A Parallel Execution Method for Minimizing Distributed Query Response Time

Chihping Wang, *Member, IEEE*, Arbee L. P. Chen, *Member, IEEE*, and Shiow-Chen Shyu

***Abstract*—Performance studies [6], [17] show that traditional semi-join processing methods are sometimes inefficient because of the storage and processing overhead. To remedy this problem, we propose a new semi-join processing method, called one-shot semi-join execution. This method allows parallel generation of all the semi-join projections, parallel transmission of all the semi-join projections, and parallel execution of all the semi-joins. We apply this method to optimize the response time for processing distributed queries. A response time model is established, which considers both data transmission time and local processing time. Based on this model, we develop and analyze an efficient query processing algorithm.**

***Index Terms*—Distributed query processing, distributed databases, one-shot semi-joins, query optimization, semi-join processing.**

I. Introduction

QUERY processing in distributed relational databases [7] often requires shipping relations between different sites. To reduce the data transmission cost, *semi-joins* were introduced [2], [3]. A semi-join from relation R_i to relation R_j, denoted by $R_j \ltimes R_i$, is defined as $\text{PROJECT}_{R_j}(R_i \bowtie R_j)$, where $R_i \bowtie R_j$ is the join of R_i and R_j, and $\text{PROJECT}_A(B)$ is the projection of relation B on the attributes of relation A. In a distributed database system, it is implemented as follows: Project R_i on the join attributes (of the join between R_i and R_j), then ship this projection (called a *semi-join projection*) to the site of R_j and perform the join with R_j.

It has been proposed that a distributed query be processed as follows [1], [4], [5]:

1) *Initial local processing*: all local operations including selections and projections are processed.
2) *Semi-join processing*: the only operations left after initial local processing are joins between relations at different sites. A *semi-join program* is derived from the remaining join operations and executed to reduce the size of the relations.
3) *Final Processing*: all relations which are needed to calculate the answer of the query are transmitted to a final site where joins are performed and the answer to the query obtained.

Numerous algorithms [8], [9], [15], [21], [23] have been developed to determine a semi-join program for optimal distributed query processing. The heuristic for SDD-1 developed by Bernstein *et al.* [4] is a typical one and can be described as follows. It repeatedly evaluates the benefit and cost of the candidate semi-joins, selects the most profitable one, updates the cardinality of the relation to be reduced by this semi-join, until there are no profitable semi-joins left. Researchers have studied this problem for processing several special classes of queries. Hevner and Yao [14] developed an optimal algorithm for simple queries. Their approach has been generalized [1] and a heuristic based on exhaustive search has been found. Chiu, Bernstein, and Ho [11] developed a dynamic programming algorithm for chain queries. Chiu and Ho [12] further generalized this algorithm to processing tree queries. Yu, Ozsoyoglu, and Lam [22] considered the same class of queries. But they further reduced the search space by identifying some properties of the optimal strategies. Pramanik and Vineyard [16] developed an algorithm for a more general class of tree queries where two relations may have more than one join attribute. Chen and Li [10] studied star queries and found an optimal algorithm based on some assumptions. Some of their assumptions were later relaxed by Wang and Li [18].

Most semi-join algorithms favor executing semi-joins sequentially such that the reduction effect of a semi-join can be propagated to reduce the cost of later semi-joins. For example, the cost of $R_j \ltimes R_i$ may be lowered if another semi-join $R_i \ltimes R_k$ is executed first. However, performance studies [6], [17] show that such semi-join processing strategies are sometimes inefficient for the following reasons:

1) *Loss of parallelism:* The sequential execution of semi-joins excludes the possibility of parallel semi-join execution in a distributed system.
2) *Processing overhead*: Before $R_i \ltimes R_j$ is executed, R_j has to be scanned in order to generate the semi-join projection. If $R_k \ltimes R_j$ also appears in the sequential semi-join program, R_j has to be scanned again, which increases the processing overhead.
3) *Loss of global semi-join optimization:* The sequential execution of semi-joins excludes the possibility of performing multiple semi-joins to the same relation simultaneously, for which global optimization techniques [20] may be applied.
4) *Inaccurate semi-join reduction estimation:* In order to find a good processing strategy, it is needed to accurately

Manuscript received November 23, 1990; revised June 10, 1991. This work was supported in part by the Republic of China National Science Council under Contract NSC 80-0408-E-007-01.

C. Wang is with the Department of Computer Science, University of California, Riverside, CA 92521.

A. L. P. Chen is with the Department of Computer Science, National Tsing Hua University, Taiwan 30043.

S.-C. Shyu is with IBM Santa Teresa Laboratory, San Jose, CA 95150.

IEEE Log Number 9107408.

estimate the cost and reduction benefit of semi-joins. If such estimation is done each time after a semi-join is executed, too much processing cost may be incurred. If all such estimates are done before the semi-join processing, the accuracy may be low because estimation errors may propagate and be magnified through the sequential execution of semi-joins. This inaccuracy affects the semi-join algorithm's ability to determine an optimal strategy.

To alleviate the above problems, we have proposed a new semi-join processing procedure, named *one-shot semi-join execution* [19]. This method executes all applicable semi-joins to the relations at a time. That is, each relation will be reduced by a set of semi-joins at a time, and the semi-join processing at all sites can be performed simultaneously. As a result, each relation needs to be scanned only once to process all applicable semi-joins. These semi-joins can be processed employing a global optimization algorithm. Moreover, since all applicable semi-joins are executed at one shot, no inaccurate estimation of the semi-join cost and benefit will be propagated. The query optimizer therefore decides a semi-join program which is a set instead of a sequence of semi-joins.

In this paper, we consider using the one-shot semi-join execution to optimize the response time of distributed query processing. Both data transmission time and local processing time (including disk I/O) are considered in the approach. This distinguishes our work from most existing query optimization methods. The one-shot approach makes use of various kinds of parallelism for minimizing the query response time. The kinds of parallelism include parallel generation of all the semi-join projections, parallel transmission of all the semi-join projections, and parallel execution of all the semi-joins. The response time, including both data transmission time and local processing time, is modeled as the time for generating and transmitting semi-join projections, executing the semi-joins, transmitting the reduced relations, and processing the final joins.

Distributed INGRES [13] and AHY [1] query optimization algorithms also considered minimizing the query response time. In distributed INGRES the optimization of response time was achieved by "equalizing" the data sizes at each site such that each site requires about the same amount of processing time. There is no cost model defined for local processing, and the assumption that equal data size requires equal processing time is rather unrealistic. Moreover, the distributed INGRES algorithm is a heuristic while our approach is optimal based on our cost model.

