[14] M. M. Zloof, "Query-By-Example: A database language," *IBM Syst. J.*, vol. 16, no. 4, pp. 324–343.

Barry E. Jacobs was born in Brooklyn, NY, in 1947. He received the B.S. degree in mathematics from Brooklyn College of the City University of New York and the M.S. and Ph.D. degrees in mathematics from the Courant Institute of Mathematical Sciences (N.Y.U.) under the direction of Prof. M. D. Davis.

He is currently an Assistant Professor of Computer Science at the University of Maryland, College Park. He has published in the area of mathematical logic, in particular, in generalized recursion theory and abstract computational complexity theory. More recently, this "born again" databaser has written over 20 papers generalizing and extending database research from the relational to the heterogeneous case (relational, hierarchical, network) using database logic as a framework.

Cynthia A. Walczak received the B.S. degree in mathematics from the University of Wisconsin, Stevens Point, in 1976 and the M.S. degree in computer science from the University of Maryland, College Park, in 1979.

She is presently a Ph.D. student at the University of Maryland and works at the National Institute of Dental Research in Bethesda, MD. Her research interests include database design and pattern recognition.

Optimization Algorithms for Distributed Queries

PETER M. G. APERS, ALAN R. HEVNER, AND S. BING YAO

Abstract—The efficiency of processing strategies for queries in a distributed database is critical for system performance. Methods are studied to minimize the response time and the total time for distributed queries. A new algorithm (Algorithm GENERAL) is presented to derive processing strategies for arbitrarily complex queries. Three versions of the algorithm are given: one for minimizing response time and two for minimizing total time. The algorithm is shown to provide optimal solutions under certain conditions.

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

I. Introduction

IN a distributed database, we have the ability to decentralize data that are most heavily used by end users at geographically dispersed locations and, at the same time, to combine data from different sources by means of queries. The decentralization of data will result in better response times and, if multiple copies are used, in a more reliable system.

The retrieval of data from different sites in a network is known as distributed query processing. The difference between query processing in a centralized database and a distributed database is the potential for decomposing a query into subqueries which can be processed in parallel, and their intermediate results can be sent in parallel to the required computers. Finding an efficient way of processing a query is important. If a query is processed inefficiently, it not only takes a long time before the end user gets his answer, but it might also decrease the performance of the whole system because of network congestion. We will investigate two optimization objectives: the minimization of response time and of total time. Which of these two objectives is better for a specific system depends upon the system's characteristics.

Distributed query processing has received a great deal of attention [15], [19]. The initial research in this area was done by Wong [24]. He proposed an optimization method based on a greedy heuristic that produces efficient, but not necessarily optimal query processing strategies. An enhanced version of this method is implemented in the SDD-1 system [5]. Epstein *et al.* [9] developed an algorithm based on the query optimization technique of decomposition [23]. This algorithm is implemented in the distributed INGRES database system. Performance studies of this algorithm are reported in [10]. A dynamic optimization algorithm for the POLYPHEME system has been proposed by Toan [17]. Further research on dynamic optimization algorithms has been done by Baldissera *et al.* [3] and Takizawa [22].

Work by Chu and Hurley [7] defined the solution space of feasible processing strategies for distributed queries. They presented an optimal, although inherently exponential, optimization algorithm. The use of the semi-join operation has led to the development of full-reduction methods of processing a

Manuscript received October 1, 1980; revised January 4, 1982 and April 1, 1982.

P.M.G. Apers is with the Informatica Wiskundig Seminarium, Vrije Universiteit, de Boelelaan 1081, HV Amsterdam, The Netherlands.

A. R. Hevner and S. B. Yao are with the College of Business and Management, University of Maryland, College Park, MD 20742.

distributed query [6], [11], [25], [4]. These methods are applicable for a special class of queries known as tree queries. Pelagatti and Schreiber [18] use an integer programming technique to minimize cost in distributed query processing. Kershberg *et al.* [16] apply query optimization to a database allocated on a star network and study the system performance.

For a special class of simple queries, Hevner and Yao developed algorithms PARALLEL and SERIAL [12] that find strategies with, respectively, minimum response time and total time. In [14], they extended these algorithms to Algorithm G that processes general distributed queries. Apers [1] showed that this algorithm had some serious drawbacks. Its complexity for worst case queries does not have a polynomial bound. Also, the analysis of the quality of the derived processing strategies is difficult. Both Hevner [13] and Apers [2] recognized these problems and developed improved algorithms. Here we present this improved algorithm, Algorithm GENERAL. It is more clearly understood and more usable as a distributed query optimization algorithm than Algorithm G.

This paper is organized as follows. In Section II, we will briefly repeat the query processing model described in [13]. Three versions of Algorithm GENERAL, for response time and total time, are presented and analyzed in Section III.

II. Definitions and Distributed System Model

A distributed database system is characterized by the distribution of the system components of hardware, control, and data. For this research, a distributed system is a collection of independent computers interconnected via point-to-point communication lines. Each computer, known as a node in the network, has a processing capability, a data storage capability, and is capable of operating autonomously in the system. Each node contains a version of a distributed DBMS.

The database is viewed logically in the relational data model [8]. The database is allocated across system nodes in units of relations (without loss of generality, relation fragments may be considered as relations for distribution). The relation distribution allows a general manner of redundancy. The only allocation constraint is that all data must be either locally or globally accessible from any system node. The distribution of data on the network is invisible to the user.

We assume that we know the following information about the relations.

For each relation R_i, $i = 1, 2, \cdots, m$,

n_i: number of tuples,

α_i: number of attributes,

s_i: size (e.g., in bytes).

For each attribute $d_{ij}, j = 1, 2, \cdots, \alpha_i$ of relation R_i,

p_{ij}: selectivity,

b_{ij}: size (e.g., in bytes) of the data item in attribute d_{ij}.

The selectivity p_{ij} of attribute d_{ij} is defined as the number of different values occurring in the attribute divided by the number of all possible values of the attribute. Thus, $0 < p_{ij} \leq 1$.

To process a query in a relational database, we only need the operations restriction, projection, and join [8]. In a distributed database, we may need to compute joins on relations which are located at different sites. Instead of computing these joins immediately, we will first reduce the sizes of the relations wherever possible by restrictions and projections. A relation, which is one of the operands of a join, can be made smaller by deleting the tuples that cannot play a role in the join. An operation called a *semi-join* [4] deletes these tuples of the relation. If relation R_i has a semi-join with attribute d_{kl} on attribute d_{ij}, then the parameters of relation R_i are changed in the following way:

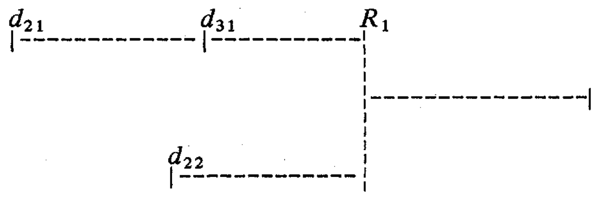
$$s_i \leftarrow s_i * p_{kl},$$

$$p_{ij} \leftarrow p_{ij} * p_{kl},$$

$$b_{ij} \leftarrow b_{ij} * p_{kl}.$$

The selectivity and size of only the joining attribute are reduced because of an assumption of attribute independence within each relation in our model.

