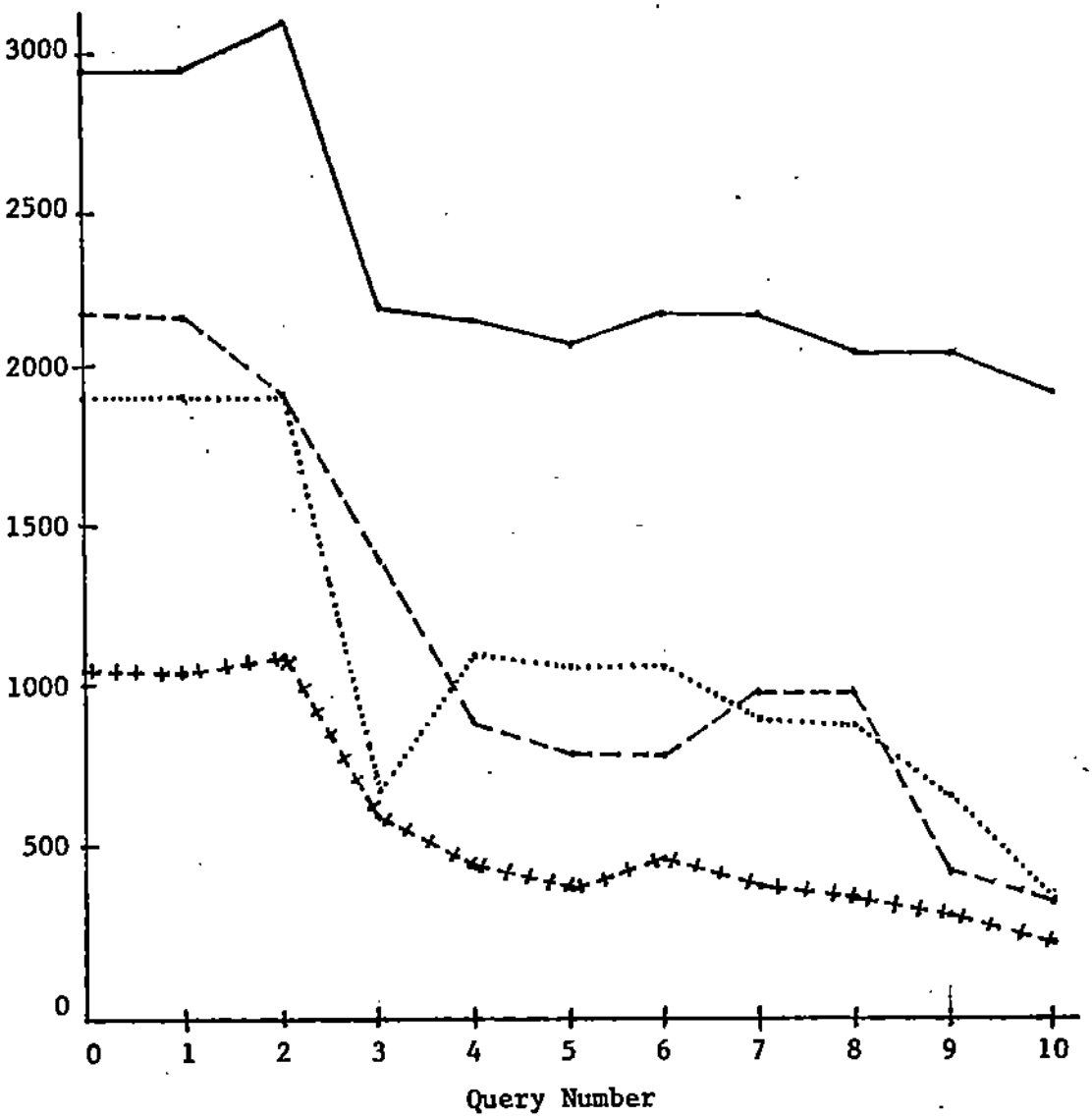
Query Number	Relation R_1		Relation R_2		Relation R_3		Relation R_4	
	s_1	p_1	s_2	p_2	s_3	p_3	s_4	p_4
1	400	.4	600	.6	800	.8	1000	1.0
2	300	.3	800	.8	1000	1.0	1000	1.0
3	300	.3	400	.4	500	.5	1000	1.0
4	200	.2	400	.4	600	.6	1000	1.0
5	200	.2	400	.4	500	.5	900	.9
6	200	.2	500	.5	500	.5	1000	1.0
7	200	.2	200	.2	800	.8	800	.8
8	200	.2	200	.2	800	.8	1000	1.0
9	100	.1	400	.4	500	.5	1000	1.0
10	100	.1	200	.2	500	.5	1000	1.0

