Query Processing in Distributed Database System

Article *in* IEEE Transactions on Software Engineering · June 1979

DOI: 10.1109/TSE.1979.234179 · Source: IEEE Xplore

2 authors, including:

Alan Hevner
University of South Florida
238 PUBLICATIONS **21,550** CITATIONS

SEE PROFILE

Query Processing in Distributed Database Systems

ALAN R. HEVNER AND S. BING YAO

Abstract—Query processing in a distributed system requires the transmission of data between computers in a network. The arrangement of data transmissions and local data processing is known as a distribution strategy for a query. Two cost measures, response time and total time are used to judge the quality of a distribution strategy. Simple algorithms are presented that derive distribution strategies which have minimal response time and minimal total time, for a special class of queries. These optimal algorithms are used as a basis to develop a general query processing algorithm. Distributed query examples are presented and the complexity of the general algorithm is analyzed. The integration of a query processing subsystem into a distributed database management system is discussed.

Index Terms—Computer network, database, distributed database systems, distributed processing, distribution strategy, heuristic algorithms, query processing, redundant data, relational data model, system modeling.

I. Introduction

THE distribution of data in a network or decentralized computer system offers several attractive advantages over the centralization of data at a single computer. These advantages include increased data reliability; faster, localized access to data; and the potential for upward scaling of data capacity [6]. One of the important problems is the efficient processing of queries in a distributed system. Accessing data that are stored at separate computers differs in two important ways from accessing data from a centralized computer. The necessary transmission of data over communication lines introduces substantial time delays. An advantage of the distributed system is the ability to process and transmit data in parallel at separate points in the network. The database management system must consider these facts and derive an effective arrangement of local data processing and data transmissions in order to process distributed queries. This arrangement of data processing and data transmissions is known as a *distribution strategy* for a query [8]. Among feasible distribution strategies for a given query, great variations in time cost exist [6], [4]. The importance of using efficient query processing methods on distributed systems is thus emphasized.

Previous research in this area has utilized classical optimization search techniques to find efficient distribution strategies. Wong has proposed an algorithm that is currently being implemented in the SDD-1 system [8], [2]. An initial feasible strategy is selected. Then a standard "hill-climbing" optimization technique recursively finds lower cost strategies until no further cost improvements can be discovered. The resultant distribution strategy is a local minimum cost solution in the search space.

Other data access methods extend centralized query tactics [10], [11] to a distributed environment in order to find feasible distribution strategies on the network [7], [3].

For a class of simple queries, it is possible to design optimization algorithms that will derive optimal distribution strategies in the sense that some cost measure is minimized [4]. In this paper an optimization algorithm for general queries in a distributed system is developed. The general algorithm is shown to be computationally efficient. The derived distribution strategy is shown to compare favorably with other feasible distribution strategies. The ability to integrate the query optimization algorithm into the design of a distributed database management system is discussed.

II. Distributed Query Processing

We consider a network of interconnected computers. Each computer, known as a node in the network, contains a distributed database management system (DDBMS) and a possibly redundant portion of the database. Data are logically viewed in the relational data model [1]. The unit of data distribution is a relation. The DDBMS will maintain system directories so that each query will receive a nonredundant consistent mapping of its required data. The distributed system can be characterized by the parameters defined in Table I.

Data transmission in the network is via communication links. The data transmission cost between any two nodes is defined as a linear function $C(X) = c_0 + c_1 X$, where X is the amount of data transmitted. We define our cost measure in units of time. The constant c_0 represents an initial start-up time for each separate transmission. An important property of $C(X)$ is that if $X \leqslant Y$, then $C(X) \leqslant C(Y)$. It is further assumed that the cost of data transmissions between nodes is significantly greater than the cost of local processing and intranodal data transfers between local storage devices.

A relational query performs the operations RESTRICTION, PROJECTION, and JOIN in order to retrieve data [1]. PROJECT eliminates unneeded domains from relations. RESTRICT selects rows of a relation that satisfy specified data conditions. JOIN

Manuscript received February 6, 1978; revised November 1, 1978. This research was supported by the National Science Foundation under Grant MCS76-16604. This is an expanded version of a paper presented at the Third Berkeley Workshop on Distributed Databases.

A. R. Hevner is with the Department of Computer Science, Purdue University, West Lafayette, IN 47907.

S. B. Yao is with the Department of Computer Science, Purdue University, West Lafayette, IN 47907.

0098-5589/79/0500-0177$0075

TABLE I
SYSTEM PARAMETERS

For the distributed system:

- N Number of nodes.
- M Number of unique relations.

For each relation $R_i, i = 1, \cdots, M$:

- n_i Number of records.
- a_i Number of domains.
- s_i Size. $s_i = n_i \cdot \sum_{j=1}^{a_i} w_{ij}$.

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

- v_{ij} Number of possible domain values.
- u_{ij} Number of values domain currently holds.
- p_{ij} Selectivity. $p_{ij} = u_{ij}/v_{ij}$. $(0 < p_{ij} \leq 1)$.
- w_{ij} Size of data item in domain.
- b_{ij} Projected size of domain (no duplicate values). $b_{ij} = u_{ij} \cdot w_{ij}$.

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_{kl}. After the RESTRICTION is performed the parameters of R_k are changed as follows:

1) $s_k \leftarrow s_k \cdot q$,
2) $p_{kl} \leftarrow p_{kl} \cdot q$.

A JOIN between domains d_{ij} and d_{kl} 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_{kl} for the restriction on R_i.

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. In a distribution strategy, all possible initial local processing should be done first. This processing results in the following parameters being defined.

- m Number of relations in the remaining query.
- α_i Number of domains in relation R_i.
- β_i Number of internodal joining domains in relation R_i.

The reduced size of each relation is $s_i = n_i \cdot \sum_{j=1}^{\alpha_i} w_{ij}$, 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 distribution strategy (the intermediate sequence of data transmission and local processing) is optimized by minimizing an appropriate cost function.

In order to visualize clearly the structure and interaction within a strategy we introduce a graphic notation. The data transmission pattern containing the transmission of relation R_i to the result node is called the *schedule for* R_i. Consider the cost graph of a distribution strategy in Fig. 1. Assume that each relation is at a separate node and the R_3 is at the result node. Consider the schedule for R_2 in the cost graph. Joining domains d_{11} and d_{32} are transmitted in parallel to relation R_4. The size of R_4 is reduced and domain d_{41} is transmitted to relation R_2. The size of R_2 is reduced and R_2 is transmitted to the result node.

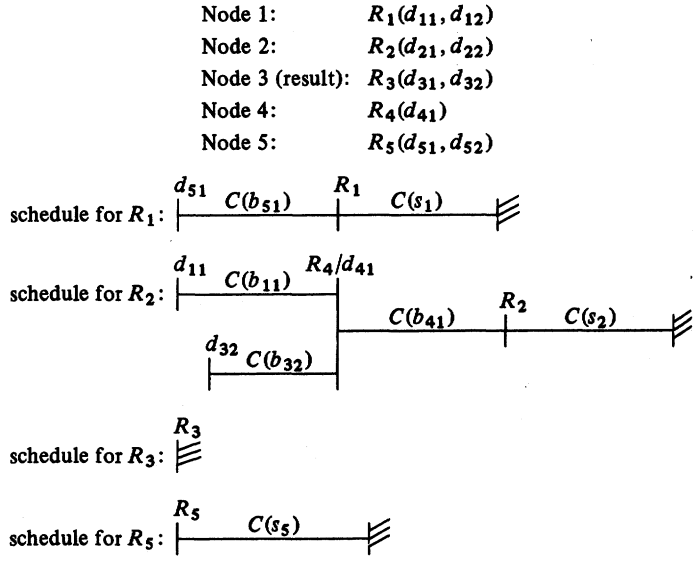


Fig. 1. Cost graph of a distribution strategy.

