Distributed Query Processing in a Multiple Database System

ARBEE L. P. CHEN, MEMBER, IEEE, DAVID BRILL, MARJORIE TEMPLETON, AND CLEMENT T. YU

Abstract—**Mermaid is a testbed system which provides integrated access to multiple databases. We have developed two query optimization algorithms for Mermaid. The semijoin algorithm tends to reduce the data transmission cost while the replicate algorithm reduces the processing cost. In this paper, we present an algorithm which integrates the features of these two algorithms to optimize the processing cost as well as the transmission cost. Particularly, we consider a dynamic network environment where processing speeds at each site and transmission speeds at each link can be variable. Moreover, distributed processing of aggregates is considered based on the functional dependency among the fragment attribute, the aggregate attribute, and the group-by attribute. We also apply semantic information for efficient query processing.**

I. Introduction

MERMAID is a testbed system which runs on top of multiple databases stored in different data management systems (DBMS's) in network computer sites [20]. Although many of the DBMS functions are actually provided by the underlying DBMS's, Mermaid appears to the users to be an extended DBMS which allows them to access data from multiple databases. The goal of Mermaid is to provide integrated access to these databases using a common language, either ARIEL [18] or SQL.

Database users may need not only their own data, but also data in other databases to solve a specific problem. Data may reside in different databases for many reasons such as ownership, security classification, performance, or size. Data may be stored redundantly in different computers for reliability or survivability. In addition, hardware or software upgrades may create a need for integrated access to both old and new databases or for a tool to aid data migration from an old to a new system.

Distributed query processing has been considered strongly related to the performance efficiency of a system with the databases distributed in a network. Many distributed query processing algorithms [1], [3], [4], [6]–[12], [16], [22], [23], [25], [28] have been proposed. Most of these algorithms assume that the data communication cost is dominant and make use of semijoins [2] to reduce the amount of data transfer. While such an assumption is reasonable for long-haul networks where data communication costs are high, it may not be valid for fast local networks. In contrast, the *fragment and replicate* query processing strategy was used in distributed INGRES [12]. Its main goal is to achieve a high degree of parallelism by partitioning one relation among the processing sites and replicating all other needed relations at each processing site. However, for many queries, substantial data transfer is required before parallel processing can take place unless many relations are duplicated in many sites.

Two algorithms have been developed for Mermaid. The first algorithm, the semijoin algorithm [24], [25], is an extension to the SDD-1 algorithm [3] which assumes that the most important cost is the number of bytes transmitted between network sites. The algorithm was extended to support fragmented and replicated relations. This algorithm is one of the most complete algorithms in the current literature [14], [17] and one of the few that have been implemented and tested. The other algorithm, the replicate algorithm [5], [27] is derived from distributed INGRES. It assumes that CPU overhead dominates network costs and uses fragmented relations to maximize the amount of parallelism in operations.

A performance analysis has been done for these two algorithms in the Mermaid testbed environment [5], [19]. It was found that the replicate algorithm outperforms the semijoin algorithm in this environment which is based on a local area network. In this paper, an improved version of the replicate algorithm, named the *integrated algorithm* will be presented. The integrated algorithm considers minimizing the processing cost as well as the network cost, provides distributed processing of aggregates, and combines the features used in the semijoin and replicate algorithms. Query response time is the optimization criterion of the integrated algorithm.

An outline of this paper is as follows. In Section II, the integrated algorithm is presented. The major parts of this algorithm, i.e., the heuristic for minimizing processing cost as well as network cost, semijoin applications, and distributed aggregate processing are further addressed in Sections III, IV, and V, respectively. We conclude in Section VI.

II. The Integrated Algorithm

The integrated algorithm consists of the following seven steps:

Manuscript received November 20, 1987; revised September 6, 1988.

A. L. P. Chen is with Bell Communications Research, Piscataway, NJ 08854.

D. Brill is with Information Sciences Institute, Marina del Rey, CA 90292.

M. Templeton is with UNISYS, Santa Monica, CA 90406.

C. T. Yu is with the University of Illinois at Chicago, Chicago, IL 60680.

IEEE Log Number 8826078.

1) execution of selection clauses
2) choice of the fragmented relation and processing sites
3) semijoin application
4) data transmission
5) parallel query processing
6) result assembly
7) final processing.

The integrated algorithm works as follows. The selection clauses in the query are executed in Step 1) at each site which contains relations/fragments referenced in the selection clauses. Then, in Step 2), a heuristic is applied to choose a fragmented relation to remain fragmented and also to choose the processing sites. Profitable semijoins are identified and executed in Step 3). These reduce the size of the relations/fragments which need to be moved according to the results in Step 2). The data movement is done in Step 4). In Step 5), the query is executed in parallel at each processing site. This includes the execution of join clauses, the retrieval of target attributes, and possibly the preprocessing of aggregates. The results at each processing site are transmitted to the *result site* (which is the site where the query originated) in Step 6). Finally, in Step 7), the final processing is done, which includes final processing of aggregates, elimination of duplicate tuples in the answer relation, and formatting of the answer relation for output printing.

We briefly discuss the possible advantages of the initial execution of selection clauses in the following.

1) The selection clauses, especially the "=" selections, tend to select just a small number of tuples from a relation, thus greatly reducing the size of a relation and reducing the cost to transmit this relation if it has to be transmitted.

2) When one selection attribute is the *fragment attribute* (defined as the attribute(s) by which a relation is fragmented) of a fragmented relation, we can discover this semantic information by initial execution of the selection clause. This may save some unnecessary transmission and processing cost.

Example: Consider the following query:

```
retrieve  PORT.Name
where     SHIP.Fleet = "2" and
          SHIP.Base = PORT.Id.
```

If the relation SHIP is fragmented by the attribute Fleet, and the relation PORT resides in its entirety at the same site as the fragment of SHIP that has "2" as the value of the Fleet attribute, then this query can be processed locally. If this semantic information is not used, some transmission and processing time will be wasted because we have to choose the processing sites, replicate PORT at each processing site, process the query at each processing site, and combine the results from the processing sites to generate the answer. We call this semantic information a *select-fragment dependency*, i.e., from the selection clause, the single useful fragment can be determined.