AHY is also a heuristic whose cost model includes only data transmission. Although it allows parallel transmission of semi-join projections, parallel generation of semi-join projections and parallel execution of semi-joins are not considered. As a result, its cost function favors sequential semi-join executions and suffers the inefficiencies as we pointed out earlier. On the other hand, our approach explores more parallelism and considers both transmission and processing time in the cost model.

The rest of the paper is organized as follows. In Section II, we describe the one-shot semi-join execution. In Section III, the response time is modeled and its minimization is formulated. A polynomial-time algorithm is developed in Section IV, together with its correctness proof, its complexity analysis, and an example to illustrate the algorithm. We conclude in Section V.

II. One-Shot Semi-Join Execution

As described in the previous section, the goal of the one-shot semi-join execution is to remedy the inefficiency of traditional semi-join processing strategies, which favor sequential execution of semi-joins. Under this new method, the initial local processing and final join processing steps remain the same. However, the query optimizer has to decide a set of semi-joins to execute in the semi-join processing step. These semi-joins are executed in three phases, namely, the projection phase, the transmission phase, and the reduction phase. They are explained in the following:

The Projection Phase: During the projection phase, each relation R_i is scanned once to generate all the necessary semi-join projections. That is, if $R_{j_1} \ltimes R_i, R_{j_2} \ltimes R_i, \cdots, R_{j_k} \ltimes R_i$, are to be executed, R_i is scanned once to generate $\prod_{r_{j_1}} R_i$, $\prod_{r_{j_2}} R_i, \cdots, \prod_{r_{j_k}} R_i$, where r_{j_h} is the join attribute between R_{j_h} and R_i. All semi-join projections are hashed at the time they are generated. Hashing is used because it speeds up the processing during the reduction phase.

To reduce memory overhead, semi-join projections can be pipelined to the transmission phase.

The Transmission Phase: All the semi-join projections are then transmitted in parallel to the corresponding sites.

The Reduction Phase: After the transmission phase, all semi-join projections for reducing a relation R_i are available at the site where R_i resides. Since all semi-join projections are hashed, R_i needs to be scanned only once to process all these semi-joins. Each tuple in R_i is checked against the semi-join projections by using hashing. For each join attribute in the tuple, if a matching value can be found from the associated semi-join projection, then this tuple is included in the result. Otherwise, it is not.

III. Minimizing Query Response Time

In this section, we study the problem of using one-shot semi-join execution to minimize the response time of distributed query processing.

The response time for processing a distributed query includes the local processing time, the semi-join processing time, the time for transmitting the relations after semi-join reduction to a final site, and the final processing time. To simplify the discussion, the time for initial local processing is excluded from the response time model. Moreover, we refer to the "original size" of the relations as the one after initial local processing.

A semi-join reduction model and a response time model are given next. They will be used to formulate the one-shot semi-join optimization problem.

A. The Semi-Join Reduction Model

A selectivity model [3] has been developed to predict the reduction effect of semi-joins. Under this model, we may assume that a selectivity, ρ_i^j, is associated with each semi-join $R_j \ltimes R_i$. ρ_i^j is a rational number ranging from 0 to 1. After $R_j \ltimes R_i$ is executed, the size of R_j becomes $\rho_i^j \cdot \mid R_j \mid$, where $\mid R_j \mid$ denotes the original size of R_j. We further assume that semi-join reduction effects are independent, i.e., after the one-shot execution of a set of semi-joins $\{R_j \ltimes R_i \mid i \in S\}$, the size of R_j becomes $(\prod_{i\in S} \rho_i^j) \cdot \mid R_j \mid$.

B. The Response Time Model

Under the one-shot semi-join execution method, the response time for processing a query includes the time for generating and transmitting the semi-join projections, the time for executing the semi-joins, the time for transmitting the reduced relations to a final site, and the time for performing the joins. Each of these delays is described next.

Generating and Transmitting Semi-Join Projections: Since the time for generating the semi-join projections is dominated by the relation scan time and a relation needs to be scanned only once to generate all its semi-join projections, we assume that multiple projections on R_i can be generated in parallel. We also assume that multiple projections on R_i can be transmitted in parallel. This is true if the network has a high bandwidth and multiple channels, and the delay due to network contention is negligible. Finally, We assume that there is no data skew, i.e., semi-join projections are uniformly hashed into the hash tables.

If a semi-join $R_j \ltimes R_i$ is to be executed, R_i has to be projected, hashed, and transmitted to the site where R_j is located. We use s_i^j to denote the total time needed for this projection (including disk I/O), hashing, and transmission. Since we assumed that multiple projections on R_i can be generated and transmitted in parallel, no waiting is necessary when R_i has to be projected for another semi-join $R_k \ltimes R_i$.

Executing the Semi-Joins: The processing of semi-joins to R_j has to wait until all semi-join projections arrive. R_j is then scanned to perform the semi-joins and to ship the remaining tuples to the communication channel. Specifically, each tuple in R_j is checked against all the semi-join projections through hashing. If it does not have a matching value in each of the projections, it is discarded; else it is sent to the communication channel. We assume the time for hashing is negligible, and the time for scanning R_j is C_j'.

Transmitting the Relations: The communication channel transmits the reduced R_j to the final site after all tuples in R_j have been scanned. Suppose there are X_j tuples left. We assume the transmission time is $D_j' \cdot X_j + E_j'$, where D_j' and E_j' are constants corresponding to the speed of the transmission and the setup time, respectively.

The Final Join Processing Time: Let $R_1, R_2, \cdots, R_n$ be the relations sent to the final site for processing the joins. The joins are processed after all these relations arrive. In this paper, we assume the worst case time complexity of the join operation. That is, the amount of time needed for processing the joins is proportional to the product of the relation cardinalities. Thus, the final join processing time can be expressed as $F' \cdot \prod_{i=1}^n X_i$, where F' is a constant.

C. Formulation

We call i the relation index of R_i. A semi-join $R_j \ltimes R_i$ is feasible if and only if $R_j \bowtie R_i$ is implied in the given query. Let U_j be a set of relation indexes such that $i \in U_j$ if and only if $R_j \ltimes R_i$ is a feasible semi-join. The one-shot semi-join execution method chooses a subset B_j of U_j to reduce R_j. Our goal is to find $\{B_1, B_2, \cdots, B_n\}$ to minimize the overall response time.