In a distribution strategy it is possible to combine two schedules. This occurs when the entire data of one relation is transmitted to another relation for a join. In Fig. 1 relation R_4 consists of only domain d_{41}. Its schedule can be eliminated since the domain d_{41} is already included in the schedule of R_2.

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. We define *schedule delay time* γ_i as the time from the start of the schedule until the start of relation R_i's data transmission to the result node. Thus, $r_i = \gamma_i + C(s_i)$. For the R_2 schedule in Fig. 1, $r_2 = C(b_{11}) + C(b_{41}) + C(s_2)$ and $\gamma_2 = C(b_{11}) + C(b_{41})$, where b_{ij} is the projected size (no duplicate values) of the domain d_{ij}. The time $C(b_{32})$ is not included because it occurs in parallel with time $C(b_{11})$. We define *schedule total time* t_i as the sum of all times in the schedule. For relation R_2, $t_2 = C(b_{11}) + C(b_{32}) + C(b_{41}) + C(s_2)$. We define the *minimal response time of a schedule* as $\hat{r}_i = \min(r_i)$ where the minimization ranges over all possible schedules. The *response time* of the distribution strategy is defined as $r = \max_{1 \leq i \leq m}(r_i)$ and the *total time* of the distribution strategy is defined as $t = \sum_{i=1}^{m} t_i$.

In this paper the optimization of distribution strategies under two cost objectives are analyzed: the minimization of response time r and the minimization of total time t.

III. Optimal Distribution Strategies for Simple Queries

A query optimization algorithm is an algorithm that derives a distribution strategy for a given query. We have introduced query optimization algorithms that derive optimal distribution strategies for a class of distributed queries called *simple queries* [4]. A simple query is defined such that after initial local processing each relation in the query contains only one domain—the common joining domain. Thus $\alpha_i = \beta_i = 1$ for all $R_i, i = 1, \cdots, m$. In what follows, we will review the query optimization algorithms for simple queries. An outline of the optimality proof is given in the Appendix. These

algorithms will be used as a basis for the development of a general optimization algorithm in the next section.

A. Minimizing Response Time

The optimization algorithm PARALLEL was shown to derive a minimal response time distribution strategy for any given simple query. Algorithm PARALLEL uses the initial feasible solution which moves all required relations directly to the result node as a starting strategy. The algorithm searches for cost beneficial data transmissions in the current system state 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. Algorithm PARALLEL searches for cost beneficial data transmissions by trying to join small relations to large relations. All relations R_j where $j<i$ are checked for potential data transmission to R_i and the data transmission that produces the greatest reduction in r_i is integrated into the distribution strategy. For the data transmission from R_j to R_i, the accumulated selectivity of R_j is $\pi_{k=1}^{j} p_k$ since all relations R_k can be transmitted to R_i in parallel with R_j for no additional response time.

Briefly, Algorithm PARALLEL can be described as follows.

Algorithm PARALLEL

1) Order relations R_i so that $s_1 \leqslant s_2 \leqslant \cdots \leqslant s_m$.
2) Initialize the parameters of the system state s_i, p_i, and $r_i = C(s_i)$ for all R_i.
3) Repeat Steps 4-6 for $i = 1$ to $i = m$.
4) (Find $\hat{r}_i$.) For each relation R_j, $j<i$, calculate the value $r_i' = \hat{r}_j + C(s_i \cdot \pi_{k=1}^{j} p_k)$. Find the j which produces the minimal such r_i' value.
5) Integrate the R_j to R_i data transmission into the distribution strategy along with the required parallel data transmissions.
6) Update s_i, p_i, and r_i values. ■

It is easy to see that the complexity of Algorithm PARALLEL is of order $O(m^2)$ where m is the number of required relations in the query [4].

B. Minimizing Total Time

Given an ordering on the required relations of a simple query, the SERIAL strategy consists of transmitting each relation, starting with R_1, to the next relation in a serial order. The strategy is represented by $R_1 \rightarrow R_2 \rightarrow \cdots \rightarrow R_m \rightarrow R_r$, where R_r is the relation at the result node. There are two cases of the SERIAL strategy. In Case 1 R_r is included in its proper order in the transmission pattern, $R_1 \rightarrow R_2 \rightarrow \cdots \rightarrow R_r \rightarrow \cdots \rightarrow R_m \rightarrow R_r$. In Case 2 R_r is not included in its proper order, $R_1 \rightarrow \cdots \rightarrow R_{r-1} \rightarrow R_{r+1} \rightarrow \cdots \rightarrow R_m \rightarrow R_r$. It was shown in [4] that for a given simple query one of the cases of the SERIAL strategy has minimal total time when the required relations are ordered so that $s_1 \leqslant s_2 \leqslant \cdots \leqslant s_m$.

C. A Simple Query Example

A simple query example is presented to illustrate the use of Algorithm PARALLEL and the SERIAL strategy. Response times and total times will be compared for the initial feasible solution, the Algorithm PARALLEL strategy, and the optimal SERIAL strategy.

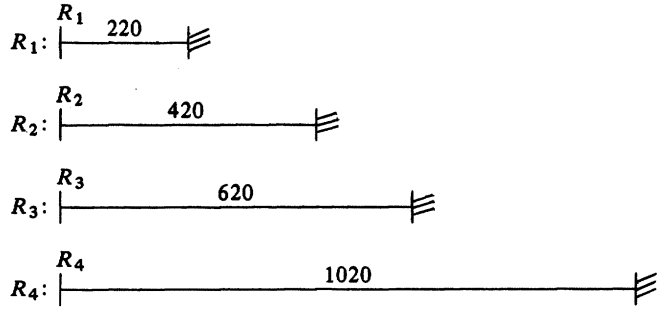
Example 1: Assume a simple query requires data from four relations at four separate nodes. After initial processing, the size and selectivity values are:

	Size (s_i)	Selectivity (p_i)
R_1:	200	0.2
R_2:	400	0.4
R_3:	600	0.6
R_4:	1000	1.0

Assume that the result node is separate from the nodes containing the required relations. Let the transmission time cost be $C(X) = 20 + X$.

Initial Feasible Solution

The cost graph of the initial feasible solution is:



Response time = $r = 1020$.
Total time = $t = 2280$.

Algorithm PARALLEL

Algorithm PARALLEL finds $\hat{r}_i$, the minimum data transmission response time for R_i, in the relation order R_1, R_2, R_3, and R_4.

R_1 response time reduction:

No transmissions to R_1 are considered.

Let $\hat{r}_1 = 220$.

R_2 response time reduction:

Transmit R_1 to R_2: $r_2' = 220 + C(0.2 \cdot 400) = 220 + 100 = 320$.