TABLE 2

Statistics of Ten Test Queries



GRAPH 1 : Response Time



GRAPH 2 : Total Time

The importance of these optimal algorithms lies in the ability to extend their tactics from a simplified environment to the general distributed environment. A general environment could include a non-totally connected network, significant local processing times, queueing delays on communication lines, and a more complex cost function for data transmission in the network. A general distribution algorithm must consider the differences between a simple query and a general distributed query which may contain any number of joining domains and output domains after initial processing. For a general query joining paths do not necessarily connect any two required relations and each node may contain several required relations. The search space of feasible distribution strategies is enormous for such a general query environment. It no longer seems possible to find a simple distribution algorithm that will derive an optimal strategy for any query.

The development and implementation of general heuristic distribution algorithms are ongoing research areas in distributed systems. The ability to retrieve and update distributed data in a timely manner is a key requirement of an effective data management system. The optimal algorithms developed in this paper will serve as a basis for the design of heuristic distribution algorithms for efficient data access in distributed systems.

REFERENCES

1. Bernstein, P.A.; Rothnie, J.B.; Shipman, D.W.; and Goodman, N. The DSS-1 Redundant Update Algorithm (The General Case), Technical Report No. CCA-77-09, Computer Corporation of America, 575 Technology Square, Cambridge, Massachusetts 02139, August 1, 1977.

2. Booth, G.M. "Distributed Information Systems", Proceedings 1976 AFIPS National Computer Conference, AFIPS Press, Vol. 45, 1976, pp. 789-794.

3. Champine, G.A. "Six Approaches to Distributed Databases", Datamation, Vol. 23 No. 5, Technical Publishing Company, Barrington Illinois, May 1977, pp. 69-72.

4. CODASYL Systems Committee, "Distributed Data Base Technology - An Interim Report of the CODASYL Systems Committee", Proceedings 1978 AFIPS National Computer Conference, AFIPS Press, Vol. 47, 1978, pp. 909-917.

5. Codd, E.F. "A Relational Model of Data for Large Shared Data Banks", Communication of the ACM, Vol. 13 No. 6, Time 1970, pp. 377-387.

6. Computer Corporation of America, "A Distributed Database Management System for Command and Control Applications: Semi-Annual Technical Report 2", Technical Report No. CCA-78-03, January 30, 1978.

7. Deppe, M.E.; and Fry, J.P. "Distributed Databases: A Summary of Research", Computer Networks, Vol. 1 No. 2, North-Holland Publishing Company Amsterdam, The Netherlands, September 1976, pp. 130-138.

8. Epstein, R.; Stonebraker, M.; and Wong, E. "Distributed Query Processing in a Relational Data Base System", Proceedings ACM 1978 SIGMOD Conference, Austin, Texas, June 1978, pp. 169-180.

9. Rothnie, J.B.; and Goodman, N. "An Overview of the Preliminary Design of SDD-1: A System for Distributed Databases", 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, University of California, Berkeley California, May 1977, pp. 39-57.

10. Rothnie, J.B.; and Goodman, N. "A Survey of Research and Development in Distributed Database Management", 1977 Proceedings on Very Large Data Bases, Tokyo, Japan, October 1977, pp. 48-62.

11. Stonebraker, M.; and Neuhold, E. "A Distributed Database Version of INGRES", 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, University of California, Berkeley California, May 1977, pp. 19-36.

12. Wong, E. "Retrieving Dispersed Data from SDD-1: A System for Distributed Databases", 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, University of California, Berkeley California, May 1977, pp. 217-235.

13. Wong, E.; and Youssefi, K. "Decomposition - A Strategy for Query Processing", ACM Transactions on Database Systems, Vol. 1 No. 3, September 1976, pp. 223-241.

14. Yao, S.B.; and DeJong, D. "Evaluation of Database Access Paths", Proceedings ACM 1978 SIGMOD Conference, Austin, Texas, June 1978, pp. 66-77.

15. Yao, S.B. "Optimization of Query Evaluation Algorithms" (unpublished manuscript), March, 1978.

16. Stonebraker, M.; Wong, E.; Kreps, P.; and Held, G. "The Design and Implementation of INGRES," ACM Transactions on Database Systems, Vol. 1, No. 3, September 1976, pp. 189-222.