We first consider the semi-join processing time for a single relation R_j. If $R_j \ltimes R_i$ is chosen, i.e., $i \in B_j$, then it takes s_i^j amount of time to generate and transmit the semi-join projection of R_i. Since semi-join projections can be generated and transmitted in parallel, the time for generating and transmitting all the semi-join projections to reduce R_j is $\max_{i\in B_j} s_i^j$. After all the semi-join projections arrive at the site where R_j is located, R_j is scanned once to process these semi-joins, which takes C_j' amount of time. The size of R_j after the semi-join processing becomes $\prod_{i\in B_j}(\rho_i^j) \cdot \mid R_j \mid$. The transmission time for sending the reduced R_j to a final site is therefore $D_j' \cdot \prod_{i\in B_j}(\rho_i^j) \cdot \mid R_j \mid + E_j'$. To summarize the above discussion, the total amount of time for R_j to be semi-joined and transmitted to the final site is $\max_{i\in B_j}(s_i^j) + C_j' + D_j' \cdot \prod_{i\in B_j}(\rho_i^j) \cdot \mid R_j \mid + E_j'$.

The processing of joins begins when all R_j's arrive at the final site. This needs $\max_{1\le j\le n}(\max_{i\in B_j}(s_i^j) + C_j' + D_j' \cdot \prod_{i\in B_j}(\rho_i^j) \cdot \mid R_j \mid + E_j')$ amount of time. The time for processing the joins is $F' \cdot \prod_{1\le j\le n}(\prod_{i\in B_j}(\rho_i^j) \mid R_j \mid)$ according to the final join processing time discussed in the last subsection. The overall response time can therefore be expressed as a function of $B_1, B_2, \cdots, B_n$ in the following:

$$RESP(B_1, B_2, \cdots, B_n) = \max_{1\le j\le n} (\max_{i\in B_j}(s_i^j) + C_j' + D_j' \cdot \prod_{i\in B_j} (\rho_i^j) \cdot \mid R_j \mid + E_j') + F' \cdot \prod_{j=1}^{n} (\prod_{i\in B_j} (\rho_i^j) \mid R_j \mid).$$

To simplify the above expression, we further make the following three definitions: 1) $C_j = C_j' + E_j'$, 2) $D_j = D_j' \cdot \mid R_j \mid$, and 3) $E = F' \cdot \prod_{j=1}^n(\mid R_j \mid)$. Minimizing the response time by one-shot semi-join execution can be stated as the following mathematical programming problem.

Definition 1 (RES): Given $P = (E, (C_1, D_1, \{(s_i^1, \rho_i^1) \mid i \in U_1\}), (C_2, D_2, \{(s_i^2, \rho_i^2) \mid i \in U_2\}), \cdots (C_n, D_n, \{(s_i^n, \rho_i^n) \mid i \in U_n\}))$, where E, C_j, D_j, s_i^j, ρ_i^j are positive rational numbers and $0 < \rho_i^j < 1$, find an optimal $B = (B_1, B_2, \cdots, B_n)$ such that

$$RE(B) = \max_{1\le j\le n} (\max_{i\in B_j}(s_i^j) + C_j + D_j \cdot \prod_{i\in B_j} (\rho_i^j)) + E \cdot \prod_{j=1}^{n} \prod_{i\in B_j} (\rho_i^j)$$

is minimized.

In the rest of the paper, we use $max_j(B_j)$, $prod_j(B_j)$, and $MAX(B)$ to denote $\max_{i\in B_j}(s_i^j)$, $\prod_{i\in B_j}(\rho_i^j)$, and $\max_{1\le j\le n}(\max_{i\in B_j}(s_i^j)+C_j+D_j\cdot\prod_{i\in B_j}(\rho_i^j))$, respectively.

Note that when $B_j = \emptyset$, $max_j(B_j)$ and $prod_j(B_j)$ are defined to be zero and one, respectively. This corresponds to the query processing strategy where no semi-join is applied to R_j. Hence, there is neither semi-join processing cost nor reduction effect on R_j.

IV. An Algorithm for RES

One way to solve RES is to examine all possible B's. Clearly this method is prohibited for the size of the search space is $(2^n)^n$. In this section, we shall explore the properties of RES and use them to reduce the search space to n^2. An efficient algorithm, P-RES, will then be presented. Its correctness and complexity will be studied and an example will be given.

A. Some Properties of the Optimal Solution

We first consider the one-shot semi-join execution on a single relation R_j. All subsets of U_j are candidates for the optimal B_j and there are $2^{|U_j|}$ of them ($| U_j |$ denotes the cardinality of U_j). To reduce this search space, we notice that if B_j is optimal and $i \in B_j$, then $\{h \mid s_h^j \le s_i^j\} \subset B_j$. Intuitively speaking, since $R_j \ltimes R_i$ is to be executed and the semi-join projections can be generated and transmitted in parallel, including another semi-join $R_j \ltimes R_h$ can further reduce the size of R_j without introducing extra delay as long as $s_h^j \le s_i^j$. From this observation, we can sort the elements in U_j into a list L' according to the increasing order of s_i^j. Let o_j be the sorting function, i.e., $s_{o_j(1)}^j \le s_{o_j(2)}^j \le \cdots \le s_{o_j(|U_j|)}^j$. The candidates for the optimal $B_j, b_k^j, 0 \le k \le| U_j |$, are defined in the following:

$$b_k^j = \begin{cases} \{o_j(1), o_j(2), \cdots, o_j(k)\} & \text{if } 1 \le k \le| U_j | \\ \emptyset & \text{if } k = 0. \end{cases}$$

The following lemma shows that the optimal B_j must be one of the b_k^j's.

Lemma 1: If $B = (B_1, B_2, \cdots, B_n)$ is optimal, then for all $1 \le j \le n$, $B_j \in \{b_0^j, b_1^j, \cdots, b_{|U_j|}^j\}$.

Proof: If B_j is $\emptyset$, then it equals b_0^j; else there is a maximum index k such that $o_j(k) \in B_j$.

Suppose $B_j \ne b_k^j$. There must exist $k' < k$ such that $o_j(k') \notin B_j$. Consider $B' = (B_1, ,B_2, ,\cdots, B_j \bigcup\{o_j(k')\}, \cdots, B_n)$. Note that $s_{o_j(k')}^j \le s_{o_j(k)}^j$ because $k' < k$. Since $0 < \rho_{o_j(k')}^j < 1$, $prod_j(B_j \bigcup\{o_j(k')\}) = \rho_{o_j(k')}^j \cdot prod_j(B_j) < prod_j(B_j)$. Also note that $\max_j(B_j \bigcup\{o_j(k')\}) = \max_j(B_j) = s_{o_j(k)}^j$. Therefore, $RE(B') < RE(B)$ and B is not optimal. Q.E.D.