Since $320 < 420 = r_2$, the transmission from R_1 to R_2 is integrated into the strategy. Let $\hat{r}_2 = 320$.

R_3 response time reduction:

Transmit R_2 to R_3: $r_3' = 320 + C(0.08 \cdot 600) = 320 + 68 = 388$.

Transmit R_1 to R_3: $r_3' = 220 + C(0.2 \cdot 600) = 220 + 140 = 360$.

Since $360 < 388 < 620 = r_3$, the data transmission from R_1 to R_3 is integrated into the strategy. Let $\hat{r}_3 = 360$.

R_4 response time reduction:

Transmit R_3 to R_4: $r_4' = 360 + C(0.048 \cdot 1000) = 360 + 68 = 428$.

Transmit R_2 to R_4: $r_4' = 320 + C(0.08 \cdot 1000) = 320 + 100 = 420$.

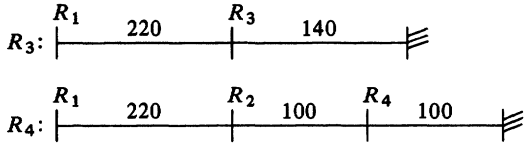
Transmit R_1 to R_4: $r_4' = 220 + C(0.2 \cdot 1000) = 220 + 220 = 440$.

TABLE II
EXAMPLE 1: STRATEGY TIMES

	Response Time	Total Time
Initial Feasible Solution	1020	2280
Algorithm Parallel Strategy	420	780
Serial Strategy	456	456

Since $420 < 428 < 440 < 1020 = r_4$, the data transmission from R_2 to R_4 is integrated into the strategy.

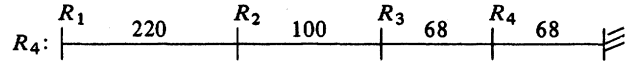
The final cost graph is:



Response time = $r = 420$.
Total time = $t = 780$.

SERIAL Strategy

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



Response time = $r = 456$.
Total time = $t = 456$.

Table II shows the times for the three distribution strategies of this query. ■

IV. A GENERALIZED ALGORITHM

While the optimization algorithms presented in the previous section work for special situations, they do not necessarily derive efficient distribution strategies for general queries. The basic distribution tactics of Algorithm PARALLEL and the ordered SERIAL strategy can, however, be extended to a general query environment. In this general environment the assumptions that have been made on the distributed system still hold but the restriction that only simple queries be considered is removed.

In a general query, each required relation may have any number of joining domains and output domains. Thus, after initial processing, $\alpha_i \geq \beta_i \geq 1$ for all $i = 1, \cdots, m$. Each node may contain any number of required relations. We define the parameter δ to represent the number of internodal joining domains in the query. For each common joining domain, we shall assume transitivity of joining paths. In other words, if a common joining domain exists on n relations at separate nodes, then data may be transmitted between any two of these relations for a JOIN RESTRICTION. For simplicity in analysis, joining domains within each relation are assumed to be independent. Thus a selectivity reduction on one domain does not affect the selectivity of the other joining domains.

The general algorithm is a heuristic that uses an improved exhaustive search to find efficient distribution strategies. Each relation R_i is examined in a small to large order to find a schedule that has a minimal response time value $\hat{r}_i$ or a minimal total time value $\hat{t}_i$, depending upon the declared cost objective. Each joining domain in the relation R_i is handled separately because of the assumption of domain independence. When the minimal time schedules are found for each joining domain, the algorithm integrates these schedules into the overall schedule for relation R_i. The distribution strategy is then constructed by synchronizing the schedules of all required relations in the query. To construct the schedules of the distribution strategy, the algorithm considers only the transmission of join domains between nodes that reduce the size and further transmission cost of the receiving relation by JOIN RESTRICTIONS. The minimal cost schedule for each joining domain is marked with an asterisk (*). However, the algorithm must maintain other candidate schedules containing cost beneficial data transmissions since these schedules may lead to more beneficial cost reductions on the other relation schedules. In fact, this can be viewed as a type of "active backtracking."

We define the following system parameters of the query after initial processing. There are m required relations. Each relation R_i has size s_i and β_i joining domains. Each joining domain d_{ij} of R_i has u_{ij} distinct values and a selectivity of p_{ij}. The algorithm uses a connectivity matrix to represent the joining paths between joining domains. Each candidate schedule for joining domain d_{ij} has the following parameters:

- s'_{ij} reduced size of relation R_i,
- b'_{ij} reduced size of distinct values in domain d_{ij},
- p'_{ij} accumulated incoming selectivity to domain d_{ij},
- γ'_{ij} schedule delay time,
- sch'_{ij} schedule representation consisting of the pattern of data transmissions to relation R_i.

Recall that when the joining domain d_{kl} is transmitted on a joining path to domain d_{ij} the accumulated selectivity p'_{kl} (Πp_{uv} such that d_{uv} is in the d_{kl} schedule) of d_{kl} is used to reduce the size of relation R_i. However, the selectivity value p_{ij} cannot be used to reduce the size of R_i by definition. Thus, we define the notation

$$p'_{kl\backslash ij} = \begin{cases} p'_{kl} \text{ if } d_{ij} \text{ is not in } d_{kl} \text{ schedule} \\ p'_{kl}/p_{ij} \text{ if } d_{ij} \text{ is in } d_{kl} \text{ schedule.} \end{cases}$$

The general algorithm is now presented. For response time minimization parallel data transmissions on a common joining domain will be considered, as was done in Algorithm PARALLEL. For total time minimization no parallel data transmission will be considered in candidate schedules.

Algorithm G

1) (Initialization.) After initial local processing, order the relations so that $s_1 \leq s_2 \leq \cdots \leq s_n$. For each joining domain d_{ij} of each relation R_i, initialize the parameters of the first marked schedule as:

$$s^*_{ij} \leftarrow s_i,$$
$$b^*_{ij} \leftarrow b_{ij},$$
$$p^*_{ij} \leftarrow 1,$$

$\gamma^*_{ij} \leftarrow 0$,
$sch^*_{ij} \leftarrow \Lambda$.

2) Repeat Step 3) for each R_i, $i = 1, \cdots, m$, then GO TO Step 4).

3) (Build candidate schedules for R_i.) For relation R_i repeat the following for each joining domain d_{ij}:

For each incoming domain d_{kl}, examine each candidate schedule of d_{kl} that has not been considered previously for a cost beneficial data transmission to d_{ij}.

If $((C(b'_{kl}) + \gamma'_{kl}) < \gamma^*_{ij})$ or $((p_{kl} \cdot p'_{kl\backslash ij}) < p^*_{ij})$ then build a new candidate schedule for d_{ij} if $(C(b'_{kl}) + \gamma'_{kl}) < (C(s^*_{ij}) + \gamma^*_{ij})$.

The parameters of the new candidate schedule are:

$s'_{ij} \leftarrow p_{kl} \cdot p'_{kl\backslash ij} \cdot s_{ij}$,
$b'_{ij} \leftarrow p_{kl} \cdot p'_{kl\backslash ij} \cdot b_{ij}$,
$p'_{ij} \leftarrow p_{kl} \cdot p'_{kl\backslash ij}$,
$\gamma'_{ij} \leftarrow C(b'_{kl}) + \gamma'_{kl}$,
$sch'_{ij} \leftarrow sch_{kl} | d_{kl}$.