To compute a semi-join, the unique values of the joining attribute of one relation are sent to the other relation. It is cheaper to compute this semi-join than the complete join. In the result node, the complete join will be computed after the reduced relations have arrived by concatenating matching tuples on the joining attributes. The data transmissions used for reducing a relation and the transmission of the reduced relation to the query computer form a *schedule* for this relation. An example of a schedule for relation R_i can be seen below.



Attribute d_{21} is sent to attribute d_{31}. A semi-join is performed on relation R_3. The reduced d_{31} attribute is sent to relation R_1 in parallel with attribute d_{22}. Further relational operations reduce the size of relation R_1. Finally, the reduced relation R_1 is sent to the result node.

It is also possible to construct a schedule for an attribute instead of a relation. A schedule for an attribute is constructed to send the attribute to a relation upon which a semi-join will be computed—for example, attribute d_{31} in the above schedule.

We assume that the *transmission cost* of the data is the same between any two computers and is a linear function of the size of the data. This function will be denoted by $C(X) \leftarrow C_0 + C_1 X$ where X is the amount of data transmitted. The *response time* of a schedule is the time elapsed between the start of the first transmission and the time at which the relation (attribute) arrives at the required computer. The *minimum response time* of a relation (attribute) is the minimum response time among all possible schedules for this relation (attribute). The *total time* of a schedule is the sum of the costs of all transmissions required in the schedule.

The *incoming selectivity* of a schedule for a relation (or attribute) is the product of selectivities of all the attributes in the schedule (more than one occurrence of an attribute in a schedule can contribute only one instance of its selectivity) excluding the attributes of the relation (or in the case of an attribute, the attribute itself).

A *distribution strategy* for a query consists of the schedules for all relations which do not reside in the result node and are used in the query. With the assumption that data transmission costs are significantly greater than local processing costs in the system, the cost of processing a query is determined by the transmission costs in its distribution strategy. Another implicit assumption that we make throughout this paper is that the query processing strategy is run on a dedicated system in order to achieve minimum execution times. Dynamic system factors such as communication line contention and subsequent queueing delays are not considered in our static query optimization algorithms.

In a distributed database, it is, in general, better to do local processing first because it reduces the amount of data to be transmitted. With local processing, we mean the computation of restrictions, projections, and semi-joins between relations that reside in the same node. After the initial processing, each node that has query data will be considered to contain only one "integrated" relation. The relations at each node remain distinct (i.e., no Cartesian product relation is formed). However, by reformatting the query so that each node becomes a variable, the distribution aspects of the query are emphasized [24]. Without loss of generality, this assumption provides a distribution abstraction to the problem and simplifies the understanding of our following algorithms.

Initial local processing results in the following parameters:

m: number of relations in the remaining query,

α_i: number of attributes in relation R_i,

β_i: number of internodal joining attributes in relation R_i.

In [12] and [14], Hevner and Yao introduced and investigated algorithms, PARALLEL and SERIAL, which, respectively, compute minimum response and total time schedules for simple queries. Queries are called *simple* if, after initial local processing, the relations contain only one attribute: the joining attribute. Thus, $\alpha_i = \beta_i = 1$ for $i = 1, 2, \cdots, m$.

Algorithm PARALLEL

1) Order relations R_i such that $s_1 \leqslant s_2 \leqslant \cdots \leqslant s_m$.

2) Consider each relation R_i in ascending order of size.

3) For each relation $R_j (j < i)$, construct a schedule to R_i that consists of the parallel transmission of the relation R_j and all schedules of relations $R_k (k < j)$. Select the schedule with minimum response time.

Algorithm SERIAL

1) Order relations R_i such that $s_1 \leqslant s_2 \leqslant \cdots \leqslant s_m$.

2) If no relations are at the result node, then select strategy:

$$R_1 \rightarrow R_2 \rightarrow \cdots \rightarrow R_n \rightarrow \text{result node.}$$

Or else if R_r is a relation at the result node, then there are two strategies:

$$R_1 \rightarrow R_2 \rightarrow \cdots \rightarrow R_r \rightarrow \cdots \rightarrow R_n \rightarrow R_r \quad \text{or}$$

$$R_1 \rightarrow R_2 \rightarrow \cdots \rightarrow R_{r-1} \rightarrow R_{r+1} \rightarrow \cdots \rightarrow R_n \rightarrow R_r.$$

Select the one with minimum total time.

The complexity of algorithm PARALLEL is $O(m^2)$ and that of algorithm SERIAL is $O(m \log_2 m)$ where m is the number of required relations in the query [14].

III. Algorithm GENERAL

A *general query* is characterized by $\alpha_i \geqslant \beta_i \geqslant 1$ for $i = 1, 2, \cdots, m$. This means that a relation can contain more than one joining attribute. Let σ represent the number of joining attributes in a query. Therefore, such a relation can be reduced in size by semi-joins on different joining attributes.

To illustrate our query optimization methods, we will use a database consisting of the following relations:

PARTS($P\#$, PNAME)
ON-ORDER($S\#$, $P\#$, QTY)
S-P-J($S\#$, $P\#$, $J\#$).

The query represented by Fig. 1 is: "List the $P\#$, PNAME, and total quantity for all parts that are currently on order from suppliers who supply that part to jobs 10 or 20."

In this query, there are two joining attributes, $P\#$ and $S\#$. Assume that each relation is located at a different node and that the result is required at a fourth node. After performing the restriction on the *S-P-J* relation, the required attributes in each relation are projected. The resulting size and selectivity parameters are given in Table I.

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

The response time and total time costs of the different query processing strategies for this query will be compared for the response time and total time versions of algorithm GENERAL.

A simple way of processing a query is to perform initial local processing, followed directly by the transmissions of all remaining data to the result node, where centralized query processing builds the result. This will be called the Initial Feasible Solution (IFS).

```
R1: ON-ORDER   R1  1020
               |---------|

R2: S-P-J      R2  2020
               |-------------------|

R3: PARTS      R3  3020
               |-----------------------------|
```

Response time = 3020. Total time = 6060.

We now present an outline of Algorithm GENERAL. Minimization of response time and total time is done by three different versions of the algorithm, which are discussed in the next three sections.

Algorithm GENERAL

1) *Do all initial local processing.*

2) *Generate candidate relation schedules.* Isolate each of the σ joining attributes, and consider each to define a *simple query* with an undefined result node.

a) To minimize *response time*, apply Algorithm PARALLEL to each simple query. Save all candidate schedules for integration in step 3.