APPENDIX A

We will use the query of Example 1 to illustrate the deficiencies of the centralized tactics in a distributed system. Assume that relations PRODUCT, ORDER, and PROD-ORD are located at separate nodes in a network. Assume the query was entered at the node containing relation ORDER. Thus, the result relation must end up at that node. After initial processing the resultant query is shown in Figure 3. The relation at the result node is indicated by a double box.

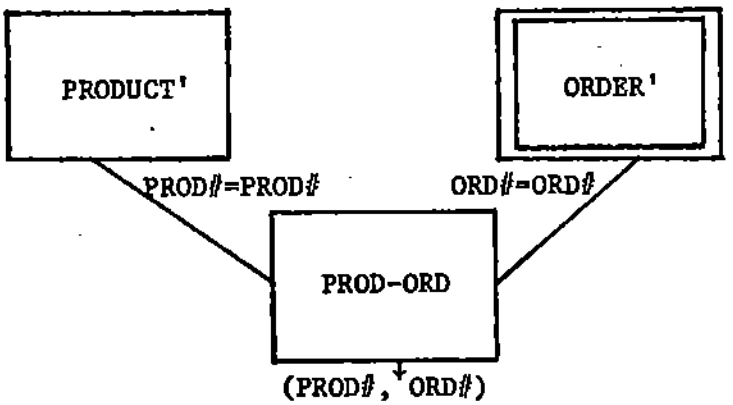
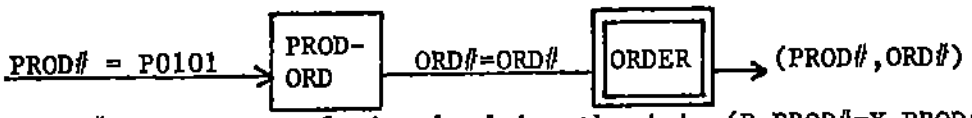


Figure A1

The relation PRODUCT' is the result of applying the restriction (P.QOH < 1000). The relation ORDER' is the result of applying the restriction (O.DATE < 780901). Assume that the system parameters of size and selectivity have the values $(s_P, P_{P.PROD\#}) = (200, \frac{1}{2})$, $(s_O, P_{O.ORD\#}) = (200, \frac{1}{2})$, and $(s_X, P_{X.PROD\#}, P_{X.ORD\#}) = (800, 1, 1)$.

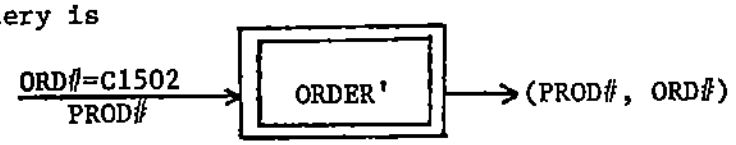
Tuple substitution can be used to find the result relation of the query. There are several different orders in which we can substitute tuples. The basic deficiency of tuple substitution is apparent regardless of substitution order. For example, let us first substitute tuples from PRODUCT'

into PROD-ORD. Let one such tuple be P.PROD# = P0101. The query is reduced to



If each PROD# value were of size 1, doing the join (P.PROD#=X.PROD#) would require 200 data transmissions of size 1. The total transmission time would be 200* C(1).

Implementing the join (X.ORD# = O.ORD#) by substituting tuples from PROD-ORD to ORDER would require approximately 200 data transmission of size 2 since X.PROD must be sent for output. For example, if X.ORD# = C1502 then the query is



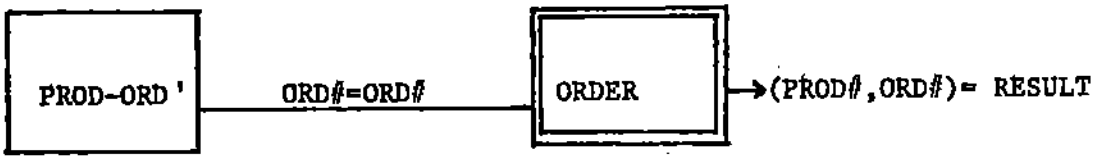
The total transmission time to solve these queries would be 200 * C(2).

The data transmission time to solve the original query using tuple substitution is 200 * C(1) + 200 * C(2) = $400c_0 + 600c_1$. However, transmitting the whole relations PRODUCT' and PROD-ORD' (after joining) would take time C(200) + C(400) = $2c_0 + 600c_1$. When the transmission startup cost, c_0, is significant, tuple substitution is inappropriate for a distribution strategy because of the large number of data transmissions that are required.

The tactic of reduction breaks a query into subqueries that must be performed in sequential order. A distribution strategy is found for each of the subqueries.