For the new candidate schedule, if $(C(s'_{ij}) + \gamma'_{ij}) < (C(s^*_{ij}) + \gamma^*_{ij})$, then mark the new candidate schedule and remove the asterisk from the previously marked candidate schedule.

After all d_{ij} candidates have been constructed, eliminate all candidates for which $(C(b'_{ij}) + \gamma'_{ij}) \geq (C(b^*_{ij}) + \gamma^*_{ij})$ and $(p'_{ij} \geq p^*_{ij})$.

4) (Integrate schedules.) Eliminate all unmarked candidate schedules for all joining domains. For each relation R_i, $i = 1, \cdots, m$, integrate its schedules over all joining domains by considering the best parallel data transmissions to reduce the required cost objective, response time or total time.

5) (Build strategy.) Eliminate the schedules for relations with no output domains and all of whose joining domains have been transmitted to another relation in the distribution strategy. ■

The feature of Algorithm G that makes the optimization computationally efficient is that data transmissions are quickly eliminated from consideration if they cannot possibly be part of a minimal time schedule. Thus only the potentially cost beneficial data transmissions are examined. In almost all cases the number of these transmissions is considerably smaller than the number of all possible data transmissions.

Algorithm G can be implemented using either the minimization of schedule response time or schedule total time as its cost objective. The difference in implementation would be found in Step 3) of the algorithm when each schedule candidate is examined as a potentially beneficial transmission. For response time all possible parallel data transmissions for that common joining domain are included and contribute to the selectivity of the considered transmission. For total time these parallel transmissions are not included. In almost all cases, considering parallel transmissions to minimize the response of a schedule increases the complexity of Algorithm G by a significant amount while the decrease in schedule response time is limited. The point will be illustrated in Section V. For this reason, the analysis of Algorithm G is carried out only for total time minimization.

The complexity of Algorithm G while minimizing total time can be measured by the number of candidate schedules that must be constructed for a given query. Since the algorithm uses an improved exhaustive search, in the worst case we can find an upper bound on algorithm complexity by considering a query with the following characteristics. Assume that all m relations are joined on all δ joining domains in a given query. In Algorithm G there are four embedded loops in Steps 2) and 3). Thus for a worst case situation the algorithm would construct one new candidate for each relation (m), for each joining domain (δ), for each incoming domain (m), and for each possible domain candidate (m). That is, the algorithm requires δm^3 schedule constructions in this situation. Therefore, Algorithm G has an upper complexity bound of $0(\delta m^3)$ where m is the number of required relations for a general query.

A lower bound on algorithm complexity would occur when a query requires no candidate schedules other than the initial feasible schedules to be constructed. In this case the complexity is $0(m)$, since one schedule must be constructed for each required relation (m).

Consider now an "average" situation. Even for a worst case type query in which all m relations are joined on all δ joining domains, Algorithm G would eliminate most nonbeneficial candidate schedule data transmissions. Assume that, on average, for each incoming joining domain a constant c number of candidates were found to require a new schedule to be constructed. The algorithm would then require $\delta c m^2$ schedule constructions to derive a distribution strategy. Thus for a general query, Algorithm G is, on average, of order $0(\delta m^2)$.

For schedule response time minimization, Algorithm G derives the identical distribution strategies for simple queries as does Algorithm PARALLEL. Algorithm PARALLEL can be seen to be a degenerate case of Algorithm G in which simple query features allow a provably minimal response time distribution strategy to be found. Only a restricted set of data transmissions are potentially cost beneficial for simple queries. Thus, the extensive looping required in Steps 2) and 3) of Algorithm G can be reduced to the simple looping of Steps 3) and 4) in Algorithm PARALLEL. The backtracking feature of Algorithm G is not required for simple queries. Also, since each relation has only one domain in a simple query, Algorithm PARALLEL implicitly eliminates relation schedules whenever the data of that relation are transmitted in the distribution strategy. These simplifications reduce Algorithm G to Algorithm PARALLEL. Therefore, for simple queries, Algorithm G derives minimal response time distribution strategies.

V. An Illustrative Example

In order to illustrate Algorithm G, let us consider a distributed database with four relations at separate network nodes.

Relation	Variable
EMPLOYEE (*E#*,ENAME,SEX)	*E*
COURSE (*C#*,CNAME,LEVEL)	*C*
STUDENT-COURSE (*E#*,*C#*)	*SC*
TEACHER-COURSE (*E#*,*C#*,ROOM)	*TC*

The multivariable query illustrated in Fig. 2 is entered into

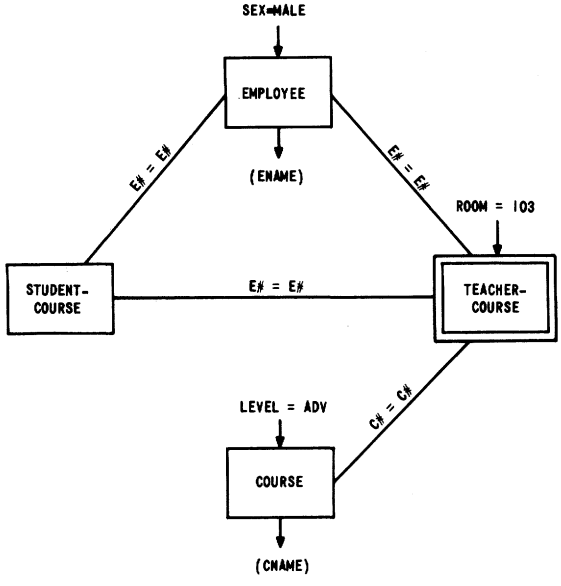


Fig. 2. Query for Example 2.

the system. Assume that the node containing the TEACHER-COURSE relation is at the result node (indicated by the double box in Fig. 2). The query can be stated as follows: for all male employees who are teaching advanced courses in Room 103 and are students in at least one course, list the employees' names and the courses they are teaching.

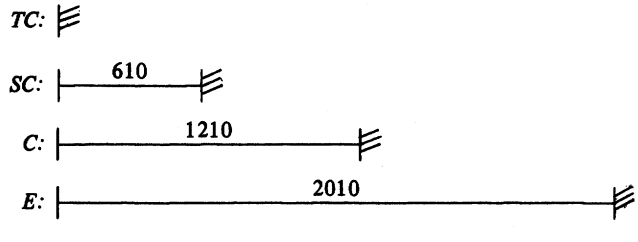
The first step of all distribution strategies is to do local processing. The local restrictions on *E*.SEX, *C*.LEVEL, and *TC*.ROOM are performed and the required joining domains and output domains *E*.ENAME and *C*.CNAME are projected. For convenience, number the relations in order of size s_i and let domain d_{i1} represent *E#* and domain d_{i2} represent *C#*. Assume the sizes and selectivities are:

		$d_{i1}(E\#)$		$d_{i2}(C\#)$	
Relation	s_i	b_{i1}	p_{i1}	b_{i2}	p_{i2}
R_1 – *TC*:	600	200	$\frac{1}{5}$	200	$\frac{1}{2}$
R_2 – *SC*:	600	600	$\frac{3}{5}$		
R_3 – *C*:	1200			100	$\frac{1}{4}$
R_4 – *E*:	2000	200	$\frac{1}{5}$		

Let $C(X) = 10 + X$.