Given b_k^j, we define the selectivity of b_k^j, denoted by p_k^j, to be the product of the selectivities of all the semi-joins in b_k^j. p_k^j can be expressed as

$$p_k^j = \begin{cases} \prod_{i\in b_k^j} \rho_i^j & \text{if } 1 \le k \le| U_j | \\ 1 & \text{if } k = 0. \end{cases}$$

Note that after the execution of b_k^j, the size of R_j becomes p_k^j times of its original size. We further define v_k^j to be the amount of time needed for executing the semi-joins in b_k^j, and transmitting the resultant R_j to the final processing site. v_k^j can be expressed as

$$v_k^j = \begin{cases} s_{o_j(k)}^j + C_j + D_j \cdot p_k^j & \text{if } 1 \le k \le| U_j | \\ C_j + D_j & \text{if } k = 0. \end{cases}$$

From Lemma 1, an optimal B can be expressed as $(b_{k_1}^1, b_{k_2}^2, \cdots, b_{k_n}^n)$ and $RE(B)$ can be rewritten as

$$RE(B) = \max_{1\le j\le n} v_{k_j}^j + E \cdot \prod_{j=1}^{n} p_{k_j}^j. \quad (1)$$

The search space for the optimal B can be further reduced. Let m be the subscript such that $v_{k_m}^m = \max_{1\le j\le n} v_{k_j}^j$. In other words, R_m is the last relation to arrive at the final site. Therefore, for any relation R_h (including R_m), $b_{k_h}^h$ must be chosen such that $v_{k_h}^h \le v_{k_m}^m$. However, among all b_a^h which satisfy this requirement, $b_{k_h}^h$ has to be the one with the smallest selectivity because the final size of R_h can thus be most reduced without incurring extra delay. Applying this observation, we may search for the optimal B by first sorting $\{b_k^j \mid 1 \le j \le n, 0 \le k \le| U_j |\}$ into a list L according to the increasing order of v_k^j. L is then scanned iteratively. During iteration i, only the first i elements in L are considered. Specifically, for each relation R_h, $b_{k_h}^h$ is chosen, among the first i elements in L, to be the b_a^h with the smallest selectivity.

A few useful notations are defined next. ($j()$, $k()$) denotes the sorting function which generates L. That is, $v_{k(1)}^{j(1)} \le v_{k(2)}^{j(2)} \le \cdots \le v_{k(T)}^{j(T)}$, where $T = \sum_{h=1}^{n}(| U_h | +1)$. We use $jk^{-1}()$ to denote the inverse of ($j()$, $k()$). That is, $jk^{-1}(j, k) = i$ means b_k^j is the ith element in L. We use a set $K_h(i)$ to specify which ones from the first i elements in L are candidates for $b_{k_h}^h$. More precisely, an index d is an element of $K_h(i)$ if and only if $d \le i$ and $j(d) = h$. $m_h(i)$ denotes the one in $K_h(i)$ with the smallest selectivity. That is, $m_h(i) \in K_h(i)$ and $p_{k(m_h(i))}^h = \min_{\forall m\in K_h(i)} p_{k(m)}^h$. $m_h(i)$ is undefined if $K_h(i) = \emptyset$. The candidate for the optimal solution at iteration i is $B(i) = (b_{k(m_1(i))}^1, b_{k(m_2(i))}^2, \cdots, b_{k(m_n(i))}^n)$. $B(i)$ is undefined if there exists h such that $m_h(i)$ is undefined, i.e., none of the first i elements in L is a candidate for the optimal B_h. We define $i_{\min}$ to be the the minimum index such that $B(i_{\min})$ is defined. Note that for any $i > i_{\min}$, $B(i)$ is always defined. Since it is possible that the candidate solutions at different iterations are the same, we say $B(i)$ is *minimal* if for all $i' < i$, $B(i') \ne B(i)$.

The following lemma states that one of the $B(i)$'s is the optimal solution.

Lemma 2: There exists i, $i_{\min} \le i \le T$, such that $B(i)$ is optimal and minimal.

Proof: Let B be an optimal solution. By Lemma 1, $B = (b_{k_1}^1, b_{k_2}^2, \cdots, b_{k_n}^n)$. Let $i = \max_{1\le g\le n} jk^{-1}(g, k_g)$. Clearly, $i_{\min} \le i \le T$. We shall first prove $B(i)$ is optimal by showing $RE(B(i)) \le RE(B)$.

By the definition of i there is h such that $(k(i), j(i)) = (k_h, h)$. So $MAX(B) = \max_{1\le g\le n} v_{k_g}^g \ge v_{k_h}^h$. But for all

$1 \leq g \leq n$, $m_g(i) \leq i$. Thus $v^g_{k(m_g(i))} \leq v^{j(i)}_{k(i)} = v^h_{k_h}$. Consequently, $MAX(B(i)) \leq v^h_{k_h} \leq MAX(B)$. From (1), it remains to show that for all g, $p^g_{k(m_g(i))} \leq p^g_{k_g}$. We note that $i \geq jk^{-1}(g, k_g)$, which implies $jk^{-1}(g, k_g) \in K_g(i)$. By the definition of $m_g(i)$, $p^g_{k(m_g(i))} \leq p^g_{k_g}$.

We have shown that $B(i)$ is optimal. If it is not minimal, let i' be the smallest index such that $B(i') = B(i)$. Then $B(i')$ is both optimal and minimal. Q.E.D.

The next lemma describes some properties of the $B(i)$ which were used in the design of P-RES.

Lemma 3:

1) $B(i)$ is minimal if and only if $m_{j(i)}(i) = i$.
2) If $B(i)$ is minimal, then $MAX(B(i)) = v^{j(i)}_{k(i)}$ and $RE(B(i)) = v^{j(i)}_{k(i)} + E \cdot \prod^n_{h=1} p^h_{k(m_h(i))}$.

Proof:

1) $m_h(i) < i$ if $h \neq j(i)$. Therefore, $m_{j(i)}(i) \neq i$ implies $B(i) = B(i-1)$ and $B(i)$ is not minimal. Conversely, since $i \notin K_{j(i)}(i')$ for all $i' < i$, $m_{j(i)}(i) = i$ implies $m_{j(i)}(i') \neq m_{j(i)}(i)$. Thus, $B(i) \neq B(i')$ and $B(i)$ must be minimal.
2) From (1), it suffices to show $v^{j(i)}_{k(i)} = MAX(B(i)) = \max_{1 \leq h \leq n} v^h_{k(m_h(i))}$. Note that $i' \leq i$ implies $v^{j(i')}_{k(i')} \leq v^{j(i)}_{k(i)}$. But by definition $m_h(i) \leq i$ for all $1 \leq h \leq n$. So $v^h_{k(m_h(i))} \leq v^{j(i)}_{k(i)}$ for all h. On the other hand, $B(i)$ is minimal implies $m_{j(i)}(i) = i$ and therefore $v^{j(i)}_{k(i)} = v^{j(i)}_{k(m_{j(i)}(i))}$. Accordingly, $v^{j(i)}_{k(i)} = \max_{1 \leq h \leq n} v^h_{k(m_h(i))}$. Q.E.D.