3) Another kind of select-fragment dependency is possible. Consider the following query:

```
retrieve  PORT.Name
where     SHIP.Id = "2001" and
          SHIP.Base = PORT.Id.
```

If SHIP.Id functionally determines [21] SHIP.Fleet (e.g., Id is a key in SHIP) and SHIP is fragmented by the attribute Fleet, then SHIP is also fragmented by the attribute Id. Moreover, if PORT resides in its entirety at the same site as the fragment of SHIP which contains a tuple of SHIP with the Id attribute "2001," then again this query can be processed locally. For the same reason as above, we can save some unnecessary transmission and processing time if we do the initial execution of the selection clause and discover this semantic information.

III. Choosing the Fragmented Relation and the Processing Sites

In this section, we develop two heuristics for selecting a fragmented relation to remain fragmented and the associated processing sites. The first heuristic simplifies the problem by ignoring possibly different processing and transmission speeds in a network, while the second takes these speeds into account. We assume that there exists at least one fragmented relation, and that two fragments of the same fragmented relation, once placed at the same site, are unioned to form a single fragment. When there is no fragmented relation, we can apply the relation partitioning techniques as described in [26] to create one.

A. A Simplified Heuristic

We define some functions and notations for the first heuristic before proceeding. (In Section III-B., these definitions will be adopted with possible modifications for the second heuristic which allows variable processing/transmission speeds.)

$\pi(J)$: A function of time for processing joins with the joining relations/fragments in set J; we assume that π is proportional to the total size of the data in J.

$t(M)$: A function of time for transmitting M units of data from a site to another site; we assume that $t(M) = \Sigma_i t(M_i)$ where $M = \Sigma_i M_i$.

F_{jx}: A fragment of the relation R_x, which resides at site j; when R_x resides in its entirety at site j, it is denoted R_{jx}.

$|F_{jx}|$: Size of F_{jx}; $|R_x|$: size of R_x.

$S_p(f)$: A set of processing sites, with R_f the chosen fragmented relation; $S_{po}(f)$ is the optimal one among all possible $S_p(f)$.

$R(Q)$: The set of relations referenced by the query Q.

$S(Q)$: The set of sites which contain a relation/fragment referenced by Q.

$R(j)$: The set of relations/fragments contained in site j.

$S(x)$: The set of sites which contain a fragment of the fragmented relation R_x. The notations $S(Q)$, $R(j)$, and $S(x)$ are all defined under the initial distribution of the data referenced in Q.

Given a $S_p(f)$, the response time for processing joins

in Q is $T(S_p(f))$ = sum of the transmission time TT and the processing time, PT.

TT consists of the time to transmit those fragments of R_f, which do not reside at a site in $S_p(f)$ to a site in $S_p(f)$, and the time to replicate at each site in $S_p(f)$ all relations in $R(Q)$ except R_f.

PT = $\text{MAX}_{j \in S_p(f)} \pi(T_f \cup \{F_{jf}\})$ where $T_f = R(Q) - \{R_f\}$. Note that the parallel effect among all processing sites is considered by the function MAX.

In the following, we present the heuristic to decide the fragmented relation R_f to remain fragmented, the set of processing sites $S_p o(f)$, and the way to move the fragments of R_f which do not reside at a site in $S_p o(f)$ to minimize $T(S_p(f))$.

Since the local processing function π depends on many factors such as the type, number and structure of the join clauses in Q, the database content and structure, the local query optimizer, and the system load and memory utilization of the computer system, it is extremely difficult to estimate. We try to derive a simplified heuristic which avoids calculating the function π while achieving reasonable performance. Also, our testbed results [5], [19] indicate that local processing time dominates data transmission time in Mermaid's environment. Our strategy of query optimization is, therefore, to optimize processing time first and then transmission time. (The transmission time can be further reduced by semijoin applications, as will be discussed in the next section.)

1) Deciding the Fragmented Relation: We discuss choosing R_f based on the minimization of the local processing time as follows. Let R_g be a fragmented relation, and F_{mg} the largest fragment of R_g among those residing at a site in $S(g)$. From the formula for PT, we have the minimal PT, which is $\pi(T_g \cup \{F_{mg}\})$ among all possible $S_p(g)$. Suppose FR is the set of all fragmented relations. Then the fragmented relation R_g with the minimum $\pi(T_g \cup \{F_{mg}\})$ where $R_g \in \text{FR}$, will be selected as R_f. Note that minimum $\pi(T_g \cup \{F_{mg}\})$ implies minimum total size of the data in $T_g \cup \{F_{mg}\}$, and also $T_g = R(Q) - \{R_g\}$.

Denote $|T_g|$ as the total size of the relations in T_g.

$$|T_g| = \sum_{x \neq g, x \in R(Q)} |R_x| \left(\text{when } |R_x| \text{ is fragmented, } |R_x| = \sum_{j \in S(x)} |F_{jx}| \right).$$

The fragmented relation which satisfies the following condition will be selected as R_f:

$$\underset{R_g \in FR}{\text{MIN}} \left(|T_g| + \underset{j \in S(g)}{\text{MAX}} |F_{jg}| \right)$$

2) Deciding the Processing Sites and Fragment Movements: In this section, we decide $S_{po}(f)$ given the fragmented relation R_f, and the movement of the fragments of R_f to minimize $T(S_p(f))$. Let $S_{po}(f)$ be $S(f)$ initially. We derive a greedy algorithm to adjust $S_{po}(f)$ to improve $T(S_p(f))$ as follows.

Based on the fact that the minimum PT = $\pi(T_f \cup \{F_{mf}\})$, we require that the size of any unioned fragment not exceed $|F_{mf}|$. We want to move fragments of R_f around such that no unioned fragments are greater in size than F_{mf} (therefore, F_{mf} can only be moved to a site which does not contain a fragment of R_f), and such that the data transmission time is minimized. The data transmission time includes fragment movement and the associated relation replication for query processing. $S_{po}(f)$ can be decided accordingly.