Initial Feasible Solution

The cost graph of the initial feasible solution is:



Response time = 2010.
Total time = 3830.

Algorithm G for Response Time

Algorithm G examines all incoming joining domain candidate schedules for each relation from small to large values of s_i. In the schedule representation sch_{ij}, the notation $d_{ij} + d_{kl}$ indicates a parallel data transmission from d_{ij} and d_{kl}.

1) Examine relation TC: On joining domain d_{11}, the initial candidate schedule is built as described in Step 1) of the algorithm.

	s^*_{11}	b^*_{11}	p^*_{11}	γ^*_{11}	sch^*_{11}
*	600	200	1	0	Λ

Each incoming candidate schedule on the *E#* joining domain is tested for cost benefit. The transmission of domain d_{21} is not potentially beneficial since the test $C(b_{21}) + \gamma_{21} < C(s_{11}) + \gamma_{11}$ or $600 < 600$ is not satisfied. The transmission of domain d_{41} is potentially beneficial since all the tests of Step 3) of the algorithm are satisfied. Thus, a new candidate schedule for d_{11} is built. The parameters of this schedule are:

$$s'_{11} \leftarrow \tfrac{1}{5} \cdot 600 = 120.$$
$$b'_{11} \leftarrow \tfrac{1}{5} \cdot 200 = 40.$$
$$p'_{11} \leftarrow \tfrac{1}{5} \cdot 1 = \tfrac{1}{5}.$$
$$\gamma'_{11} \leftarrow C(200) = 210.$$
$$sch'_{11} \leftarrow d_{41}.$$

The new candidate is compared with the marked candidate for the most cost benefit. In this case the new candidate is more beneficial since

$$(C(s'_{11}) + \gamma'_{11}) < (C(s^*_{11}) + \gamma^*_{11}) \quad \text{or} \quad 130 + 210 < 610.$$

Therefore, the new schedule becomes the marked candidate.

In summary, the candidate schedules for domain d_{11} are:

	s_{11}	b_{11}	p_{11}	γ_{11}	sch_{11}
	600	200	1	0	Λ
*	120	40	$\frac{1}{5}$	210	d_{41}

On joining domain d_{12}, only one potential data transmission exists from d_{32}. This transmission is potentially beneficial since the tests of Step 3) of the algorithm are passed. The parameters of the new candidate schedule are:

$$s'_{12} \leftarrow \tfrac{1}{4} \cdot 600 = 150.$$
$$b'_{12} \leftarrow \tfrac{1}{4} \cdot 200 = 50.$$
$$p'_{12} \leftarrow \tfrac{1}{4} \cdot 1 = \tfrac{1}{4}.$$
$$\gamma'_{12} \leftarrow C(100) = 110.$$
$$sch'_{12} \leftarrow d_{32}.$$

This new candidate is more beneficial than the initial marked candidate since $(C(s'_{12}) + \gamma'_{12}) < (C(s^*_{12}) + \gamma^*_{12})$ or $160 + 110 < 610$.

In summary, the candidate schedules for domain d_{12} are:

	s_{12}	b_{12}	p_{12}	γ_{12}	sch_{12}
(deleted)	600	200	1	0	Λ
*	150	50	$\frac{1}{4}$	110	d_{32}

The first candidate schedule can be eliminated from further consideration since $(C(b'_{12}) + \gamma'_{12}) > (C(b^*_{12}) + \gamma^*_{12})$ and $p'_{12} > p^*_{12}$. Only the marked candidate remains for domain d_{12}.

2) Examine relation SC: On joining domain d_{21}, the initial candidate schedule is:

	s^*_{21}	b^*_{21}	p^*_{21}	γ^*_{21}	sch^*_{21}
*	600	600	1	0	Λ

Four potential data transmissions to relation *SC* are tested. The two candidate schedules from domain d_{11} are considered for transmission. Candidate schedules are built for both.

For the data transmission of d_{11} to d_{21}, the parameters are:

$$s'_{21} \leftarrow \tfrac{1}{5} \cdot 600 = 120.$$
$$b'_{21} \leftarrow \tfrac{1}{5} \cdot 600 = 120.$$
$$p'_{21} \leftarrow \tfrac{1}{5} \cdot 1 = \tfrac{1}{5}.$$
$$\gamma'_{21} \leftarrow C(200) = 210.$$
$$sch'_{21} \leftarrow d_{11}.$$

For the data transmission of the marked candidate schedule (d_{41}, d_{11}) to d_{21}, the selectivity becomes $\frac{1}{5} \cdot \frac{1}{5} = \frac{1}{25}$. The parameters of the new candidate are:

$$s'_{21} \leftarrow \tfrac{1}{25} \cdot 600 = 24.$$
$$b'_{21} \leftarrow \tfrac{1}{25} \cdot 600 = 24.$$
$$p'_{21} \leftarrow \tfrac{1}{25} \cdot 1 = \tfrac{1}{25}.$$
$$\gamma'_{21} \leftarrow C(40) + 210 = 260.$$
$$sch'_{21} \leftarrow d_{41} | d_{11} = d_{41}, d_{11}.$$

The candidate schedules are compared. The candidate schedule $sch'_{21} = (d_{41}, d_{11})$ is the most beneficial and is marked.

The data transmission from d_{41} to d_{21} is tested. A new candidate is not built. In order to minimize schedule response time, Algorithm G will also test parallel data transmissions. The parallel transmission of d_{11} and d_{41} to d_{21} is tested and a new schedule candidate is built. The parameters are:

$$s'_{21} \leftarrow \tfrac{1}{25} \cdot 600 = 24.$$
$$b'_{21} \leftarrow \tfrac{1}{25} \cdot 600 = 24.$$
$$p'_{21} \leftarrow \tfrac{1}{25} \cdot 1 = \tfrac{1}{25}.$$
$$\gamma'_{21} \leftarrow C(200) = 210.$$
$$sch'_{21} \leftarrow d_{11} + d_{41}.$$

After testing, this parallel schedule is found to be the most beneficial and marked.

In summary, the candidate schedules for domain d_{21} are:

	s_{21}	b_{21}	p_{21}	γ_{21}	sch_{21}
(deleted)	600	600	1	0	Λ
(deleted)	120	120	$\frac{1}{5}$	210	d_{11}
(deleted)	24	24	$\frac{1}{25}$	260	d_{41}, d_{11}
*	24	24	$\frac{1}{25}$	210	$d_{11} + d_{41}$

The first three schedule candidates can be eliminated from further consideration since in each case $(C(b'_{21}) + \gamma'_{21}) > (C(b^*_{21}) + \gamma^*_{21})$ and $p'_{21} \geq p^*_{21}$. Only the marked candidate remains for domain d_{21}.