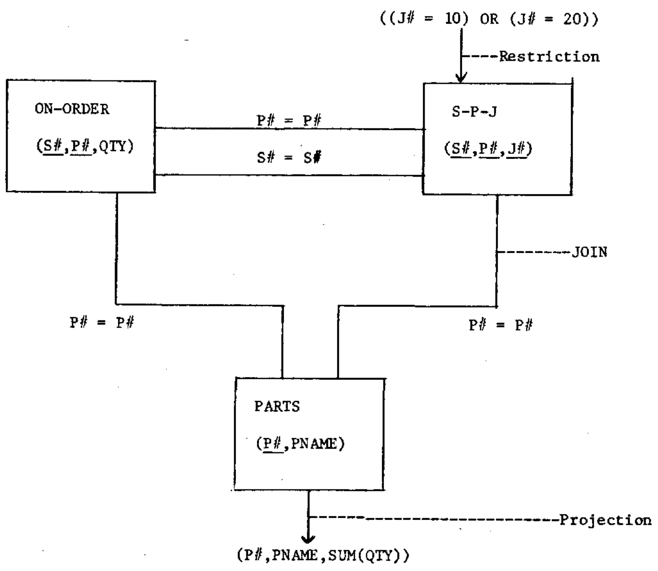


Fig. 1. Example query 1.

TABLE I

Relation R_i	Size s_i	$d_{i1} = P\#$		$d_{i2} = S\#$	
		b_{i1}	p_{i1}	b_{i2}	p_{i2}
R_1: ON-ORDER	1000	400	0.4	100	0.2
R_2: S-P-J	2000	400	0.4	450	0.9
R_3: PARTS	3000	900	0.9	–	–

b) To minimize *total time*, apply Algorithm SERIAL to each simple query. This results in one schedule per simple query. From these schedules, the candidate schedules for each joining attribute are extracted. Consider joining attribute d_{ij}. Its candidate schedule is identical to the schedule produced by Algorithm SERIAL, applied to the simple query in which d_{ij} occurs, up to the transmission of d_{ij}. All transmissions after that are deleted from the schedule.

3) *Integrate the candidate schedules.* For each relation R_i, the candidate schedules are integrated to form a processing schedule for R_i. The integration is done by procedure RESPONSE for response time minimization and by procedure TOTAL or procedure COLLECTIVE for total time minimization.

4) *Remove schedule redundancies.* Eliminate relation schedules for relations which have been transmitted in the schedule of another relation.

Algorithm GENERAL derives a query processing strategy for either response time or total time minimization by using the procedures RESPONSE, TOTAL, and COLLECTIVE.

A. Response Time Version

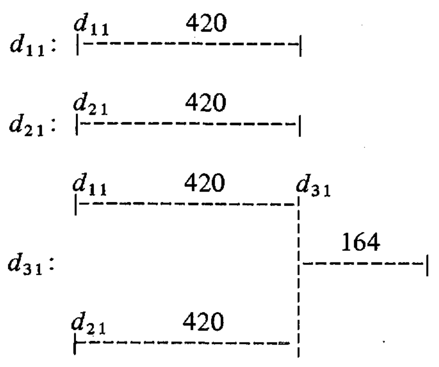
To minimize the response time of a relation R_k, we have to test whether transmitting an attribute d_{ij} to R_k is cost beneficial. Therefore, we have to know how long it takes to get d_{ij} to the site where R_k is located. Algorithm PARALLEL derives minimum response time schedules for joining attributes. In fact, there is no procedural difference between computing the minimum response time schedules for joining attributes whether they are sent to a result node or the node where R_k is located. Therefore, in step 2 of Algorithm GENERAL, Algorithm PARALLEL is applied to each joining attribute individually to compute a minimum response time schedule for each. All these candidate schedules are saved for integration by procedure RESPONSE.

Procedure RESPONSE

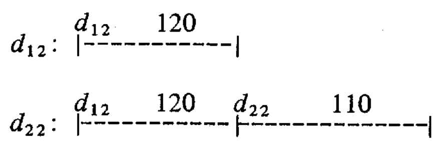
1) *Candidate schedule ordering.* For each relation R_i, order the candidate schedules on joining attribute $d_{ij}, j = 1, 2, \cdots, \sigma$ in ascending order of arrival time. Let ART_l denote the arrival time of candidate schedule CSCH_l. (For the d_{ij} joining attributes not in R_i, disregard the corresponding candidate schedules.)

2) *Schedule integration.* For each candidate schedule CSCH_l in ascending order, construct an integrated schedule for R_i that consists of the parallel transmission of CSCH_l and all CSCH_k with $k < l$. Select the integrated schedule with minimum response time.

Applying Algorithm GENERAL (response time) to the previous example, after the initial processing, two simple queries are formed, one having $d_{i1} = P\#$ as a common joining attribute and the other having $d_{i2} = S\#$. Running Algorithm PARALLEL on the $P\#$ query, the resulting candidate schedules are



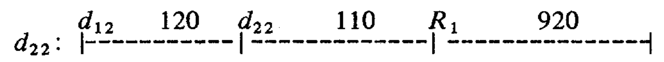
Applying Algorithm PARALLEL to the $S\#$ query, the candidate schedules are



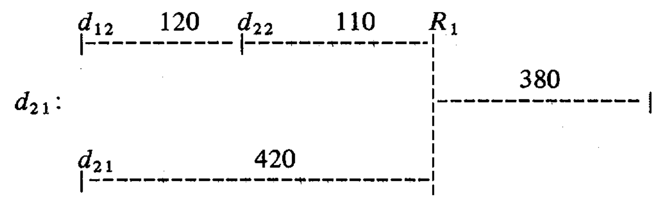
The construction of the schedule for R_1 will be given in detail. In step 1 of Procedure RESPONSE, the schedules of attributes that can be applied to relation R_1 are ordered on their arrival time in the node where R_1 is located. This gives the following result:

Attribute d_k	Arrival Time ART_k
d_{22}	330
d_{21}	420
d_{31}	584

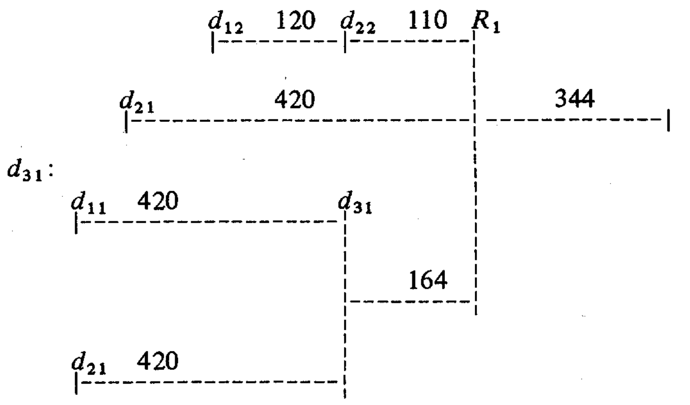
In step 2, for each of these attributes d_k, an integrated schedule for R_1 is constructed, consisting of the parallel transmission of all attributes having an arrival time less than or equal to ART_k. The following three integrated schedules are constructed:



Response time = $C(100) + C(0.2 * 450) + C(0.9 * 1000)$
= 120 + 110 + 920
= 1150.