We define some functions for the discussion.

$$W_i = \sum_{x \neq f, x \in R(i)} |P_{ix}| \text{ where } P_{ix} = F_{ix} \text{ or } R_{ix},\ i \in S(Q)$$

$$M_i = |T_f| - W_i,\ i \in S(Q)$$

$$B_i = M_i - |F_{if}|,\ i \in S(f).$$

W_i is the *weight* of each site, defined on the total size of the data except R_f contained in site i. $|T_f|$, as defined in the previous subsection, is the total size of the data in $R(Q) - \{R_f\}$. Therefore, $t(M_i)$ is the *cost* needed for the replication at site i, and $t(B_i)$ is the *benefit* of moving F_{if} out of site i (no replication cost is needed at site i when F_{if} is moved). Initially, TT = $\Sigma_{i \in S(f)} t(M_i)$. When $B_i \leq 0$, it is obviously unprofitable to move F_{if} out of site i [and site i will be designated one of the sites in the final $S_{po}(f)$]. When $B_i > 0$, we try to move F_{if} out of site i. Denote the set of sites with the associated $B > 0$ as Ω. We can move fragments from sites in Ω to other sites, in the following two ways.

1) F_{if} is moved to a site j in the current $S_{po}(f)$. B_i units of transmission time are reduced from the current TT, and, therefore, this movement is profitable.

2) F_{if} is moved to a site j in $S(Q) - S_{po}(f)$. In this case, an extra replication cost at site j need be paid for query processing. The cost for this replication is $t(M_j)$. Therefore, only when $B_i > M_j$ can this movement be profitable. The profit of this movement is $t(B_i - M_j)$.

The complete heuristic works as follows.

Select-and-Move:

1) Calculate $|T_g|$ for each R_g in FR and select R_f which satisfies the condition stated in Section III-A1),

2) Calculate W_i, M_i for each i in $S(Q)$, and B_i for each i in $S(f)$,

3) Designate the set of i with $B_i > 0$ as Ω (it is the set of sites from which we try to move the fragments) and designate $S_{po}(f)$ as $S(f)$,

4) Choose j from Ω where B_j is the largest; with F_{mf} denoting the largest fragment of R_f, try to select a k from $S_{po}(f) - \{j\}$ such that $|F_{jf}| + |F_{kf}| \leq |F_{mf}|$; if such a site is found, move F_{jf} to k, then update B_k, update Ω (when B_k changes from > 0 to ≤ 0), and update $S_{po}(f)$ to $S_{po}(f) - \{j\}$; else if $B_j > M_k$, $k \in S(Q) - S_{po}(f)$, move F_{jf} to site k [with M_k the smallest among all possible k in $S(Q) - S_{po}(f)$] then calculate B_k, update Ω (when $B_k > 0$) and update $S_{po}(f)$ to $S_{po}(f) - \{j\} \cup \{k\}$, and

5) Delete j from Ω; if Ω is not empty then go to 4), else stop.

Proposition 1: When F_{if} is moved from site i to site j, $B_j' < B_i$ where B_j' is the resultant B at site j.

Proof: We prove this proposition by considering the following two cases.

1) Site j is in $S_{po}(f)$. By the definition of B_j, $B_j = M_j - |F_{jf}|$. When F_{if} is moved into site j, F_{if} and F_{jf} will be unioned together and form a larger F_{jf}. However, M_j remains the same. Therefore, $B_j' < B_j$. Moreover, by the heuristic, $B_j \leq B_i$. We conclude that $B_j' < B_i$.

2) Site j is in $S(Q) - S_po(f)$. By the heuristic, $B_i > M_j$. If $B_j' \leq 0$, then obviously $B_j' < B_i$. If $B_j' > 0$ then by the definition of B_j', $M_j > B_j'$. Therefore, $B_j' < B_i$. Q.E.D.

Proposition 2: Once F_{if} is moved out of site i, it is impossible to move some F_{jf} into site i.

Proof: Assume that after F_{if} is moved out, F_{jf} can be moved from site j to site i. That is, $B_j > M_i$. By the definition of B_i, $M_i > B_i$. Moreover, by the heuristic and the Proposition 1, $B_i \geq B_j$. Therefore, $M_i > B_j$, which contradicts the assumption. Q.E.D.

Proposition 3: If there exists an F_{if} which cannot be moved by the heuristic, then for the subsequent F's, it is not necessary to check the sites in $S(Q) - S_{po}(f)$ for possible places to move them in.

Proof: Since F_{if} cannot be moved, $B_i \leq M_j$ for each j in $S(Q) - S_{po}(f)$. The subsequent F's have B's $\leq B_i$, therefore $\leq M_j$. Site j cannot be a site to accept the subsequent F's. Q.E.D.

Proposition 4: A site in the network which is not in $S(Q)$ cannot be a site to accept any fragment.

Proof: If a site j is in the network, but not in $S(Q)$, then $M_j = |T_f|$. For a fragment F_{if} at site i, $B_i = M_i - |F_{if}|$, and $B_i < M_i$. Since $M_i \leq M_j$ (by the definition of M) $B_i < M_j$, i.e., F_{if} cannot be moved to site j. Q.E.D.

3) An Example: Let Table I represent the initial data distribution for processing joins in query Q. The size of each relation/fragment is specified accordingly.

1) The fragmented relation is chosen as follows:

$$|T_1| = |R_2| + |R_3| + |R_4|$$
$$= 170 + 50 + 210 = 430$$
$$|T_2| = |R_1| + |R_3| + |R_4|$$
$$= 140 + 50 + 210 = 400$$
$$|T_1| + |F_{41}| = 430 + 80 = 510$$
$$|T_2| + |F_{62}| = 400 + 100 = 500 < |T_1| + |F_{41}|.$$

Therefore, R_2 is selected as the fragmented relation.

2) The processing sites are selected as follows. First of all, we calculate W_i, M_i for each i in $S(Q)$ ($\{1, 2, 3, 4, 5, 6\}$) and B_i for each i in $S(2)$ ($\{2, 3, 5, 6\}$). The results are listed in Table II. From the above two tables, we can see that initially $\Omega = S_po(2) = \{2, 3, 5, 6\}$.