3) Examine relation C: On joining domain d_{32}, a candidate schedule is built for the data transmission from the marked schedule of d_{12}.

	s_{32}	b_{32}	p_{32}	γ_{32}	sch_{32}
	1200	100	1	0	Λ
*	600	50	$\frac{1}{2}$	170	d_{32}, d_{12}

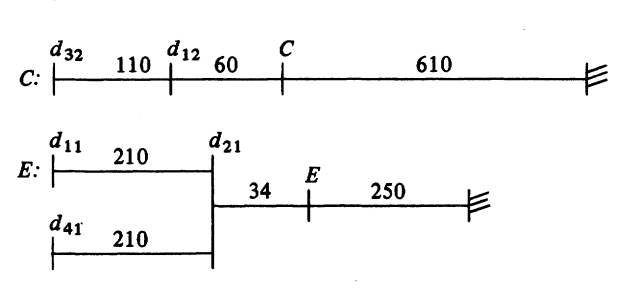
4) Examine relation E: On joining domain d_{41}, four candidate schedules are built.

	s_{41}	b_{41}	p_{41}	γ_{41}	sch_{41}
	2000	200	1	0	Λ
	400	40	$\frac{1}{5}$	210	d_{11}
(deleted)	240	24	$\frac{3}{25}$	610	$d_{11} + d_{21}$
*	240	24	$\frac{3}{25}$	252	$(d_{11} + d_{41}), d_{21}$

The third candidate schedule is eliminated from further consideration.

A second loop through the relations finds no further new candidate schedules. Therefore, the algorithm integrates the marked schedules into a distribution strategy. No schedule is needed for relation *TC* since it is at the result node. Since relation *SC* consists of only one joining domain that is transmitted to relation *E* in the strategy, its schedule can be eliminated.

The final distribution strategy is



Response time = 780.
Total time = 1484.

Algorithm G for Total Time

For total time schedule minimization, Algorithm G does not consider parallel data transmission of candidate schedules. Thus, the processing steps are similar to Algorithm G for response time except fewer candidates are built. The candidate schedules built for each relation joining domain are:

1) Relation TC:
Domain d_{11}.

	s_{11}	b_{11}	p_{11}	γ_{11}	sch_{11}
	600	200	1	0	Λ
*	120	40	$\frac{1}{5}$	210	d_{41}

Domain d_{12}.

	s_{12}	b_{12}	p_{12}	γ_{12}	sch_{12}
(deleted)	600	200	1	0	Λ
*	150	50	$\frac{1}{4}$	110	d_{32}

2) Relation SC:
Domain d_{21}.

	s_{21}	b_{21}	p_{21}	γ_{21}	sch_{21}
(deleted)	600	600	1	0	Λ
(deleted)	120	120	$\frac{1}{5}$	210	d_{11}
*	24	24	$\frac{1}{25}$	260	d_{41}, d_{11}

3) Relation C:
Domain d_{32}.

	s_{32}	b_{32}	p_{32}	γ_{32}	sch_{32}
	1200	100	1	0	Λ
*	600	50	$\frac{1}{2}$	170	d_{32}, d_{12}

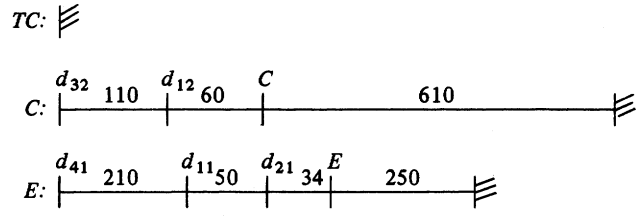
4) Relation E:
Domain d_{41}.

	s_{41}	b_{41}	p_{41}	γ_{41}	sch_{41}
	2000	200	1	0	Λ
	400	40	$\frac{1}{5}$	210	d_{11}
*	240	24	$\frac{3}{25}$	302	d_{41}, d_{11}, d_{21}

A second loop finds no new candidate schedules. The algorithm integrates the marked schedules into a distribution strategy, eliminating the schedule for relation *SC* as before. The final distribution strategy is:

TABLE III
PROCESSING TIMES FOR THE DISTRIBUTION STRATEGIES

	Response Time	Total Time
Initial Feasible Solution	2010	3830
Algorithm G (Response Time)	780	1484
Algorithm G (Total Time)	780	1324



Response time = 780.
Total time = 1324.

We note in this example that the response time case required 14 candidate schedules while the total time case required 12 candidate schedules. Fewer candidate schedule constructions allow more efficient processing. Another advantage of the total time algorithm is that in most cases implementing a total time strategy on a network requires less synchronization and message passing between nodes.

Table III shows the times for the three distribution strategies of this example.

The most striking observation in Table III is the large time differences between the initial feasible solution and the other two distribution strategies. The use of an optimization algorithm can be concluded to be an important part of a distributed database management system. For this example, the total time strategy of Algorithm G performs best in both cost measures.

VI. IMPLEMENTATION

A distributed database system contains four major subsystems: the query processing subsystem, the integrity subsystem, the scheduler subsystem, and the reliability subsystem.

The query processing subsystem which implements the query optimization algorithm is highly interdependent with other subsystems in the DDBMS. This interdependence is illustrated in Fig. 3. In this section the interactions between distributed query processing using an optimization algorithm such as Algorithm G and other distributed system functions are discussed.

Consider a query submitted to the query processing subsystems. Before the query can be processed by an optimization algorithm the subset of the database required to satisfy the query must be determined. Once the data needs (i.e., the required relations) of the query are recognized by query analysis, the location of the data in the network must be found by using a data directory. Since data may be stored redundantly at network nodes, a specific nonredundant materialization [5] of the required data must be assigned to each query for its processing. The optimization algorithm uses a specific data

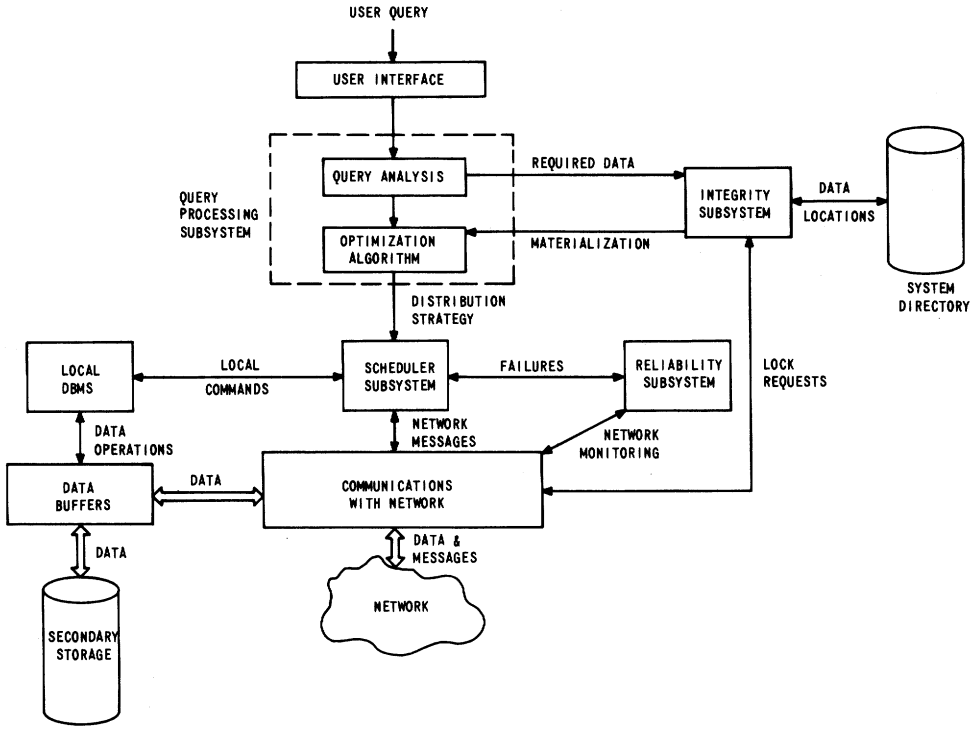


Fig. 3. Query processing in a distributed database system.

materialization to design an efficient distribution strategy for each query.