Response time = $C(400) + C(0.9 * 0.4 * 1000)$
= 420 + 380
= 800.



Response time = $C(400) + C(0.4 * 0.4 * 900)$
+ $C(0.9 * 0.4 * 0.9 * 1000)$
= 420 + 164 + 344
= 928.

From these three schedules for R_1 and the Initial Feasible Solution for R_1, the schedule with the minimum response time is chosen. This is the second schedule and its response time is 800.

The schedules for R_2 and R_3 are constructed in a similar way. The Algorithm GENERAL (response time) query processing strategy for the example query is

A comparison to the response time of the IFS shows a considerable cost reduction.

Procedure RESPONSE is given the minimum response time schedules of all joining attributes. In step 2 of Algorithm GENERAL, Algorithm PARALLEL is applied to σ different simple queries. Therefore the minimum response time schedules for one common joining attribute are given in order of response time. This means that putting the candidate schedules in order of arrival time will take at most $O(\sigma m \log_2 \sigma)$, which is the merging complexity.

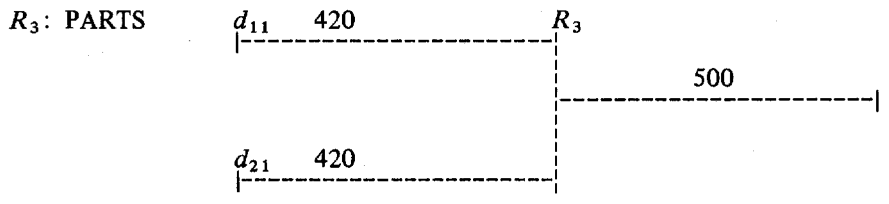
In step 2 of Algorithm GENERAL, Algorithm PARALLEL is applied σ times and its complexity is $O(m^2)$. However, the cost of applying procedure RESPONSE for every relation R_i is $O(\sigma m^2 \log_2 \sigma)$. Therefore, the complexity of Algorithm GENERAL (response time) is $O(\sigma m^2 \log_2 \sigma)$.

Now we will investigate the quality of the derived schedules. The schedules produced by algorithm PARALLEL for the joining attributes have a minimum response time. From this fact, we now can prove that each relation schedule has a minimum response time and, consequently, that the total query processing strategy has a minimum response time. Implicit within our proofs, we require the model assumption of attribute independence within each relation. Initial studies indicate that the effect of this assumption on the performance of the resulting query processing strategies may not be significant [20].

Theorem 1: Procedure RESPONSE derives a minimum response time integrated schedule for R_i.

Proof: The candidate schedules used in procedure RESPONSE are all minimum response time schedules because of the optimality of Algorithm PARALLEL [13]. Procedure RESPONSE puts these candidate schedules in ascending order of arrival time, and it only considers integrated schedules for relation R_i that consist of the parallel transmission of joining attributes with arrival time less than or equal to the arrival time of a certain CSCH_k. We will show that no other integrated schedule needs to be considered.

Assume that we are given a minimum response time schedule for R_i. This schedule contains the transmissions of some joining attribute which arrives last (i.e., has the greatest value of ART_k in the schedule). The corresponding integrated schedule (based on ART_k as the largest ART value) considered



Response time = 920. Total time = 2910.

by procedure RESPONSE contains at least as many transmissions of joining attributes as the previous schedule and, therefore, its selectivity is at least as small. Hence, its response time must be less than or equal to that of the minimum response time schedule.

Theorem 2: Algorithm GENERAL (response time) derives a minimum response time processing strategy for any distributed query.

Proof: The response time of a processing strategy is the maximum response time of the relation schedules in the strategy. Theorem 1 showed that these relation schedules have minimum response time. Hence, the theorem follows.

B. Total Time Version

The motivation for using the minimization of the total time as the objective of Algorithm General comes from the use of the algorithm in a multiprocessing environment. Minimizing response time leads to an increased number of parallel data transmissions in the query processing strategy. In multiprocessing systems, under moderate to heavy loads, these extra transmissions may lead to significant queueing delays and synchronization delays, delays which may cause poor query response time. By minimizing the total time in a query processing strategy, fewer transmissions will be included and improved actual response times may result in certain system environments.

The candidate schedules produced in step 2 of Algorithm GENERAL (total time) look very much like the schedules produced by Algorithm SERIAL. Algorithm SERIAL produces minimum total time schedules for simple queries. In its optimality proof [14], it is shown that parallel transmissions of common joining attributes can be avoided. Therefore, we will do the same in Algorithm GENERAL (total time). Only parallel transmissions of *different* joining attributes are allowed. This means that in constructing a schedule for relation R_i, we will consider only one candidate schedule per joining attribute of R_i. This candidate schedule will be the one that minimizes the total time of transmitting R_i if only that one joining attribute is considered.

In [1], it was shown that it is not sufficient to just look at the candidate schedules produced in step 2 of Algorithm GENERAL. For every candidate schedule *to* relation R_i containing a transmission of a joining attribute *from* the same relation R_i, we have to add another candidate schedule, namely, one without the transmission of this joining attribute. This is necessary since the size of relation R_i cannot be reduced by the selectivity of its own attribute. Thus, this data transmission in the incoming schedule may not be cost beneficial. Among this extended set of candidate schedules, we select the schedule which minimizes the total time of transmitting R_i if only one joining attribute is considered. This selected schedule for relation R_i considering joining attribute d_{ij} will be called BEST_{ij}.

We define SLT_{ij} to be the accumulated attribute selectivity of the BEST_{ij} candidate schedule into R_i. Note that j may take on values from 1 to σ only if the common joining attributes d_{ij} appear in R_i.

Procedure TOTAL

1) *Adding candidate schedules.* For each relation R_i and each candidate schedule CSCH_l, do the following.

If this schedule contains a transmission of a joining attribute of R_i, say d_{ij}, then add another candidate schedule which is the same as CSCH_l except that the transmission of d_{ij} is deleted.

2) *Select the best candidate schedule.* For each relation R_i and for each joining attribute $d_{ij} (j = 1, 2, \cdots, \sigma)$, select the candidate schedule which minimizes total time for transmitting R_i if only the joining attributes are considered which can be joined with d_{ij}.