- Since B_6 is the greatest, we try to move F_{62} first. F_{62} is the largest fragment of R_2. Therefore, we cannot move it to any other site containing a fragment of R_2, i.e., S_2, S_3, or S_5 because the resulting union would increase the size of the largest fragment. This leaves S_1 and S_4 as candidates to accept F_{62}. We choose S_1 because the replication cost M is smaller there.
- Calculate B_1, which is equal to 0. Update $S_po(2)$ to $\{1, 2, 3, 5\}$ and Ω to $\{2, 3, 5\}$.
- Now, F_{22} is the fragment to move. S_1 cannot be chosen from the candidate sites in $S_po(2)$ because it contains the largest fragment, F_{12}. This leaves S_3 and S_5, from which S_5 is arbitrarily chosen.
- Update B_5 to 90, Ω to $\{3, 5\}$, and $S_po(2)$ to $\{1, 3, 5\}$.
- Move F_{32} to site 5. Update B_5 to 70, Ω to $\{5\}$, and $S_po(2)$ to $\{1, 5\}$.
- F_{52} cannot be moved to S_1 since F_{12} is the largest fragment. This exhausts the candidate processing sites in $S_po(2)$. Therefore, we compare the costs and benefits at the other sites. F_{52} cannot be moved to S_2, S_3, or S_6 by Proposition 2, and it cannot be moved to S_4 because $B_5 < M_4$. Thus, we stop here.

The fragmented relation is R_2. The processing sites are S_1 and S_5. The data transmissions for the replication at S_1 are move F_{31} and F_{41} to S_1; for the replication at S_5 are move F_{11}, F_{31}, and F_{41} to S_5. The other data transmissions for moving fragments of R_2 are move F_{62} to S_1, move F_{22} to S_5, and move F_{32} to S_5.

TABLE I

	R_1	R_2	R_3	R_4
S_1	F_{11}, 40		R_3, 50	R_4, 210
S_2		F_{22}, 10		R_4, 210
S_3	F_{31}, 20	F_{32}, 20	R_3, 50	R_4, 210
S_4	F_{41}, 80		R_3, 50	
S_5		F_{52}, 40	R_3, 50	R_4, 210
S_6		F_{62}, 100		

TABLE II

	S_1	S_2	S_3	S_4	S_5	S_6
W	300	210	280	130	260	0
M	100	190	120	270	140	400
B		180	100		100	300

B. Considering Variable Processing/Transmission Speeds

Since Mermaid could be applied to an environment in which heterogeneous computer systems are connected through different links in a network, we have to consider variable processing speeds and variable transmission speeds in the heuristic. In this subsection, we derive another heuristic for select-and-move, which assumes that processing speeds at each site and transmission speeds at each link are variable.

Denote $\pi_i(J)$ as the processing time at site i, for processing joins with the joining relations/fragments in set J, $t_{ij}(M)$ as the minimum transmission time for transmitting M units of data from site i to site j, and $t(M_i)$ as the minimum transmission time needed to replicate at a processing site i all relations referenced by a query Q except the fragmented relation. (We must consider minimum times here because for any replication, there may be mul-

tiple copies of relations at different sites and multiple paths through the network.) The definitions of F_{if}, $|F_{if}|$, $S_p(f)$, $S_{po}(f)$, $S(f)$, and $T(S_p(f))$ for a fragmented relation R_f are the same as before. However, the definitions of TT, the transmission time, and PT, the processing time have to be modified, namely, the replication time in TT should be the minimum one [i.e., $\Sigma_{i \in S_p(f)} t(M_i)$] and the π in PT should be replaced by π_j to reflect the consideration of variable processing and transmission speeds.

Whereas the previous heuristic selected a relation to remain fragmented and then tried moving fragments, the new heuristic tries moving fragments to decide which relation and associated processing sites to select. The new heuristic works as follows.

For each fragmented relation R_g in FR (the set of fragmented relations), we decide $S_{po}(g)$ and calculate $T(S_{po}(g))$. The relation with the minimum $T(S_{po}(g))$, say R_f, is the relation to remain fragmented; and $S_{po}(f)$ is the set of processing sites. The procedures for deciding $S_{po}(g)$ are the following.

1) Let $S_{po}(g)$ be $S(g)$ initially.

2) Calculate $T(S_{po}(g))$, which is equal to $\Sigma_{i \in S_{po}(g)} t(M_i) + \text{MAX}_{i \in S_{po}(g)} \pi_i(T_g \cup \{F_{ig}\}$ where $T_g = R(Q) - \{R_g\}$.

3) We try to move F_{ig} around to reduce $T(S_{po}(g))$:

a) For each F_{ig}, calculate P_{ij} for each site j where $i \in S_{po}(g)$, $j \in N$, the set of all sites in the network, $j \neq i$ and P_{ij} is the profit of the move of F_{ig} from site i to site j (as discussed below).

b) Let P_{xy} be the largest among all such P's.

c) If $P_{xy} > 0$ then go to a) assuming that F_{xg} has been moved into site y; otherwise an optimal sequence of moves (which could be empty) has been obtained and the final $S_{po}(g)$ has been decided.

We discuss the profit of a move for F_{ig} from site i to site j by considering the following four cases (note that profit = benfit − cost). The maximum processing time for processing joins in Q among all processing sites is denoted TIME1, and the next to the minimum one is denoted TIME2 for the discussion.

- Site i generates TIME1, and site j is in $S_{po}(g)$.

The benefit = $t(M_i)$ + [TIME1 − MAX ($\pi_j(T_g \cup \{F_{jg}\} \cup \{F_{ig}\})$, TIME 2)] where $\pi_j(\cdots)$ is the new processing time at site j.

The cost = $t_{ij}(|F_{ig}|)$.

- Site i generates TIME1, and site j is not in $S_{po}(g)$.

The benefit = $t(M_i)$ + [TIME1 − MAX ($\pi_j(T_g \cup \{F_{ig}\})$, TIME2)].

The cost = $t_{ij}(|F_{ig}|) + t(M_j)$.

- Site i does not generate TIME1, and site j is in $S_{po}(g)$.

The benefit = $t(M_i)$ + [TIME1 − MAX ($\pi_j(T_g \cup \{F_{jg}\} \cup \{F_{ig}\})$, TIME1)].

The cost = $t_{ij}(|F_{ig}|)$.

- Site i does not generate TIME1, and site j is not in $S_{po}(g)$.

The benefit = $t(M_i)$ + [TIME1 − MAX ($\pi_j(T_g \cup \{F_{ig}\})$, TIME1)].