Let i_{opt} be the smallest index such that $B(i_{opt})$ is both optimal and minimal. Algorithm P-RES iterates to find $MAX(B(i_{opt}))$, and $RE(B(i_{opt}))$. During each iteration i, $RE(B(i))$ is calculated only if $B(i)$ is minimal. If $B(i)$ turns out to be the best candidate solution found so far, then both $RE(B(i))$ and $MAX(B(i))$ are recorded. To calculate $RE(B(i))$, $\prod^n_{h=1} p^h_{k(m_h(i))}$ needs to be computed. Instead of taking $(n-1)$ multiplications in each iteration, we use an array $p(h)$, $1 \leq h \leq n$, to store $p^h_{k(m_h(i))}$. Another variable $PROD$ is used to record $\prod_{1 \leq h \leq n} p^h_{k(m_h(i))}$. We notice that $m_h(i+1) = m_h(i)$ except when $h = j(i+1)$, in which case $m_{j(i+1)}(i+1) = i+1$ if $p^{j(i+1)}_{k(m_{j(i+1)}(i))} > p^{j(i+1)}_{k(i+1)}$. Therefore, only one of the $p(h)$'s, namely $p(j(i+1))$, may be changed during iteration $i+1$. If $p(j(i+1))$ is changed, $\prod_{1 \leq h \leq n} p^h_{k(m_h(i+1))}$ can be evaluated as

$$\Big(\prod_{1 \leq h \leq n} p^h_{k(m_h(i))}\Big) \cdot p^{j(i+1)}_{k(i+1)} / p^{j(i+1)}_{k(m_{j(i+1)}(i))}$$

$$= PROD^i \cdot \frac{p^{j(i+1)}_{k(i+1)}}{p^i(j(i+1))}$$

where $PROD^i$ and $p^i(j(i+1))$ are the values stored in $PROD$ and $p(j(i+1))$, respectively, after iteration i. Thus, only one multiplication and one division are required in order to compute $\prod_{1 \leq h \leq n} p^h_{k(m_h(i+1))}$.

After determining $MAX(B(i_{opt}))$, P-RES enters the second phase to construct $B(i_{opt})$ ($B(i_{opt})$ is not recorded during the first phase). Specifically, we use the fact that $k(m_h(i_{opt}))$ should be the biggest index such that $v^h_{k(m_h(i_{opt}))} \leq MAX(B(i_{opt}))$. This is formally stated in the following lemma:

Lemma 4: If $B(i)$ is optimal, then $k(m_h(i)) = \max_{0 \leq g \leq |U_h|, v^h_g \leq MAX(B(i))} g$.

Proof: It is clear that $v^h_{k(m_h(i))} \leq \max_{1 \leq c \leq n} v^c_{k(m_c(i))} = MAX(B(i))$. We shall show that there does not exist $d > k(m_h(i))$ such that $v^h_d \leq MAX(B(i))$. Suppose there is such d. Consider $B' = (b^1_{k_1}, b^2_{k_2}, \cdots, b^n_{k_n})$ where

$$k_a = \begin{cases} k(m_a(i)) & \text{if } a \neq h \\ d & \text{if } a = h. \end{cases}$$

$MAX(B') \leq MAX(B(i))$ because $v^h_d \leq MAX(B(i))$. Furthermore, $k(m_h(i)) < d$ implies $b^h_{k(m_h(i))} \subset b^h_d$, and therefore $p^h_{k(m_h(i))} > p^h_d$. Hence, $RE(B') < RE(B(i))$ and $B(i)$ is not optimal. Q.E.D.

P-RES is formally stated on the next page.

B. Correctness

Some properties of the "for loop" starting at line 8 are studied first. We define $p^i(h)$, MAX^i, $PROD^i$, and RE^i to be the values stored in $p(h)$, MAX, $PROD$, and RE after iteration i if $i_{\min} < i \leq T$, and to be the initial values (after line 7) stored in $p(h)$, MAX, $PROD$, and RE if $i = i_{\min}$.

Lemma 5: For all h, $1 \leq h \leq n$, and for all i, $i_{\min} \leq i \leq T$. $p^i(h) = p^h_{k(m_h(i))}$.

Proof: The proof is by induction on i. From line 5, the hypothesis is true when $i = i_{\min}$.

In the induction step we assume the hypothesis is true when $i_{\min} \leq i \leq d$. Consider when $i = d+1$. For any h, $1 \leq h \leq n$, there are two cases:

1) $j(d+1) \neq h$: $p(h)$ is not updated during iteration $i = d+1$ and therefore $p^{d+1}(h) = p^d(h)$. Also, $p^h_{k(m_h(d))} = p^h_{k(m_h(d+1))}$ because $d+1 \notin K_h(d+1)$. But $p^d(h) = p^h_{k(m_h(d))}$ from the induction hypothesis. Thus, $p^{d+1}(h) = p^d(h) = p^h_{k(m_h(d))} = p^h_{k(m_h(d+1))}$.
2) $j(d+1) = h$: From the induction hypothesis $p^d(h) = p^h_{k(m_h(d))}$. Then from line 11 and line 18, $p^{d+1}(h) = \min\{p^h_{k(m_h(d))}, p^{j(d+1)}_{k(d+1)}\}$. But since $K_h(d+1) = K_h(d) \bigcup \{d+1\}$, $\min\{p^h_{k(m_h(d))}, p^{j(d+1)}_{k(d+1)}\} = p^h_{k(m_h(d+1))}$. Hence, $p^{d+1}(h) = p^h_{k(m_h(d+1))}$. Q.E.D.

Lemma 6: For all i, $i_{\min} \leq i \leq T$, $PROD^i = \prod^n_{h=1} p^i(h)$.

Proof: The proof is by induction on i. If $i = i_{\min}$, then $PROD = \prod_{1 \leq h \leq n} p(h)$ from line 6.