3) *Candidate schedule ordering.* For each relation R_i, order the candidate schedules BEST_{ij} on joining attributes d_{ij}, $j = 1, 2, \cdots, \sigma$, so that $\text{ART}_{i1} + C(s_1 * \text{SLT}_{i1}) \leqslant \cdots \leqslant \text{ART}_{i\sigma} + C(s_i * \text{SLT}_{i\sigma})$. (For the joining attributes not in R_i, disregard BEST_{ij}.) ART_{ij} denotes the arrival time of the BEST_{ij} schedule.

4) *Schedule integration.* For each BEST_{ij} in ascending order of j, construct an integrated schedule to R_i that consists of the parallel transmission of candidate schedule BEST_{ij} and all schedules BEST_{ik} where $k < j$. Select the integrated schedule that results in the minimum total time value

$$\text{TOTT}_i = \sum_{k=1}^{j} \left[\text{ART}_{ik} + C\left(s_i * \prod_{k=1}^{j} \text{SLT}_{ik}\right) \right].$$

Applying Algorithm GENERAL (total time) to the example, two simple queries are formed on the following joining attributes, $d_{i1} = P\#$ and $d_{i2} = S\#$ in the example query. In step 2 of Algorithm GENERAL (total time), the following serial candidate schedules are formed.

For $P\#$,

```
      d11    420
d11:  |-----------|

      d11    420    d21    180
d21:  |-----------|-----------|

      d11    420    d21    180    d31    164
d31:  |-----------|-----------|-----------|
```

For $S\#$,

```
      d12    120
d12:  |-----------|

      d12    120    d22    110
d22:  |-----------|-----------|
```

Again, the construction of the schedule for R_1 will be discussed in detail. We will treat the two attributes of R_1 in turn.

Attribute d_{11}

In step 1 of Procedure TOTAL, the following two schedules are added to the above schedules for $P\#$. The first one is obtained by deleting the transmission of d_{11} from the schedule for d_{21}, and the second one by deleting d_{11} from the schedule for d_{31}:

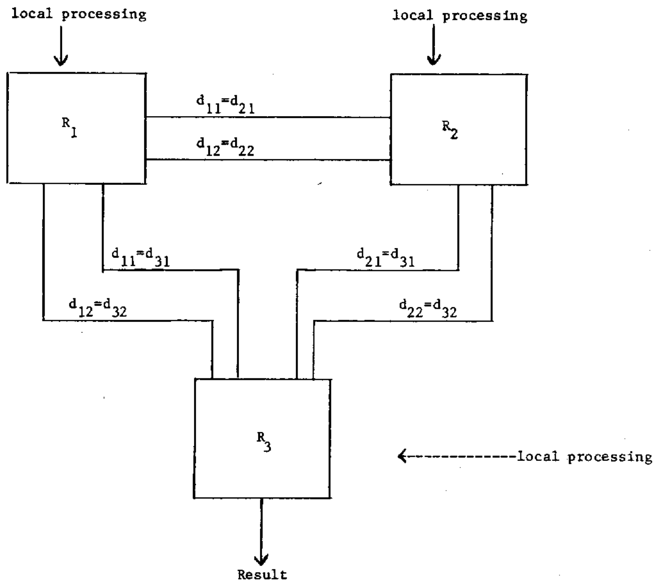


Fig. 2. Example query 2.

d_{21} 420
d_{21}^1: |--------------|

d_{21} 420 d_{31} 380
d_{31}^1: |--------------|--------------|

Thus, there are five schedules that only contain transmissions of the $P\#$ attribute. Four of them will be tested. The transmission of d_{11} to R_1 is meaningless as a schedule.

TABLE II

Relation	Size	d_{i1}		d_{i2}	
R_i	s_i	b_{i1}	p_{i1}	b_{i2}	p_{i2}
R_1	1000	300	0.6	300	0.25
R_2	2000	100	0.2	900	0.75
R_3	2000	400	0.8	600	0.5

d_{11} 420 d_{21} 180 R_1 420
d_{21}: |--------------|------------|--------------------|

Total time $= C(400) + C(0.4 * 400) + C(0.4 * 1000)$
$= 420 + 180 + 420$
$= 1020.$

d_{11} 420 d_{21} 180 d_{31} 164 R_1 380
d_{31}: |--------------|------------|------------|--------------|

Total time $= C(400) + C(0.4 * 400) + C(0.4 * 0.4 * 900) + C(0.4 * 0.9 * 1000)$
$= 420 + 180 + 164 + 380$
$= 1144.$

d_{21} 420 R_1 420
d_{21}^1: |--------------|--------------|

Total time $= C(400) + C(0.4 * 1000)$
$= 420 + 420$
$= 840.$

d_{21} 420 d_{31} 380 R_1 380
d_{31}^1: |--------------|----------------------------------|--------|

Total time $= C(400) + C(0.4 * 900) + C(0.4 * 0.9 * 1000)$
$= 420 + 380 + 380$
$= 1180.$

Because the schedule of d_{21}^1 has the smallest total time, it is chosen as the BEST_{11} schedule.

Attribute d_{12}

In step 2 of Procedure TOTAL, only one schedule is added to the above schedules for attribute $S\#$.

d_{22} 470
d_{22}^1: |------------------|

Each of the schedules for $S\#$ is applied to R_1. Again, transmitting d_{12} to R_1 is not considered.

```
      d12   120   d22   110   R1      920
d22:  |-----------|-----------|---------------|
```

Total time $= C(100) + C(0.2 * 450) + C(0.9 * 1000)$
$= 120 + 110 + 920$
$= 1150.$

```
      d22       470               R1    920
d22': |---------------------------|-----------|
```

Total time $= C(450) + C(0.9 * 1000)$
$= 470 + 920$
$= 1390.$

Because the schedule of d_{22} has the smallest total time, it is chosen as the BEST_{12} schedule.

In steps 3 and 4 of the Procedure TOTAL, the above obtained BEST_{1j} schedules are ordered on their total time, and the following two integrated schedules are constructed:

```
d21   420                 R1            420
|-------------------------|-------------------------|

d21   420                 R1
|-------------------------|
                          |
                          |            380
                          |--------------------|
                          |
    d12   120   d22 110   |
    |-----------|---------|
                          |
```

The first of these two has the smallest total time; it is chosen as the solution of Algorithm GENERAL for R_1.

The schedules for R_2 and R_3 are constructed in a similar way. The Algorithm GENERAL (total time) query processing strategy for the example query is

```
R1: ON-ORDER   d21      420      R1      420
               |-----------------|-----------------|

R2: S-P-J      d12    120   R2    420
               |------------|------------|

R3: PARTS      d11      420      d21  180  R3      500
               |-----------------|---------|----------------------|
```

Response time = 1100. Total time = 2480.

The total time of this strategy is considerably smaller than the total time of the IFS. Also, the response time of its strategy for the example query is not much larger than the response time of the strategy produced by the response time version of Algorithm GENERAL, although it only tries to minimize total time.