The cost = $t_{ij}(|F_{ig}|) + t(M_j)$.

The benefit is the saving of the replication time at site i plus possible reduction of the processing time. (Actually, it is impossible to reduce the processing time if site i does not generate TIME1.) The cost is the transmission time for F_{ig} plus possible replication time at site j.

IV. Semijoin Application

Once we have selected the fragmented relation and the processing sites, we have also selected all the necessary transmissions, which include the fragment movement of the fragmented relation and the relation replication at each processing site. For those relations/fragments which have to be moved, we try to apply semijoins to reduce their size before their move. For easy discussion, again we ignore the fact that there may be different processing/transmission speeds in a network. The procedure for applying semijoins is as follows.

Reduce-before-Move:

1) List all the possible semijoins incurred in query Q (Note that for a join $R_x \leftrightarrow R_y$, if R_y is fragmented then each fragment of it will be treated individually and associated with a semijoin $R_x \rightarrow F_{iy}$).

2) Cross out those semijoins which are applied to some relations/fragments not to be moved according to the select-and-move heuristic.

3) Select the most profitable semijoins one by one according to the following cost functions until no profitable semijoin exists.

The profit of a semijoin S = the benefit of S − the cost of S.

For a semijoin $R_x \overset{a}{\rightarrow} R_y$ (where a denotes the joining attribute between R_x and R_y) we discuss its profitability based on the fragmentation of R_y as follows.

1) R_y is an unfragmented relation. Denote $S(y)$ as the set of sites which contain a copy of R_y. If R_y has not yet been reduced by some semijoin, then we choose a site j in $S(y)$ to be the site to process this semijoin where j contains either a copy of the unfragmented relation R_x or the largest fragment of R_x among those in $S(y)$ if R_x is fragmented. If we cannot find such j, then let j be any site in $S(y)$. If R_y has been reduced, then let j be the site which contains the reduced copy of R_y (which is designated the *primary copy* of R_y).

The benefit of the semijoin = $n \times t(|R_y| - |R'_y|)$ where $n \geq 1$, representing the number of transmissions needed for R_y, $|R'_y|$ is the resultant size of R_y after this semijoin is applied ($|R'_y|$ can be derived using a formula in [25]).

We denote $\pi(\Delta)$ as the processing time needed for processing the operation Δ here.

The cost of this semijoin =

- π(projection of attribute a from R_x) + $t(|R_x.a|)$ + $\pi(R_x.a \leftrightarrow R_y)$ when R_x is unfragmented and does not reside at site j;
- $\pi(R_x \rightarrow R_y)$ when R_x is unfragmented and resides at

site j; (notice that if R_x has been reduced, only the primary copy of R_x applies in the above two cases).

- $\text{MAX}_{i \in S(x)}\ \pi$(projection of attribute a from F_{ix}) + $\Sigma_{i \in S(x), i \neq j}\ t(|F_{ix}.a|)$ + $\pi(R_x.a \leftrightarrow R_y)$ when R_x is fragmented. Note that the projection can be done in parallel, which has been considered by the function MAX.

2) R_y is a fragmented relation. In this case, we have $|S(y)|$ semijoins incurred where $|S(y)|$ denotes the number of the sites which contain a fragment of R_y. We treat each semijoin independently for reducing the size of each fragment of R_y. The same analysis as in case 1) holds here except that there is only one site which contains a fragment of R_y. No selection of the semijoin processing site is needed.

The improvement techniques for semijoin applications as described in [10], [25] may be further used to reduce the cost for query processing.

Example: For the example in Section III-A3), we assume that the query Q contains the following three join clauses:

$$R_1 \leftrightarrow R_3,\ R_2 \leftrightarrow R_4, \text{ and } R_3 \leftrightarrow R_4.$$

All possible semijoins are listed as follows:

$$R_1 \rightarrow R_3,\ R_3 \rightarrow F_{11},\ R_3 \rightarrow F_{31},\ R_3 \rightarrow F_{41},$$

$$R_2 \rightarrow R_4,\ R_4 \rightarrow F_{22},\ R_4 \rightarrow F_{32},\ R_4 \rightarrow F_{52},$$

$$R_4 \rightarrow F_{62},\ R_3 \rightarrow R_4,\ R_4 \rightarrow R_3.$$

Among them, $R_1 \rightarrow R_3$, $R_2 \rightarrow R_4$, $R_4 \rightarrow F_{52}$, $R_3 \rightarrow R_4$, and $R_4 \rightarrow R_3$ will be crossed out since R_3, R_4, and F_{52} are not to be moved according to the heuristic select-and-move.

For the semijoin $R_3 \rightarrow F_{31}$, the semijoin processing site is S_3, the benefit $= 2 \times t(|F_{31}| - |F'_{31}|)$ [F_{31} needs to be moved to S_1 and S_5] and the cost $= \pi(R_3 \rightarrow F_{31})$.

V. Distributed Aggregate Processing

In this section, we consider aggregate processing based on the following assumptions:

- aggregates appear in the target of a query only,
- aggregates are of three forms: agg($R_x.a$), agg($R_x.a$ by b) and $\text{agg}_1(\text{agg}_2(R_x.a \text{ by } b))$,
- when there is more than one aggregate in the target, the aggregates have to operate on the same relation,
- agg($R_x.a$ by b) can be in the target with some attribute, say $R_y.o$, under the restriction that the values in $R_x.b$ and $R_y.o$ have a one-to-one correspondence, and
- the operators supported for the aggregation are MAX, MIN, SUM, AVERAGE, and COUNT; the last three operators can be specified with the operator UNIQUE.

Recalling that parallel query processing is performed in Step 5) of the integrated algorithm and final processing is done in Step 7), aggregates may be processed in three ways: a) preprocessing locally at each processing site both before and after joins are executed in Step 5), and final processing globally at the result site in Step 7), b) preprocessing locally after joins are executed in Step 5), and final processing globally in Step 7), and c) global processing in Step 7) only. Preprocessing of aggregates has two advantages, i.e., it can reduce the size of relations and reduce the response time for aggregate processing by parallelism. However, to process aggregates at too many places (as in method a), we process aggregates before join execution, after join execution, and at the final processing) complicates the problem and may not be cost beneficial. Our policy on aggregate processing is, therefore, to apply method b) when possible, or c) when b) is not applicable.