Assume the hypothesis is true when $i = d$. Consider $i = d+1$. There are two cases:

1) *Line 13 and line 18 are executed during iteration $i = d+1$:* Since line 18 is executed, $p^{d+1}(j(d+1)) = p^{j(d+1)}_{k(d+1)}$. Note that $p^{d+1}(h) = p^d(h)$ if $h \neq j(d+1)$. Thus, $\prod^n_{h=1} p^{d+1}(h) = (\prod^n_{h=1} p^d(h)) \cdot p^{j(d+1)}_{k(d+1)} / p^d(j(d+1))$. From the induction hypothesis, $PROD^d = \prod_{1 \leq h \leq n} p^d(h)$. Consequently, $\prod^n_{h=1} p^{d+1}(h) = PROD^d \cdot p^{j(d+1)}_{k(d+1)} / p^d(j(d+1))$. The

ALGORITHM P-RES

```
0)  input $(n,\ E,\ \{(C_j, D_j, U_j) \mid 1 \le j \le n\},\ \{(s_i^j,\ \rho_i^j) \mid 1 \le j \le n, i \in U_j\})$
1)  $1 \le j \le n$, sort $\{s_i^j \mid i \in U_j\}$ to compute $p_k^j,\ v_k^j$.
2)  $T \leftarrow n + \sum_{a=1}^{n} \mid U_a \mid$;
3)  derive the sorting function (k(),j()) such that $v_{k(1)}^{j(1)} \le v_{k(2)}^{j(2)} \le \cdots \le v_{k(T)}^{j(T)}$;
4)  find the minimum $i$, denoted by $i_{\min}$, such that $\forall 1 \le h \le n, \exists i_h, i_h \le i$ and $j(i_h) = h$;
5)  $1 \le h \le n,\ p(h) \leftarrow \min_{\forall i,\ i \le i_{\min} \wedge j(i)=h} p_{k(i)}^{j(i)}$;
6)  $PROD \leftarrow \prod_{h=1}^{n} p(h)$;
7)  $MAX \leftarrow v_{k(i_{\min})}^{j(i_{\min})}$; $RE \leftarrow v_{k(i_{\min})}^{j(i_{\min})} + E \cdot PROD$;
8)  for $i \leftarrow i_{\min} + 1$ to $T$
9)  begin
10)     $(k,j) \leftarrow (k(i), j(i))$;
11)     if $(p_k^j < p(j))$ then
12)     begin
13)         $PROD \leftarrow PROD \cdot p_k^j / p(j)$;
14)         if ( $RE > v_k^j + E \cdot PROD$) then
15)         begin
16)             $MAX \leftarrow v_k^j$; $RE \leftarrow v_k^j + E \cdot PROD$;
17)         end {if}
18)         $p(j) \leftarrow p_k^j$;
19)     end {if}
20) end {for}
21) for $j \leftarrow 1$ to $n$
22) begin
23)         find the maximum k such that $v_k^j \le MAX$;
24)         $B_j \leftarrow b_k^j$;
25) end {for}
26) output ( $B_1, B_2, \cdots, B_n$; $RE$)
```

hypothesis is therefore true when $i = d + 1$ according to line 13.

2) *Line 13 and 18 are not executed:* Then $p^{d+1}(h) = p^d(h)$ for all h and $PROD^{d+1} = PROD^d$. Hence $PROD^{d+1} = \prod_{h=1}^{n} p^d(h) = \prod_{h=1}^{n} p^{d+1}(h)$. Q.E.D.

RE and MAX are initialized at line 7 and updated at line 16. The next two lemmas study when these two variables are updated and what values they record.

Lemma 7: For all i, $i_{\min} + 1 < i \le T$, the "if" statement at line 14 is executed during iteration i if and only if $B(i)$ is minimal.

Proof: From Lemma 3, $B(i)$ is minimal if and only if $m_{j(i)}(i) = i$. For all $i > i_{\min}$, $m_{j(i)}(i) = i$ if and only if $p_{k(i)}^{j(i)} < p_{k(m_{j(i)}(i-1))}^{j(i)}$. But from Lemma 5, $p_{k(m_{j(i)}(i-1))}^{j(i)} = p^{i-1}(j(i))$. Thus, $B(i)$ is minimal if and only if the "if condition" at line 11 becomes true during iteration i. Finally, note that line 14 is executed if and only if the "if condition" at line 11 is true. Q.E.D.

Lemma 8: For all i, $i_{\min} + 1 < i \le T$, if line 16 is executed during iteration i, then $RE^i = RE(B(i))$ and $MAX^i = MAX(B(i))$.

Proof: line 16 is executed only if line 14 is executed. $B(i)$ is therefore minimal from Lemma 7. Then by Lemmas 3, 5, and 6, $RE^i = RE(B(i))$ and $MAX^i = MAX(B(i))$. Q.E.D.

Recall i_{opt} denotes the smallest index such that $B(i_{opt})$ is both optimal and minimal. Lemma 2 guarantees the existence of i_{opt}. The next lemma shows that after the completion of the "for" loop starting at line 8, $RE(B(i_{opt}))$ and $MAX(B(i_{opt}))$ are stored in RE and MAX, respectively.

Lemma 9: $RE^T = RE(B(i_{opt}))$ and $MAX^T = MAX(B(i_{opt}))$

Proof: It suffices to prove that 1) $RE^{i_{opt}} = RE(B(i_{opt}))$ and $MAX^{i_{opt}} = MAX(B(i_{opt}))$, and 2) line 16 will not be executed after iteration $i = i_{opt}$.

We first prove that $RE^{i_{opt}} = RE(B(i_{opt}))$ and $MAX^{i_{opt}} = MAX(B(i_{opt}))$. If $i_{opt} = i_{\min}$, then $RE^{i_{\min}} = RE(B(i_{opt}))$ and $MAX^{i_{\min}} = MAX(B(i_{opt}))$ from line 7. If $i_{opt} > i_{\min}$, then from Lemma 8, it is enough to show that line 16 will be executed during iteration $i = i_{opt}$. Since $B(i_{opt})$ is minimal, line 14 will be executed from Lemma 7. Let $i = i'$ be the iteration during which RE is last updated before iteration $i = i_{opt}$. If there is no such i', let $i' = i_{\min}$. Note that $RE^{i_{opt}-1} = RE(B(i'))$. But $B(i')$ cannot be optimal because i_{opt} is the smallest index such that $B(i_{opt})$ is optimal. Accordingly, $RE^{i_{opt}-1} > RE(B(i_{opt}))$ and the "if condition" at line 14 becomes true during iteration $i = i_{opt}$. Line 16 will therefore be executed.

It remains to prove that line 16 will not be executed after iteration $i = i_{opt}$. During any iteration $i = i'$, $i' > i_{opt}$, line 16 is executed only if $RE^{i'-1} > RE(B(i'))$ (from line 14). But from line 14 and line 16, the value stored in RE never increases during the execution of the "for" loop. Since the

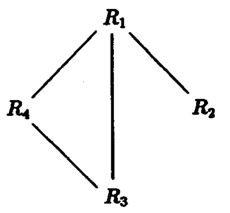


Fig. 1. An example query.

minimum value, i.e., $RE(B(i_{opt}))$, has already been stored in RE during iteration $i = i_{opt}$, $RE^{i'-1} \leq RE(B(i'))$. Accordingly, line 16 will not be executed during iteration $i = i'$. Q.E.D.

Theorem 1: P-RES is correct.