The subsystem that is responsible for assigning materializations to queries is the *integrity subsystem.* This subsystem uses the system data directory to find the most cost beneficial copies of data to include in a query's materialization. At the same time, however, this subsystem is responsible for maintaining database integrity and consistency.

Queries on a distributed system may update data in the database. The integrity subsystem must handle the problem of synchronizing update queries on the redundant data in the network. All redundant copies of data must be updated correctly and consistently by update queries; and retrieval queries should be guaranteed correct, up-to-date data at the time a materialization is assigned.

Update synchronization is controlled in nondistributed systems by data locking rules. For consistency, most update queries require exclusive locking on all copies of the data to be modified while retrieval queries require a share lock on just the copy of data it is using [9]. However, in a distributed database environment the time delays introduced by transmitting this locking information throughout the network is often substantial and unmanageable. The integrity subsystem must solve the problem of controlling concurrent queries so that database integrity and consistency is maintained while the communication overhead of transmitting control information among network nodes is minimized.

Once the query processing subsystem receives a materialization for each query, the optimization algorithm (such as Algorithm G) is called upon to derive a distribution strategy for the query. The distribution strategy is passed to a subsystem that schedules the local processing and data transmissions of the strategy.

One function of the *scheduler subsystem* is to send commands to the local DBMS's in the network in order to do local data processing. In addition, the scheduler must interact with the network communication facilities at each node in order to transmit data between nodes as determined by the distribution strategy. The scheduler coordinates the various schedules in the strategy so that the query response is presented at the result node. It is easily recognized that a complex distribution strategy will require considerable network coordination of transmissions and local processing. This, in turn, requires increased message passing by the scheduler. Often simple distribution strategies are beneficial in a distributed system because the scheduler is required to transmit fewer messages on the network. This consideration must be kept in mind when optimization algorithms are designed. Complex, optimal distribution strategies may be suboptimal when the coordination requirements (i.e., message passing) of implementing the complex strategy are considered. This is a major part of the rationale behind constructing Algorithm G to minimize the total time of relation schedules as its cost objective. Parallel data transmissions in a schedule often require considerable coordination in return for little reduction in schedule response time.

One of the primary advantages of a distributed database system is system reliability. With the storage of redundant data at separate nodes, even though single system components such as nodes and communication lines may fail, specific data items

will still be available at other nodes. The *reliability subsystem* of a DDBMS continuously monitors system components for failures. In the event of failures, control information is distributed in the network so that normal operations may continue while the failed component is being repaired. The reliability subsystem then attempts to integrate the repaired component back into the system in a state of database consistency.

A component failure affects query processing when the failed node contains data that are part of a query's materialization. Since the distribution strategy was derived on that specific materialization, the strategy may no longer be valid. Two alternatives exist: to wait for the failed node to be repaired and continue the original distribution strategy; or to back out the query and restart it for a new materialization and new distribution strategy. The latter alternative would seem to be preferable in most cases.

The complete design of a distributed database system requires a thorough understanding of interactions among its subsystems. The implementation of a distribution algorithm such as Algorithm G for query processing is an integral part of the DDBMS development. The efficiency of the query processing subsystem has a strong influence on the overall DDBMS efficiency.

VII. Conclusion

Efficient query processing is an essential function of a distributed database system. In this paper a general algorithm (Algorithm G) is developed that will derive a distribution strategy for a general distributed query. Algorithm G is developed in a two-step approach. In a previous paper [4], two simple algorithms (Algorithm PARALLEL for response time and the ordered SERIAL strategy for total time) were proven to produce minimal time distribution strategies for a special class of queries.

Algorithm G is shown to be a straightforward extension of these simple algorithms to a general query environment. It is computationally efficient for average distributed queries. An example illustrated that the distribution strategies derived from Algorithm G are much more cost effective than an initial feasible distribution strategy in which no data access optimization is done.

It is clear that the query processing subsystem embodying the optimization algorithm cannot function alone in a distributed database system. The interdependency of the query processing subsystem with other subsystems is discussed. An especially critical phase of query processing is the actual implementation of the derived distribution strategy on the network by the scheduler subsystem. A distribution strategy that requires extensive message passing on the network will adversely affect the performance of the entire system. In a distributed database system, the need for data access optimization is evident, however, at the same time, distribution strategies should be kept as simple as possible for efficient implementation.

Algorithm G is developed in such a way that its use in the query processing subsystem is efficient. In addition, the derived distribution strategies of Algorithm G are simple enough that they can be executed efficiently on the distributed system.

Appendix

The formal proofs that 1) Algorithm PARALLEL minimizes the response time for a simple query and 2) the SERIAL strategy minimizes the total time for a simple query are detailed in [4]. A brief outline of these proofs will be given here.

A. Minimizing Response Time

We prove that Algorithm PARALLEL finds a distribution strategy for a simple query that is optimal in the sense of having a minimal response time. The basis of the proof is found in the following Lemma. Recall that $\hat{r}_i$ is the minimal response time for the schedule of relation R_i.

Lemma 1: For a simple query if, after initial processing, the required relations $R_1, \cdots, R_m$ are ordered so that $s_1 \leqslant s_2 \leqslant \cdots \leqslant s_m$ then $\hat{r}_1 \leqslant \hat{r}_2 \leqslant \cdots \leqslant \hat{r}_m$. ∎

This lemma is proved by induction on relation order $i = 1$ to $i = m$. From the data transmission cost assumption, if $s_i \leqslant s_j$ then $C(s_i) \leqslant C(s_j)$. Initially $r_i = C(s_i)$ for all i. Therefore it is clear that $\hat{r}_1 = C(s_1)$ and $\hat{r}_1 \leqslant \hat{r}_j$ for all $j = 1, \cdots, m$. For an arbitrary i the minimum response time schedule of R_i is studied. Two cases are considered depending upon whether the schedule of R_{i-1} is part of the schedule of R_i or not. For both cases it is shown that $\hat{r}_{i-1} \leqslant \hat{r}_i$.

Theorem 1: For a simple query, if, after initial processing, the required relations $R_1, \cdots, R_m$ are ordered so that $s_1 \leqslant s_2 \leqslant \cdots \leqslant s_m$ then Algorithm PARALLEL will derive a minimum response time distribution strategy. ∎

The proof shows inductively that Algorithm PARALLEL correctly derives minimum response time schedules for each R_i, $i = 1, \cdots, m$. The result of Lemma 1 limits the search space for cost beneficial data transmissions so that to reduce the response time of the R_i schedule, only transmissions from relations R_j, $j < i$, when $r_j = \hat{r}_j$, need be considered. Algorithm PARALLEL is shown to implement a strategy that finds $\hat{r}_i$ from $i = 1$ to $i = m$. Then from Lemma 1 the response time for a simple query $r = \max_i(r_i) = \max_i(\hat{r}_i) = \hat{r}_m$. Since the response time of any distribution strategy must be at least $\hat{r}_m$, $r = \hat{r}$ as required.