Algorithm GENERAL (total time) has a slightly better worst case complexity than the response time version, $O(\sigma m^2)$. Again, assume that a general query requires data from m relations, and all m relations are joined on σ joining attributes. In step 2, algorithm SERIAL is applied to each simple query. The complexity of this is $O(\sigma m \log_2 m)$ because the joining attributes have to be ordered by size.

The complexity of the procedure TOTAL is $O(\sigma m^2)$. In step 1, no more than $O(\sigma m)$ candidate schedules are added. This means that for every relation, the procedure has to determine the BEST_{ij} schedule among $O(\sigma m)$ candidate schedules. Hence, the cost of step 2 is $O(\sigma m^2)$. This means that for an arbitrary general distributed query, Algorithm GENERAL (total time) has a processing complexity no worse than $O(\sigma m^2)$.

The quality of the resulting query processing strategies is much harder to analyze than those for minimizing response time. The BEST_{ij} schedules were shown to be the best possible schedules for minimizing the total transmission time of relation R_i if only one common joining attribute is considered [1]. However, the optimality is lost during the integration of the different BEST_{ij} schedules. In [13], it was shown that finding the minimum total time schedule is equivalent to a problem which is proved to be NP-hard.

C. Handling Redundant Data Transmissions

A major reason why the query processing strategy derived by Algorithm GENERAL (total time) is not optimal is that procedure TOTAL does not consider the existence of redundant data transmissions in separate relation schedules. (Note that such redundant transmissions between relation schedules do not effect the minimum response time strategies of Section III-A. This is because Algorithm GENERAL (response time) minimizes the response time of each relation schedule separately.)

This example illustrates the benefit of redundant data transmissions. A database contains three relations: R_1, R_2, and R_3. The query represented by Fig. 2 is entered in a distributed database system wherein each relation is located at a separate node and the result of the query is required at a different node.

The sizes and selectivities for each d_{ij}, after local processing, are given in Table II. Assume the cost function for the system to be $C(X) = 25 + X$.

By applying Algorithm GENERAL (total time) to this example query (the derivation appears in the Appendix), the resulting processing strategy is

```
d21    125    R1     225
|-------------|-------------|

d12    325            d32    175    R2          275
|---------------------|-------------|---------------------|

d21    125    d11     85     R3  265
|-------------|--------------|-------------------|
```

Total time = 1600.

The first redundant transmission one can clearly recognize is the transmission of the attribute d_{21} towards R_1. This transmission will physically take place only once, so its cost (125) should be accounted for only once as well. We might visualize this removal as follows:

```
                  R1         225
                  |-----------------|
                  |
d21     125       |
|-----------------|
                  |
                  |d11     85     R3     265
                  |-------------|-------------|
                  |
d12         325          d32          175          R2          275
|------------------------|--------------------------|--------------------------|
```

Total time = 1475.

Apart from this type of direct transmission redundancy, it is possible to discern a somewhat more complicated type. It might happen that the transmission of d_{ij} towards the node of R_k is part of some schedule, while it does not occur in the schedule of R_k itself. Because the values of this attribute d_{ij} are available for a semi-join with R_k, it seems fair enough to assume that this semi-join will actually take place, and that the cost for the final transmission of R_k will subsequentially decrease. In such a case, it is clear that R_k cannot be transmitted before all attributes have arrived, which means that the schedule must be synchronized. This could lead to an increased response time. In our example, we can also detect such a case.

In the R_2 schedule, d_{12} is sent to d_{32}. So d_{12} is available for R_3, although it is not part of the R_3 schedule. The final transmission of R_3 will only cost $C(0.6 * 0.2 * 0.25 * 2000) = 85$ instead of 265. Again we try to illustrate this:

```
                    R1     225
                    |----------------------------|
                    |
d21     125         |
|-------------------|
                    |
                    |d11     85     R3     85
                    |--------------|--------------|
                    |              |
d12            325                 |d32    175    R2    275
|----------------------------------|---------------|------------|
                                   |
```

Total time = 1295.

(Note that R_3 can be sent at time $t = 325$, instead of at time $t = 210$.)

As these aspects are not considered during the selection of the final schedules, a few beneficial strategies will not be found by Algorithm GENERAL (total time). This is mainly due to the characteristic that a best schedule is separately chosen for each relation, without regarding the "collective" benefits. It means that for some R_k, a schedule might be just too expensive and thus rejected, while it had many transmissions in common with selected schedules of other relations, and thus would have been a very good choice for the strategy as a whole.

In order to test the above statements, we developed an alternative version of Algorithm GENERAL (total time) called Algorithm GENERAL (collective), completely based on redundant transmissions. It is simpler in that it constructs only one basic strategy for the entire query, after which a few variations are tried out. The schedules of the basic strategy include as many semi-joins as possible for each relation. This tends towards a large number of redundant transmissions. The investments for such an extended integrated schedule are shared, as it were, by all R_k. The strategy is then perturbed by subsequentially trying to drop schedule components (i.e., linear parts of an integrated schedule) for some relation R_k, and finally to drop such a component, which actually stands for the semi-joins on one particular attribute, from the entire strategy.

Algorithm GENERAL (collective) performs step 2b) in Algorithm GENERAL. The candidate schedule integration is performed in step 3 by the following procedure.

Procedure COLLECTIVE

1) *Select candidate schedule.* For each relation R_i and for each joining attribute $d_{ij} (j = 1, 2, \cdots, \sigma)$, select the minimum cost candidate schedule that contains the transmission of all components of attribute j with selectivities <1.

2) *Build processing strategy.* For each relation R_i, define the schedule to be the parallel transmission of all d_{ij} candidate schedules to R_i.

3) *Test variations of strategy.* Using a removal heuristic, derive new strategies by removing the most costly data transmissions. Compare the total time cost of the new and old strategies. Maintain the less costly strategy. Continue testing until no cost benefit can be obtained.

Consider the state of the relations R_1, R_2, and R_3 in Table III after local processing for the query in Fig. 3. The transmission cost function is $C(X) = 10 + X$.

The schedule components for the three attributes are

```
d31     160    d21     70
|-------------|-----------|

d32     190    d22     154
|-------------|--------------|

d13   40  d23  19
|--------|------|
```

leading to the basic strategy (integrating as much as possible)

```
d13   40  d23   19  R1        310
|--------|---------|-------------------|

d31     160          R2
|-------------------|
                    |
d32     190         |         55
|-------------------|-------------------|
                    |
d13     **          |
|-------------------|

         d31      **      d21      70     R3
         |---------------|---------------|
                                         |      410
                                         |---------------------|
                                         |
d32     **     d22      154              |
|-------------|--------------------------|
```

Total Time = 1408.