Proof: From Lemma 9, $RE^T = RE(B(i_{opt}))$ and $MAX^T = MAX(B(i_{opt}))$. From Lemma 4, the "for" loop starting at line 21 computes $b^h_{k(m_h(i_{opt}))}$, for $1 \leq h \leq n$. The output of P-RES is therefore $B(i_{opt})$ together with $RE(B(i_{opt}))$. Q.E.D.

C. The Complexity of P-RES

We assume it takes constant time to add, multiply, or divide two rational numbers. To compute p^j_k's, and v^j_k's, one needs to find the sorting functions $\{o_j() \mid 1 \leq j \leq n\}$. Thus, it takes $O(n^2 \log n)$ time to execute line 1. Similarly, it takes $O(n^2 \log n)$ time to execute line 3. Line 4 and 5 takes $O(n^2)$ time. The "for" loop starting at line 8 iterates n^2 times, each iteration takes $O(1)$ time. The "for" loop staring at line 21 iterates n times. During each iteration, it takes $O(n)$ time to find k and to construct b^j_k. To summarize, The overall time complexity of P-RES is $O(n^2 \log n)$.

We assume it takes constant space to store a rational number. The space complexity for storing s^j_i's, ρ^j_i's, v^j_k's, and p^j_k's is $O(n^2)$. b^j_k is only used at line 24 and it can be constructed by using the ordering function $o_j()$. Therefore, no extra space is needed. We need $O(n^2)$ space to hold B_j, $1 \leq j \leq n$. The overall space complexity of P-RES is $O(n^2)$.

D. An Example

We use an example to illustrate P-RES. The example query is represented as the join graph in Fig. 1:

The parameters are:

1) $E = 15$
2) s^j_i's and ρ^j_i's are given in Table I.
3) C_j's and D_j's are given in Table II:

P-RES first compute $o_j(k)$, p^j_k, and v^j_k. They are recorded in Table III. Take $(j,k) = (1,2)$ for example. From Table I, $s^1_2 < s^1_4 < s^1_3$. Thus, $o_1(2) = 4$; $p^1_2 = \rho^1_2 \cdot \rho^1_4 = 0.75 \cdot 0.8 = 0.6$; and $v^1_2 = s^1_4 + C_1 + D_1 \cdot p^1_2 = 7.3$.

After sorting v^j_k's, P-RES starts to scan the sorted list L at lines 4 and 5. From the first four elements in L (Table IV), $i_{\min}$ is 4. After line 7, We have $p(1) = 0.3$, $p(2) = 1$, $p(3) = 1$, $p(4) = 0.36$, $PROD = 0.108$, $MAX = v^3_0 = 6.5$, and $RE = 6.5 + 15 \cdot 0.108 = 8.12$. The execution of

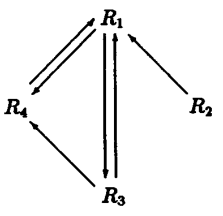


Fig. 2. The optimal set of semi-joins.

TABLE I
THE VALUES OF s^j_i'S AND ρ^j_i'S IN THE EXAMPLE

$(s^j_i, \rho^j_i) =$

$i\backslash j$	1	2	3	4
1	$\perp$	1.5, 0.9	1.2, 0.6	1.5, 0.9
2	1, 0.75	$\perp$	$\perp$	$\perp$
3	2, 0.5	$\perp$	$\perp$	2, 0.4
4	1.8, 0.8	$\perp$	2.5, 0.5	$\perp$

$\perp$ denotes "undefined."

TABLE II
THE VALUES OF C_j'S AND D_j'S IN THE EXAMPLE

j	C_j	D_j
1	2.5	5
2	3.4	3
3	4.5	2
4	3	4

the "for loop" starting at line 8 is summarized in Table V. Take iteration $i = 11$ for example. Since $p^2_1 = 0.9 < 1 = p(2)$, $PROD = 0.0324 \cdot 0.9 = 0.02916$. $v^2_1 + E \cdot PROD = 7.6 + 15 \cdot 0.02916 > 7.872$. Thus, RE remains unchanged.

The final value of MAX is 6.9. The "for" loop at line (21) generates the following B_j's: $B_1 = b^1_3 = \{2,3,4\}$; $B_2 = b^2_0 = \emptyset$; $B_3 = b^3_1 = \{1\}$; $B_4 = b^4_2 = \{1,3\}$. The optimal set of semi-joins for minimizing the response time is represented as the semi-join graph in Fig. 2, where $R_i \rightarrow R_j$ denotes $R_j \ltimes R_i$.

V. CONCLUSIONS

We employed the one-shot semi-join execution strategy to optimize the response time for processing distributed queries. This strategy selects a set of semi-joins as the semi-join program, and executes these semi-joins in parallel in three phases: the projection phase, the transmission phase, and the reduction phase. A response time model was established, which considers the semi-join processing time, the time for transmitting the relations to a final site, and the final processing time. This model takes into account the parallelism of local processing and data transmission. A polynomial-time algorithm was then developed based on this response time model.

The proposed algorithm may be generalized. Specifically, we like to consider the average processing time for the final

TABLE III
THE VALUES OF $o_j(k)$'S, p_k^j'S, AND v_k^j'S

$(o_j(k), p_k^j, v_k^j) =$

$k \backslash j$	1	2	3	4
0	$\perp$, 1, 7.5	$\perp$, 1, 6.4	$\perp$, 1, 6.5	$\perp$, 1,7
1	2, 0.75, 7.25	1, 0.9, 7.6	1, 0.6, 6.9	1, 0.9, 8.1
2	4, 0.6, 7.3	$\perp$	4, 0.3, 7.6	3, 0.36, 6.44
3	3, 0.3, 6	$\perp$	$\perp$	$\perp$

TABLE IV
THE ELEMENTS IN L WHICH DETERMINE $i_{\min}$

i	$(k(i), j(i))$	$v_{k(i)}^{j(i)}$	$p_{k(i)}^{j(i)}$
1	(3,1)	6	0.3
2	(0,2)	6.4	1
3	(2,4)	6.44	0.36
4	(0,3)	6.5	1

TABLE V
THE SUMMARY OF THE EXECUTION OF THE "FOR LOOP" AT LINE 8

i	$(k(i), j(i))$	$v_{k(i)}^{j(i)}$	$p_{k(i)}^{j(i)}$	$PROD$	MAX	RE
5	(1, 3)	6.9	0.6	0.0648	6.9	7.872
6	(0, 4)	7	1	0.0648	6.9	7.872
7	(1, 1)	7.25	0.75	0.0648	6.9	7.872
8	(2, 1)	7.3	0.6	0.0648	6.9	7.872
9	(0, 1)	7.5	1	0.0648	6.9	7.872
10	(2, 3)	7.6	0.3	0.0324	6.9	7.872
11	(1, 2)	7.6	0.9	0.02916	6.9	7.872
12	(1, 4)	8.1	0.9	0.02916	6.9	7.872

joins and relax the assumption that semi-join reduction effects are independent.