To process aggregates by method b), the aggregate operators have to be modified for local and global processing. Table III specifies the operation to be performed at the local sites L(agg) and the operation to be performed globally at the result site G(agg).

For example, if the aggregate operator is AVERAGE, at each local site we compute two quantities, namely, the SUM and the COUNT. Then, at the result site, we obtain the SUM of the sums computed at the local sites divided by the SUM of the counts.

In an agg contains a UNIQUE operator which can be processed by method b), then L(agg) contains UNIQUE, but G(agg) does not. We shall illustrate this by an example later.

After joins are processed in Step 5) of the integrated algorithm, the joined relation is actually fragmented at each processing site by the fragment attribute of the fragmented relation we chose before. Denote the joined relation R_j, and the fragment attribute $R_j.f$. For an aggregate agg($R_x.a$ by b), $R_x.a$ is called an *aggregate attribute*, and $R_x.b$ a *group-by attribute*.

A. Processing of Aggregates Yielding a Scalar

Agg($R_x.c$) and $\text{agg}_1(\text{agg}_2(R_x.a \text{ by } b))$ are the two aggregates which yield a scalar. If the following two conditions [one for agg($R_x.c$) and one for $\text{agg}_1(\text{agg}_2(R_x.a$ by b))] are satisfied, then we can process these aggregates by method b).

1) agg does not contain UNIQUE, or $R_x.c \Rightarrow R_j.f$, and
2) agg_1 does not contain UNIQUE, and $R_x.b \Rightarrow R_j.f$.

This is because when $R_x.k \Rightarrow R_j.f$, $R_x.k$ is also a fragment attribute of R_j such that distributed aggregate processing is possible.

The query has to be decomposed as follows. The target of the local queries contains $L(\text{agg}(R_x.c))$ and $L(\text{agg}_1(\text{agg}_2(R_x.a \text{ by } b)))$, and the target of the global query contains $G(\text{agg}(R_j.c))$ and $G(\text{agg}_1(R_j.a))$. There is a qualification (which is the same as the one in the original query) associated with the target in the local queries while no qualification is in the global query since all selection and join clauses have already been processed when the global query is about to be executed.

We can check if $R_x.k \Rightarrow R_j.f$ by checking if a) $R_x.k$ is $R_j.f$, i.e., R_x is the fragmented relation and $R_x.k$ is the fragment attribute, b) R_x is R_j, $R_x.k$ is a key, or c) R_x is not R_j, $R_x.k$ is a foreign key of R_j.

TABLE III

agg	L(agg)	G(agg)
MAX	MAX	MAX
MIN	MIN	MIN
COUNT	COUNT	SUM
SUM	SUM	SUM
AVERAGE	SUM,COUNT	SUM(sums)/SUM(counts)

Example: The content of the joined relation over relations EMP(E#, D#, Rank, Sal) and DEPT(D#, Dname, College) is as follows. It is fragmented by the attribute D# at site 1 and site 2.

E#	D#	Rank	Sal	D#	Dname	College
1	1	P	50K	1	EECS	ENG
2	1	AP	45K	1	EECS	ENG
3	1	P	40K	1	EECS	ENG
4	1	AsP	35K	1	EECS	ENG
5	2	P	40K	2	ME	ENG
6	2	AsP	35K	2	ME	ENG
7	3	P	35K	3	CHE	ENG
8	3	AsP	34K	3	CHE	ENG
9	4	P	32K	4	CIE	ENG
10	4	AP	30K	4	CIE	ENG

SITE 1

E#	D#	Rank	Sal	D#	Dname	College
11	5	P	35K	5	ISE	ENG
12	5	AsP	34K	5	ISE	ENG
13	6	P	32K	6	BIOE	ENG
14	6	AP	30K	6	BIOE	ENG

SITE 2

Q_1 = {MAX(EMP.Sal), MIN(MAX(EMP.Sal by D#))|DEPT.D# = EMP.D# and DEPT.College = "ENG"}.

This query can be processed by method b). It is decomposed into two local queries that are of the same form as the original query. At site 1, we generate {(50K, 32K)} as the result, while at site 2 it is {(35K, 32K)}. These results are transmitted to the result site and the global query {MAX(Sal), MIN(Sal)} is performed to generate the answer, which is {(50K, 32K)}.

When the conditions above are satisfied, we say that the aggregates (or the query) can be *completely processed.* There is another situation where although the aggregate $\text{agg}(R_x.c)$ can be completely processed, the aggregate $\text{agg}_1(\text{agg}_2(R_x.a \text{ by } b))$ cannot. We can *partially process* these aggregates by using a group-by operator. Partial processing of aggregates has to satisfy the following two conditions:

1) agg does not contain UNIQUE, or ($R_x.c \Rightarrow R_j.f$ and $R_x.c \Rightarrow R_x.b$), and

2) agg_2 does not contain UNIQUE, or $R_x.a \Rightarrow R_j.f$, or $R_x.b \Rightarrow R_j.f$.

The target of each local query contains $L(\text{agg}(R_x.c \text{ by } b))$, $L(\text{agg}_2(R_x.a \text{ by } b))$, and $R_x.b$ which will be used for references in the final processing (when $R_x.b \Rightarrow R_j.f$, $R_x.b$ need not be included). The target of the global query contains $G(\text{agg}(R_j.c))$ and $\text{agg}_1(G(\text{agg}_2(R_j.a \text{ by } b)))$, $(G(\text{agg}(R_j.c)))$ and $\text{agg}_1(R_j.a)$ when $R_x.b \Rightarrow R_j.f$).

Example:

$$Q_2 = \{\text{MAX(EMP.Sal)}, \text{MIN(MAX(EMP.Sal by Rank))} \mid (\cdots)\}.$$

Since Rank does not functionally determine D#, this query cannot be completely processed. However, it can be partially processed as follows.