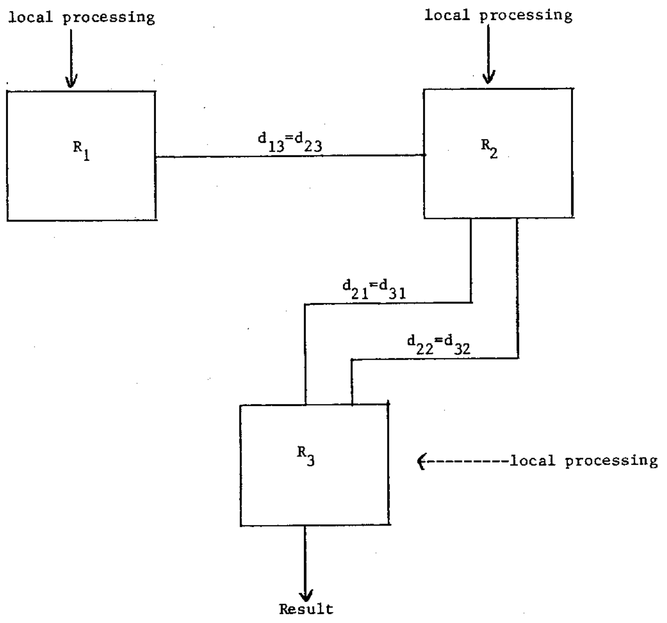


Fig. 3. Example query 3.

TABLE III

Relation	Size	d_{i1}		d_{i2}		d_{i3}	
R_i	s_i	b_{i1}	s_{i1}	b_{i2}	s_{i2}	b_{i3}	s_{i3}
R_1	1000	–	–	–	–	30	0.1
R_2	2500	200	0.4	240	0.8	90	0.3
R_3	1250	150	0.3	180	0.6	–	–

(Here ** stands for transmissions, which are already accounted for.)

During the variation trials, it turns out that it is beneficial to drop the component of attribute 2 from the schedule for R_3. This saves 154 time units, while the cost for the transmission of R_3 will increase: $C(0.4 * 1250) = 510$. Together this means a gain of 54. In the second variation round, it is found that it is better to leave out the component of attribute 2 at all relations.

Thus, the final solution of the "COLLECTIVE" algorithm for this example is

```
d13    40    d23   19   R1    310
|-----------|----------|-------------|

d31      160        R2
|-----------------|
                  |        85
                  |-----------------|
d13    **         |
|-----------------|

d31    **           d21    70   R3     510
|-----------------|----------|-----------------|
```

Total time = 1194.

Algorithm GENERAL (collective) has the same worst case complexity as Algorithm GENERAL (total time) since the complexity of procedure COLLECTIVE is $O(\sigma m^2)$. In step 1, for every relation (m possibilities) and for every joining attribute (σ possibilities), the desired candidate schedule must be selected from m possible schedules. The optimization of step 3 is a greedy heuristic procedure with linear order.

For a significant number of queries, the collective version of Algorithm GENERAL will produce a smaller total time strategy than will the total time version. Such improvement is due to the recognition and inclusion of redundant data transmissions among separate relation schedules in the overall query processing strategy.

IV. Conclusions

We claim Algorithm GENERAL to be an efficient algorithm of polynomial complexity that derives close to optimal query processing strategies on distributed systems. The algorithm was designed as a straightforward extension of the processing tactics found optimal for simple queries in Algorithm PARALLEL and Algorithm SERIAL.

There are two primary versions of Algorithm GENERAL. To minimize response time of a processing strategy, parallel data transmissions are emphasized by the use of Algorithm PARALLEL and Procedure RESPONSE. Algorithm GENERAL (response time) can be proved to derive minimum response time strategies under the assumption of attribute independence within query relations. To minimize the total time of a processing strategy, serial time transmissions are emphasized by the use of Algorithm SERIAL and Procedure TOTAL in Algorithm GENERAL (total time).

Recognizing the existence of identical data transmissions in different relation schedules may lead to a further reduction in the total time of a query processing strategy. We develop a third version of Algorithm GENERAL (collective) that uses Algorithm SERIAL and Procedure COLLECTIVE to produce strategies with increased data transmission redundancy among schedules. In many cases, the total time of these strategies is less than the total time of strategies produced by Algorithm GENERAL (total time).

Algorithm GENERAL can be applied to any general distributed query environment. It is relatively simple to program and has the added flexibility that all versions can be implemented together. Then, depending upon run-time factors, such as system load or query complexity, the optimization objective can be changed by a simple switch in the program.

Appendix

The derivation of the query processing strategy using Algorithm GENERAL (total time) on the database state found in Table II proceeds as follows.

Two simple queries are formed on the joining attributes d_{i1} and d_{i2}. In step 2 of Algorithm GENERAL (total time), the following serial candidate schedules are formed.

For attribute d_{i1}:

```
d11:  d21        125       d11   85
      |-----------------|-----------|

d21:  d21        125
      |-----------------|

d31:  d21        125       d11   85     d31   73
      |-----------------|-----------|---------|
```

For attribute d_{i2}:

```
d_12:  d_12      135
       |--------------|

d_22:  d_12      325       d_32   175          d_22  137
       |--------------|--------------|----------|

d_32:  d_12      325       d_32   175
       |--------------|--------------|
```

Entering Procedure TOTAL, the BEST_{i1} and BEST_{i2} schedules are determined as follows:

the linear schedules (i.e., the first in the BEST_{ij} ordering) are the cheapest solutions. Thus, our strategy will be

```
d_21   125   R_1      225
|-----------|-------------|

d_12   325          d_32   175    R_2    275
|----------------|-----------|-------------|

d_21   125   d_11    85    R_3     265
|-----------|------------|-------------|
```

Total Time = 1600

```
                                                                 Total Time:
d_21    125   R_1    225
|-----------|----------------|                                   →  350

d_21    125   d_11    85    d_31    73    R_1    185
|-----------|-----------|-----------|-----------|------------|   →  568

d_21    125   d_31   105   R_1      185
|-----------|---------|----------------|                         →  415
```

The first schedule becomes our BEST_{11}.
For BEST_{12}, we have to choose among

```
d_12   325           d_32   175    R_1    525
|----------------|-----------|--------------|                    → 1025

d_12   325           d_32   175    d_22   137    R_1    400
|----------------|-----------|-----------|-----------|-----------|  → 1037

d_32   625                R_1   525
|-------------------|----------------|                           → 1150

d_32   625                 d_22   425    R_1    400
|--------------------|-----------|---------|                     → 1450
```

The first schedule thus will be our BEST_{12}.
In the same way, we find

```
BEST_21:  d_21   125   d_11   85   d_31    73       R_2    985
          |-----------|---------|-------------|------------------|   → 1268

BEST_22:  d_12   325        d_32  175  R_2    275
          |--------------|---------|-------------|                   →  775

BEST_31:  d_21   125   d_11   85  R_3    265
          |-----------|---------|-------------|                      →  475

BEST_32:  d_12   325        R_3    525
          |--------------|----------------|                          →  850
```

By now we can determine the solution for each R_i.
For example, for R_1, there are two possibilities, which are

```
I)      d_21   125   R_1    225
        |-----------|-------------|                               →  350

and

II)                        d_21    125
                           |-------------|
                                         |R_1    125
                                         |-------------|          →  750
        d_12   325              d_32    175  |
        |------------------|-------------|
```

We conclude that the first schedule is superior.
For the other relations, it also (coincidentally) turns out that

References

[1] P. M. G. Apers, "Critique on and improvement of Hevner and Yao's distributed query processing algorithm G," Vrije Univ., Amsterdam, The Netherlands, IR 48, Feb. 1979.