The query in Figure 3 can be reduced into the two subqueries in Figure A2.

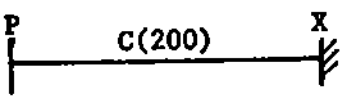
Subquery 1 (SQ1)



Subquery 2 (SQ2)

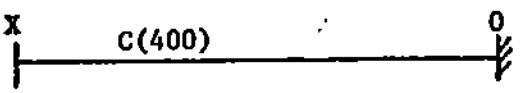
Figure A2

The minimum time distribution strategy for SQ1 would be to transmit PRODUCT' to PROD-ORD and do the join.



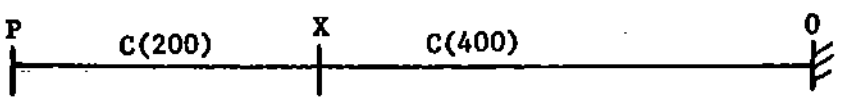
Response time=Total time= C(200).

The minimum time distribution strategy for SQ2 would be to transmit PROD-ORD' to ORDER' and do the join.



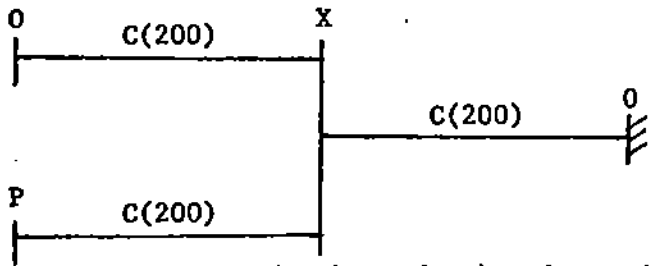
Response time=Total time= C(400).

The distribution strategy for the overall query would combine the strategies for SQ1 and SQ2. Since the strategy for SQ2 cannot begin until the SQ1 strategy completes the resultant distribution strategy would be:



Response time= Total time = C(200) + C(400)
$= 2C_0 + 600C_1$.

However, consider the following distribution strategy.



Response time= C(200) + C(200) = $2c_o + 400c_1$.

Total time = C(200) + C(200) + C(200) = $3c_o + 600c_1$.

The response time is greatly reduced and the total time is nearly the same in comparison with the previous distribution strategy. It is evident that the tactic of reduction is, in this case, inefficient because of the procedural constraints that it places upon the search for an optimal distribution strategy for a given query.

APPENDIX B

Algorithm C is presented in a Pascal-type language. The set sch_i contains the relations in the schedule of R_i and the set drop contains schedules that can be eliminated. Relation R_r is assumed to be at the result node.

Algorithm C

```
begin
   sel_0 := 1.0;                    (* sel_i holds cumulative selectivity
                                       product *)
   for i := 1 to m do
   begin
      r_i := C(s_i);                (* initial feasible solution *)
      sel_i := sel_i-1 * p_i;
      sch_i := [i];                 (* relations in R_i schedule *)
   end;
   drop := [r];                     (* schedules to be eliminated *)
   i := 2;                          (* current relation *)
(* Find most beneficial data move to R_i *)
   while (i < m+1) do
   begin
      low := i - 1;
      sizopt := r_i-1;              (* Initialize R_i-1 as best transmission *)
      selopt := sel_i-1;
```

```
(* Compare each relation R_j transmission with best previous transmission *)
      for j := i - 2 down to 1 do
      if (r_j + C(sel_j * s_i) <  sizopt + C(selopt * s_i)) then
      begin
         low := j;
         sizopt := r_j;
         selopt := sel_j;
      end;
(* Check if best transmission is beneficial - if so, add to solution *)
      if (r_i >  sizopt + C(selopt * s_i)) then
      begin
         "move R_low to R_i";
         sch_i := sch_i + sch_low;      (* add relations in sch_low to sch_i *)

(* Add parallel moves *)
         for k := low - 1 down to 1 do
         begin
            drop := drop + [ k ] ;
            if (k not in sch_i) and (p_k ≠ 1) then
            begin
               "transmit R_k to R_i";
               sch_i := sch_i + sch_k;
            end;
         end;
         "perform local processing at R_i";
         s_i := selopt * s_i;          (* new size *)
         r_i := sizopt + C(s_i);       (* r̂_i *)
      end;
(* minimize next relation schedule *)
      i := i + 1;
   end; (* of while i ≤ m + 1 loop *)
   "eliminate schedules for R_ℓ where ℓ in drop";
end;  (* of Algorithm C *)
```