We decompose this query into two local queries, which are of the same form {MAX(EMP.Sal by Rank), MAX(EMP.Sal by Rank), Rank|(···)}. These two local queries will generate {(50K, 50K, P), (45K, 45K, AP), (35K, 35K, AsP)} at site 1 and {(35K, 35K, P), (30K, 30K, AP), (34K, 34K, AsP)} at site 2. The global query is {MAX(Sal), MIN(MAX(Sal by Rank))}, which will generate the answer {(50K, 35K)}.

$$Q_3 = \{\text{AVERAGE UNIQUE(EMP.Sal)} \mid (\cdots)\}.$$

This query cannot be processed by method b) since attribute Sal does not functionally determine attribute D#. It has to be globally processed at the result site.

B. Processing of Aggregates Yielding a Relation

$\text{Agg}(R_x.a \text{ by } b)$ is the aggregate which yields a relation. When $R_x.b \Rightarrow R_j.f$, we say that the aggregate can be *completely processed.* If this is not the case, then if agg does not contain UNIQUE, or $R_x.a \Rightarrow R_j.f$, then the aggregate can be *partially processed.* In both situations, we can preprocess aggregates locally.

For the aggregates which can be completely processed, the local queries will be of the same form as the original query. There is no associated global query. The answer of the query is just the union of the results produced by the execution of the local queries. For the aggregates which can be partially processed, the target of the local queries contains $L(\text{agg}(R_x.a \text{ by } b))$, $R_x.b$, and possibly attribute $R_y.o$ which is in the target of the original query. The target of the global query contains $(G(\text{agg}(R_j.a \text{ by } b))$ and possibly attribute $R_j.o$.

Example:

$$Q_4 = \{\text{SUM(EMP.Sal by D\#), DEPT. Dname} \mid (\cdots)\}.$$

Since D# ⇒ D#, this query can be completely processed. The local queries will generate {(170K, EECS), (75K, ME), (69K, CHE), and (62K, CIE)} and {(69K, ISE), (62K, BIOE)} at site 1 and site 2, respectively. The answer of this query is the union of these two sets.

$$Q_5 = \{\text{SUM(EMP.Sal by Rank), EMP.Rank} \mid (\cdots)\}.$$

This query can be partially processed. The local queries are of the same form {SUM(EMP.Sal by Rank), EMP.Rank, EMP.Rank|(· · ·)}, which will generate {(197K, P, P), (75K, AP, AP), (104K, AsP, AsP)} at site 1 and {(67K, P, P), (30K, AP, AP), (34K, AsP, AsP)} at site 2. The global query will generate the answer {(264K, P), (105K, AP), (138K, AsP)}.

$$Q_6 = \{\text{COUNT UNIQUE(EMP.D\# by Sal)},\ \text{EMP.Sal}|(\cdots)\}.$$

Since D# ⇒ D#, this query can be partially processed. The local queries are of the form {COUNT UNIQUE(EMP.D# by Sal), EMP.Sal, EMP.Sal|(· · ·)}, and will generate {(1, 50K, 50K), (1, 45K, 45K), (2, 40K, 40K), (3, 35K, 35K), (1, 34K, 34K), (1, 32K, 32K), (1, 30K, 30K)} at site 1 and {(1, 35K, 35K), (1, 34K, 34K), (1, 32K, 32K), (1, 30K, 30K)} at site 2. The global query is {SUM(EMP.D# by Sal), Sal} which produces the answer {(1, 50K), (1, 45K), (2, 40K), (4, 35K), (2, 34K), (2, 32K), (2, 30K)}.

Note that for processing Q_6, the global aggregate SUM does not contain UNIQUE. This is because D# is the fragment attribute. Once its value is unique locally, it must also be unique globally. Also, in the local queries of Q_5 and Q_6, there are redundant attributes in the target. We may keep only one of these attributes in these cases.

VI. Conclusions

In this paper, we present a distributed query optimization algorithm which integrates the features of semijoin and replicate query processing strategies.

The major component of this algorithm is the select-and-move heuristics which choose a fragmented relation to remain fragmented and the associated processing sites. One policy for choosing them is based on optimizing the processing cost first and then the transmission cost. The other policy considers a dynamic network environment where processing and transmission speeds can be variable. This policy is especially valuable if query processing is done by adaptive techniques [29] which detect the current status of sites/links and adjust the processing/transmission speeds accordingly. Distributed aggregate processing is based on the fact that after the join clauses are processed at each processing site, we have a joined relation which is fragmented by the fragment attribute of the fragmented relation we chose by select-and-move. We can, therefore, check the functional dependency among the fragment attribute, the aggregate attribute, and the group-by attribute to possibly preprocess aggregates in parallel.

Semantic information can be used for efficient query processing. The semantic query optimization approach, as proposed in [13], [15], could be adopted as an enhancement to the integrated algorithm.

References

[1] P. Apers, A. Hevner, and S. B. Yao, "Optimization algorithm for distributed queries," *IEEE Trans. Software Eng.*, vol. SE-9, Jan. 1983.

[2] P. Bernstein and D. M. Chiu, "Using semi-joins to solve relational queries," *JACM*, Jan. 1981.

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

[4] P. Black and W. Luk, "A new heuristic for generating semi-join programs for distributed query processing," presented at IEEE COMPSAC, 1982.

[5] D. Brill, M. Templeton, and C. T. Yu, "Distributed query processing strategies in Mermaid, a frontend to data management systems," presented at IEEE Data Eng. Conf., 1984.

[6] D. M. Chiu, P. Bernstein, and Y. C. Ho, "Optimizing chain queries in a distributed database system," *SIAM J. Comput.*, Feb. 1984.

[7] D. M. Chiu and Y. C. Ho, "A method for interpreting tree queries into optimal semi-join expressions," presented at ACM SIGMOD, 1980.

[8] J. M. Chang, "A heuristic approach to distributed query processing," *Proc. VLDB*, 1982.

[9] A. L. P. Chen and V. O. K. Li, "Optimizing star queries in a distributed database system," *Proc. VLDB*, 1984.

[10] ——, "Improvement algorithms for semijoin query processing programs in distributed database systems," *IEEE Trans. Comput.*, Nov. 1984.

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

[12] R. Epstein, M. Stonebraker, and E. Wong, "Distributed query processing in a relational database system," presented at ACM SIGMOD, 1978.

[13] M. Hammer and S. Zdonik, "Knowledge-based query processing," *Proc. VLDB*, 1980.