[2] —, "Distributed query processing: Minimum response time schedules for relations," Vrije Univ., Amsterdam, The Netherlands, IR 50, Mar. 1979.

[3] C. Baldissera, G. Bracchi, and S. Ceri, "A query processing strategy for distributed data bases," in *Proc. EURP-IFIP 79*, 1979, pp. 667–678.

[4] P. A. Bernstein and D. W. Chiu, "Using semi-joins to solve relational queries," *J. Ass. Comput. Mach.*, vol. 28, Jan. 1981.

[5] P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. Rothnie, "Query processing in a system for distributed databases (SDD-1)," *ACM Trans. Database Syst.*, vol. 6, Dec. 1981.

[6] D. Chiu and Y. Ho, "A methodology for interpreting tree queries into optimal semi-join expressions," in *Proc. ACM SIGMOD Conf.*, Santa Monica, CA, 1980.

[7] W. W. Chu and P. Hurley, "A model for optimal processing for distributed databases," in *Proc. 18th IEEE COMPCON*, Spring 1979, pp. 116–122.

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

[9] R. Epstein, M. R. Stonebraker, and E. Wong, "Distributed query processing in a relational data base system," in *Proc. ACM-SIGMOD*, May 1978, pp. 169–180.

[10] R. Epstein and M. Stonebraker, "Analysis of distributed data base processing strategies," in *Proc. Int. Conf. Very Large Data Bases*, Montreal, P.Q., Canada, 1980.

[11] N. Goodman and O. Shmueli, "Hierarchies of database state reductions," Aiken Computation Lab., Harvard Univ., Cambridge, MA, Tech. Rep. TR-18-80, 1980; also, *ACM Trans. Database Syst.*, to be published.

[12] A. R. Hevner and S. B. Yao, "Query processing in distributed database systems," in *Proc. 3rd Berkeley Workshop on Distributed Data Management and Comput. Networks*, Berkeley, CA, 1978.

[13] A. R. Hevner, "The optimization of query processing on distributed database systems," Ph.D. dissertation, Database Syst. Res. Cen., Dep. Comput. Sci., Purdue Univ., W. Lafayette, IN, Rep. DB-80-02, Dec. 1979.

[14] A. R. Hevner and S. B. Yao, "Processing in distributed database systems," *IEEE Trans. Software Eng.*, vol. SE-5, pp. 177–187, May 1979.

[15] A. R. Hevner, "Data allocation and retrieval in distributed systems," in *Advances in Database Management, Vol. II*, P. Fisher, Ed. New York: Wiley, 1983.

[16] L. Kershberg, P. Ting, and S. B. Yao, "Query optimization in a Star network," Bell Lab. Tech. Rep., 1979; also, *ACM Trans. Database Syst.*, to be published.

[17] N. G. Toan, "Decentralized dynamic query decomposition for distributed database systems," in *Proc. ACM Pacific Conf.*, San Francisco, CA, 1980.

[18] G. Pelagatti and F. A. Schreiber, "Evaluation of transmission requirements in distributed database access," in *Proc. ACM SIGMOD Conf.*, Boston, MA, 1979.

[19] G. M. Sacco and S. B. Yao, "Query optimization in distributed database systems," in *Advances in Computers, Vol. 21.* New York: Academic, 1982.

[20] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access path selection in a relational database management system," in *Proc. ACM SIGMOD Conf.*, Boston, MA, 1979.

[21] M. Stonebraker and E. Neuhold, "A distributed version of INGRES," in *Proc. 2nd Berkeley Workshop Distributed Data Management and Comput. Networks*, May 1977, pp. 19–36.

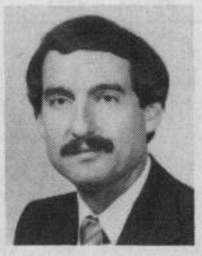
[22] M. Takizawa, "Operational query decomposition algorithm in distributed databases," Japan Inform. Processing Develop. Cen. Rep. TR80/2, 1980.

[23] E. Wong and K. Youssefi, "Decomposition–A strategy for query processing," *ACM Trans. Database Syst.*, vol. 1, Sept. 1976.

[24] E. Wong, "Retrieving dispersed data from SDD-1: A system for distributed databases," in *Proc. 2nd Berkeley Workshop Distributed Data Management and Comput. Networks*, May 1977, pp. 217–235.

[25] Z. Ozsoyoglu, "Query processing in distributed relational databases," Ph.D. dissertation, Univ. Alberta, 1980.

Peter M. G. Apers, photograph and biography not available at the time of publication.



Alan R. Hevner received the Ph.D. degree in computer science in 1979 from Purdue University, West Lafayette, IN.

He is currently an Assistant Professor of Information Systems at the University of Maryland, College Park. His research interests include database systems design, distributed database systems, database performance evaluation, and office information systems.

Dr. Hevner is a member of the Association for Computing Machinery and the IEEE Computer Society. He is an Associative Editor of *Database Engineering*, a publication of the IEEE Technical Committee on Database Engineering. He edited the December 1982 special issue of that publication on current research on distributed database systems.

S. Bing Yao received the Ph.D. degree from the University of Michigan, Ann Arbor, in 1974.

He is currently an Associate Professor of Business Management and Computer Science and is Director of the Database Systems Research Group at the University of Maryland, College Park. He is involved in the design and implementation of several database management systems and tools including XQL, a relational database system. He has authored over 50 technical papers on database storage structures, database performance optimization, database design, distributed database systems, database machines, and office automation. He is Editor of *Data Base Design Techniques* (Springer-Verlag, 1982) and *Principles of Data Base Design* (Prentice-Hall, 1982). He is founder of Software Systems Technology, a computer form specializing in database systems and machines. His international consulting clients include Bell Laboratories, IBM, and the Swedish Technical Development Board. He is an Editor of the *ACM Transactions on Office Information Systems*, is on the Editorial Board of the *ACM Transactions on Data Systems*, was Program Chairman of the 4th International Conference on Very Large Data Bases and the NYU Symposium on Database Design, and is the Conference Chairman of the 9th International Conference on Very Large Data Bases.