B. Minimizing Total Time

We prove that the SERIAL strategy for simple queries is optimal in the sense of having a minimal total time. We define a serial schedule for a relation to be a schedule in which there are no parallel data transmissions. 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 at relation R_i are identical. Three lemmas are used to prove that the ordered serial strategy finds the minimal total time for a simple query.

Lemma 2: Given a serial schedule $\bar{Q} = R_{i_1}, \cdots, R_{i_l}$, if $i_j = i_k$ for some $j < k$ then the schedules $\bar{Q}$ and $\bar{Q}' = R_{i_1}, \cdots, R_{i_j}, \cdots, R_{i_{k-1}}, R_{i_{k+1}}, \cdots, R_{i_l}$ are equivalent and $\text{COST}(\bar{Q}') < \text{COST}(\bar{Q})$. ∎

This lemma follows directly from the definition of relation selectivity and size reduction. Once a relation's selectivity is included in the accumulated selectivity of a serial schedule, another transmission on the same relation can cause no further selectivity or size reduction.

Lemma 3: Given a nonserial schedule containing at least one instance of parallel data transmission, an equivalent serial schedule $\bar{Q}$ exists such that COST($\bar{Q}$) is less than or equal to the total time cost of the nonserial schedule. ■

It is proved in a straightforward manner that any instance of data transmissions occurring in parallel can be transformed into a sequence of serial transmissions which has a total time less than or equal to the parallel transmissions.

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

Lemma 4: For a simple query if, after initial processing, the required relations $R_1, \cdots, R_m$ are ordered so that $s_1 \leqslant s_2 \leqslant \cdots \leqslant s_m$ then the serial schedule $\bar{Q} = R_{i1}, \cdots, R_{il}$, where $i_j < i_{j+1}$ for all $j = 1, \cdots, l-1$, has the minimum total cost among all equivalent schedules. ■

Theorem 2 can now be proved by the use of Lemmas 2-4.

Theorem 2: For a simple query if, after initial processing, the required relations $R_1, \cdots, R_m$ are ordered so that $s_1 \leqslant s_2 \leqslant \cdots \leqslant s_m$ then the SERIAL strategy has the minimum total time. Case 1 of the 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 SERIAL strategy is optimal. ■

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 minimal total time. By the use of Lemmas 2-4 any such optimal distribution strategy has an equivalent SERIAL strategy with less or equal total time. Thus the SERIAL strategy must have minimal total time.

There are two possible cases of the SERIAL strategy. Which case has the minimum total time must be tested. From the definition of the SERIAL strategy, the total time for Case 1, where R_r is included in the ordered SERIAL strategy, is

$$\sum_{i=1}^{m} C\left(s_i \cdot \prod_{j=1}^{i-1} p_j\right).$$

For Case 2, where R_r is not included in the SERIAL strategy, the total time is

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

Case 1 of the SERIAL strategy is optimal if:

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

Algebraic manipulation of this equation results in the formula given in Theorem 2.

References

[1] E. F. Codd, "A relational model of data for large shared data banks," *Commun. Ass. Comput. Mach.*, vol. 13, pp. 377–387, June 1970.

[2] Computer Corporation of America, "A distributed database management system for command and control applications: Semiannual technical report 2," Tech Rep. CCA-78-03, Jan. 30, 1978.

[3] R. Epstein, M. Stonebraker and E. Wong, "Distributed query processing in a relational data base system," in *Proc. ACM 1978 SIGMOID Conf.*, Austine, TX, June 1978, pp. 169–180.

[4] A. R. Hevner and S. B. Yao, "Optimization of data access in distributed systems," Comput. Sci. Dep. Purdue Univ., Tech. Rep. TR281, July 1978.

[5] J. B. Rothnie and N. Goodman, "An overview of the preliminary design of SDD-1: A system for distributed databases," in *Proc. 1977 Berkeley Workshop on Distributed Data Management and Computer Networks*, Lawrence Berkeley Lab., Univ. California, Berkeley, May 1977, pp. 39–57.

[6] —, "A survey of research and development in distributed database management," in *1977 Proc. Very Large Data Bases*, Tokyo, Japan, Oct. 1977, pp. 48–62.

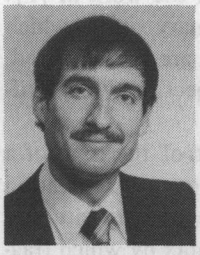
[7] M. Stonebraker and E. Neuhold, "A distributed database version of INGRES," in *Proc. 1977 Berkeley Workshop on Distributed Data Management and Computer Networks*, Lawrence Berkeley Lab., Univ. California, Berkeley, May 1977, pp. 19–36.

[8] E. Wong, "Retrieving dispersed data from SDD-1: A system for distributed databases," in *Proc. 1977 Berkeley Workshop on Distributed Data Management and Computer Networks*, Lawrence Berkeley Lab., Univ. California, Berkeley, May 1977, pp. 217–235.

[9] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger, "The notion of consistency and predicate locks in a database system," *Commun. Ass. Comput. Mach.*, vol. 19, pp. 624–633, Nov. 1976.

[10] E. Wong and K. Youssefi, "Decomposition–A strategy for query processing," *ACM Trans. on Database Systems*, vol. 1, pp. 223–241, Sept. 1976.

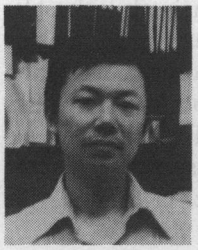
[11] S. B. Yao, "Optimization of query evaluation algorithms," *ACM Trans. on Database Systems*, vol. 4, June 1979, to be published.



Alan R. Hevner received the B.S. and M.S. degrees in computer science from Purdue University, West Lafayette, IN.

He is currently completing the requirements for the Ph.D. degree in computer science at Purdue and is teaching the database systems course in the Department of Computer Science. During the Summer of 1978 he taught database systems as an Adjunct Faculty Member in the Graduate School of Business Administration at New York University. His research and teaching interests are in database systems, distributed systems, and computer system modeling and analysis.

Mr. Hevner is a member of the Association for Computing Machinery.



S. Bing Yao was born in Shanghai, China, in 1944. He received the B.S. degree in mathematics from the National Taiwan University, Taiwan, the M.S. degree in mathematics from the Western Michigan University, Kalamazoo, and the Ph.D. degree in computer communication sciences from the University of Michigan, Ann Arbor, in 1974.

He is currently an Associate Professor of Computer Sciences at Purdue University, West Lafayette, IN, and a consultant at Bell Laboratories, Holmdel, NJ. He has published over 30 technical papers and is currently editing the book *Principles of Database Design.* His current research interests include the design, modeling, and performance evaluation of database systems.

Dr. Yao is a member of the Association for Computing Machinery and the IEEE Computer Society. He is also a member of the Editorial Board of the *ACM Transactions on Database Systems*, and is Program Chairman of the 4th International Conference on Very Large Databases.