[14] M. Jarke and J. Koch, "Query optimization in database systems," *ACM Comput. Surveys*, June 1984.

[15] J. King, *Query Optimization by Semantic Reasoning*. UMI Research Press, 1984.

[16] W. Luk and L. Luk, "Optimizing semi-join programs for distributed query processing," presented at Int. Conf. Databases, 1983.

[17] G. Lohman, C. Mohan, L. Hass, B. Lindsay, P. Selinger and P. Wilms, "Query processing in R*," IBM Intern. Rep., 1984.

[18] R. MacGregor, "ARIEL—A semantic frontend to relational DBMS's," *Proc. VLDB*, 1985.

[19] M. Templeton, D. Brill, A. L. P. Chen, S. Dao, and E. Lund, "Mermaid—Experiences with network operation," presented at IEEE Data Eng. Conf., 1986.

[20] M. Templeton, D. Brill, A. L. P. Chen, S. Dao, E. Lund, R. MacGregor, and P. Ward, "Mermaid—A front-end to distributed heterogeneous databases," *Proc. IEEE*, May 1987.

[21] J. D. Ullman, *Principles of Database Systems*. Rockville, MD: Computer Science, 1982.

[22] R. Williams, *et al.*, "R*: An overview of the architecture," presented at Int. Conf. Databases, 1982.

[23] E. Wong, "Retrieving dispersed data from SDD-1: A system for distributed databases," presented at Berkeley Workshop Distrib. Data Manage. Comput. Networks, 1977.

[24] C. T. Yu, C. C. Chang, M. Templeton, D. Brill, and E. Lund, "On the design of a query processing strategy in a distributed database environment," presented at ACM SIGMOD, 1983.

[25] ——, "Query processing in a fragmented relational distributed system: MERMAID," *IEEE Trans. Software Eng.*, vol. SE-11, Aug. 1985.

[26] C. T. Yu, K. Guh, D. Brill, and A. L. P. Chen, "Partitioning relation for parallel processing in fast local networks," presented at IEEE Int. Conf. Parallel Process., 1986.

[27] C. T. Yu, K. Guh, C. C. Chang, C. H. Chen, M. Templeton, and D. Brill, "An algorithm to process queries in a fast distributed network," presented at IEEE Real-Time Syst. Symp., 1984.

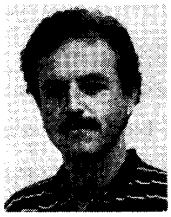
[28] C. T. Yu, K. Lam, C. C. Chang, and S. K. Chang, "A promising approach to distributed query processing," presented at Berkeley Workshop Distrib. Data Manage. Comput. Networks, 1982.

[29] C. T. Yu, L. Lilien, K. Guh, M. Templeton, D. Brill, and A. L. P. Chen, "Adaptive techniques for distributed query optimization," presented at IEEE Data Eng. Conf., 1986.

Arbee L. P. Chen (S'80–M'84) received the B.S. degree from National Chiao Tung University, Taiwan, Republic of China, in 1977, the M.S. degree from Stevens Institute of Technology, Hoboken, NJ, in 1981, both in computer science, and the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, CA, in 1984.

He is currently a Member of Technical Staff at Bell Communications Research, Piscataway, NJ, and an Adjunct Assistant Professor in the Department of Electrical Engineering and Computer Science, Polytechnic University, Brooklyn, NY, where he teaches a course on compiler and formal languages. Prior to joining Bellcore, he was a Research Scientist at System Development Corporation (now, UNISYS Corporation), Santa Monica, CA. His research interests include distributed databases, data models, computer networks, and network operations simulation modeling.

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

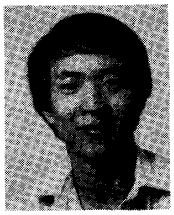


David Brill received the B.A. degree from City University of New York in 1968, and the M.A. degree in communication research from Stanford University, Stanford, CA, in 1969. He did additional graduate work at the Stanford Artificial Intelligence Project.

He worked on the ARPA Speech Project at Speech Communications Research Laboratory, Santa Barbara, CA. From 1977 to 1987, he was with System Development Corporation in Santa Monica, CA (now Unisys), where he specialized in distributed query optimization and natural language interfaces to data management systems. He is currently doing knowledge representation research at USC Information Sciences Institute, Marina del Rey, CA.

Marjorie Templeton received the B.A. degree in economics from Carleton College, Northfield, MN, in 1963.

She was the principal designer of the Mermaid system. Currently, she is Manager of the Knowledge and Data Systems Branch of the UNISYS West Coast Research Center and is Technical Area Manager for data management for the System Development Group of UNISYS. She has been with UNISYS (formerly SDC) since 1972 and has spent most of that time working on some aspect of heterogeneous access to existing databases. The Mermaid project began in 1982 with the goal of providing a common structured query language to access multiple existing databases as though they were one. She worked on a contract with the Defense Intelligence Agency to design a standard query language with translation to one of several DBMS's. Previously, she was Project Manager for EUFID which developed an English language interface to existing databases stored under either Ingres or WWDMS. She joined SDC to work on DS/3, a commercial DBMS. She was formerly employed by Planning Research, IBM, and the Federal Reserve Bank.



Clement T. Yu received the B.Sc. degree in applied mathematics from Columbia University, New York, NY, in 1970, and the Ph.D. degree in computer science from Cornell University, Ithaca, NY, in 1973.

He is currently a Professor in the Department of Electrical Engineering and Computer Science at the University of Illinois at Chicago. He has published in various journals and conference proceedings, including *Journal of the ACM*, *Communication of the ACM*, *ACM Transactions on Data Base Systems*, *ACM Computing Survey*, *Journal of Theoretical Computer Science*, *Journal of Computer & System Science*, *Information Processing & Management*, *Information Processing Letters*, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, IEEE TRANSACTION ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, IEEE TRANSACTIONS ON COMPUTERS, INFORMATION TECHNOLOGY, *Canadian Journal of Operation Research & Information Processing*, *ACM SIGMOD*, *VLDB*, *ACM SIGIR*, *IFIP*, IEEE COMPSAC, IEEE, DATA ENGINEERING, and *ASIS*. He has been serving as a consultant for various corporations. His research interests are database management and information retrieval.