We proposed to use hashing to process multiple semi-joins in this paper. The size of the hash tables and the reduction of the relations represent a tradeoff for the selection of the hash functions. We intend to investigate this issue under the same semi-join execution strategy and response time model.

REFERENCES

[1] P. M. G. Apers, A. R. Hevner, and S. B. Yao, "Optimization algorithms for distributed queries," *IEEE Trans. Software Eng.*, vol. SE-9, pp. 57–68, Jan. 1983.

[2] P. A. Bernstein and D. M. Chiu, "Using semi-joins to solve relational queries," *J. ACM*, vol. 28, no. 1, pp. 25–40, Jan. 1981.

[3] P. A. Bernstein and N. Goodman, "The power of natural semijoins," *SIAM J. Comput.*, vol. 10, no. 4, pp. 751–771, Nov. 1981.

[4] P. A. Bernstein *et al.*, "Query processing in a system for distributed databases (SDD-1)," *ACM Trans Database Syst.*, vol. 6, no. 4, pp. 602–625, Dec. 1981.

[5] P. A. Black and W. S. Luk, "A new hueristic for generating semi-join programs for distributed query processing," in *Proc. IEEE COMPSAC*, Dec. 1982, pp. 581–588.

[6] D. Brill, M. Templeton, and C. T. Yu, "Distributed query processing strategies in Mermaid, A frontend to data management systems," in *Proc. IEEE Data Eng. Conf.*, Feb. 1984.

[7] S. Ceri and G. Pelagatti, *"Distributed Databases: Principles and Systems.* New York: McGraw-Hill, 1984.

[8] A. L. P. Chen, D. Brill, M. Templeton, and C. T. Yu, "Distributed query processing in a multiple database system," *IEEE J. Select. Areas Commun.*, issue on Databases in Communications Systems, Apr. 1989.

[9] A. L. P. Chen and V. O. K. Li, "Improvement algorithms for semi-join query processing programs in distributed database systems," *IEEE Trans. Comput.*, vol. C-33, pp. 959–967, Nov. 1984.

[10] —— "An optimal algorithm for distributed star queries," *IEEE Trans. Software Eng.*, vol. SE-11, pp. 1097–1107, Oct. 1985.

[11] D. M. Chiu, P. A. Bernstein, and Y. C. Ho, "Optimizing chain queries in a distributed database system," *SIAM J. Comput.*, vol. 13, no. 1, pp. 116–134, Feb. 1984.

[12] D. M. Chiu and Y. C. Ho, "A method for interpreting tree queries into optimal semi-join expressions," in *Proc. ACM SIGMOD Int. Conf. Mananagement Data*, 1980.

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

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

[15] W. Perrizo and C. Chen, "Composite semijoins in distributed query processing," *Inform. Sci.*, Mar. 1990.

[16] S. Pramanik and D. Vineyard, "Optimizing join queries in distributed databases," *IEEE Trans. Software Eng.*, vol. SE-14, pp. 1319–1326, Sept. 1988.

[17] M. Templeton, D. Brill, A. L. P. Chen, S. Dao, and E. Lund, "Mermaid—Experiences with network operation," in *Proc. 2nd IEEE Data Eng. Conf.*, Feb. 1986.

[18] C. P. Wang and V. O. K. Li, "The relation-partitioning approach to distributed query processing," in *Proc. 2nd IEEE Data Eng. Conf.*, Feb. 1986.

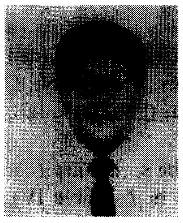
[19] C. P. Wang, V. O. K. Li, and A. L. P. Chen, "One-shot semi-join execution strategies for processing distributed queries," in *Proc. 7th IEEE Data Eng. Conf.*, Apr. 1991.

[20] E. Wong and K. Youssefi, "Decomposition—A strategy for query processing," *ACM Trans. Database Syst.*, 1976.

[21] C. T. Yu, K. Guh, and A. L. P. Chen, "An integrated algorithm for distributed query processing," in *Proc. IFIP Conf. Distributed Processing*, Oct. 1987.

[22] C. T. Yu, Z. M. Ozsoyoglu, and K. Kam, "Optimization of distributed tree queries," *J. Comput. Syst. Sci.*, vol. 29, no. 3, pp. 409–445, Dec. 1984.

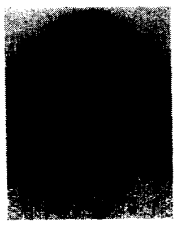
[23] C. T. Yu *et al.*, "Query processing in a fragmented relational distributed system: Mermaid," *IEEE Trans. Software Eng.*, Aug. 1985.



Chihping Wang (S'84–M'88) received the B.S. degree in electrical engineering from National Taiwan University, Taiwan, Republic of China, in 1983, and the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, in 1988.

Since September, 1988 he has been with the Department of Computer Science, University of California, Riverside, where he is an Assistant Professor. His research interests include distributed databases, object-oriented databases, distributed systems, and performance modeling.

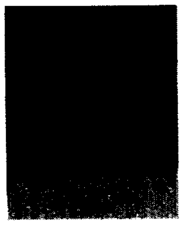
Dr. Wang is a member of the Association for Computing Machinery and the IEEE Computer Society.



Arbee L. P. Chen (S'80–M'84) received the B.S. degree in computer science from National Chiao Tung University, Taiwan, Republic of China, in 1977, and the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, in 1984.

He is a Professor in the Department of Computer Science, National Tsing Hua University, Taiwan. He was a Member of Technical Staff at Bell Communications Research, NJ, from 1987 to 1990, an Adjunct Associate Professor in the Department of Electrical Engineering and Computer Science, Polytechnic University, Brooklyn, NY, and a Research Scientist at Unisys, Santa Monica, CA, from 1985 to 1986. He is currently also an Advisor to Industrial Technology Research Institute and Institute for Information Industry in Taiwan. His research interests include distributed databases, heterogeneous databases, active databases, and distributed computing systems. He has published over 30 papers in various journals and conference proceedings.

Dr. Chen is a member of the Association for Computing Machinery and the IEEE Computer Society, and was a member of the ANSI/X3/SPARC/Database Systems Study Group.



Shiow-Chen Shyu was born in Panchiou, Taiwan, in 1961. She received the B.S. degree in computer engineering from National Chiao Tung University, Taiwan, in 1984, and the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, in 1988.

She is currently a staff member at the IBM Santa Teresa Laboratory, San Jose, CA. Her research interests include distributed algorithms, distributed databases, performance modeling, object-oriented databases, and real-time systems.