Partition Strategy for Distributed Query Processing in Fast Local Networks

CLEMENT T. YU, KEH-CHANG GUH, MEMBER, IEEE, DAVID BRILL, AND ARBEE L. P. CHEN, MEMBER, IEEE

Abstract—**A partition and replicate strategy for processing distributed queries referencing no fragmented relation is sketched. An algorithm is given to determine which relation and which copy of the relation is to be partitioned into fragments, how the relation is to be partitioned and where the fragments are to be sent for processing. Simulation results show that the partition strategy is useful for processing queries in fast local network environments. The results also show that the number of partitions does not need to be large. The use of semijoins in the partition strategy is discussed. A necessary and sufficient condition for a semijoin to yield an improvement is provided.**

Index Terms—**Distributed database, fast local network, partition and replicate, query optimization, relational database.**

I. INTRODUCTION

DISTRIBUTED query processing is an important factor in the overall performance of a distributed database system. Surveys on distributed query optimization have been given in [22], [42]. Many distributed query processing algorithms [1]–[4], [6], [8]–[17], [19]–[21], [24], [26], [38]–[41], [43], [47], [49]–[51] have been proposed. Most of these algorithms assume that the data communication cost is dominant in long-haul networks where data communication costs are high and make use of semijoins to reduce the amount of data transfer. While such an assumption is reasonable for long-haul networks, it may not be valid for fast local networks. In contrast, the "fragment and replicate" query processing strategy was used in distributed Ingres [17]. The strategy requires that a relation to be fragmented and placed in a number of sites, while other relations are replicated at the sites of the fragmented relation. The user query is decomposed into the same number of subqueries and each subquery is processed at one of these sites. Its main feature is to have parallel processing at the different sites to improve response time. Several other algorithms also take advantage of fragmentation to process queries [18], [31], [34], [46], [48].

Manuscript received September 4, 1987; revised January 10, 1989.

C. T. Yu is with the Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, Chicago, IL 60680.

K. C. Guh is with the Department of Electrical Engineering and Computer Science, University of Wisconsin—Milwaukee, Milwaukee, WI 53201.

D. Brill is with Information Science Institute, University of Southern California, Marina del Rey, CA 90292.

A. L. P. Chen is with Bell Communications Research, 444 Hoes Lane, Piscataway, NJ 08854.

IEEE Log Number 8927389.

A performance analysis for the semijoin algorithm [44] and the replicate algorithm [46] was done by [7], [36]. It was found that the replicate algorithm outperforms the semijoin algorithm in a fast local network environment. Although one algorithm may not outperform the other algorithm in all local networks, it is of interest to investigate further the usefulness of the fragment and replicate algorithm. In this paper, we concentrate on the situation where a query references several relations, but none of the relations is fragmented. In order to use the fragment and replicate algorithm, it is necessary to decide the relation and its copy (if multiple copies exist) to be partitioned into fragments, how the relation is to be partitioned and where the fragments are to be sent for processing [45]. This is the subject of this paper.

Relation partitioning has been used as a strategy for data allocation [32], [39], or dynamic query processing [17], [33], [45], to achieve a high degree of parallelism among processing units and to improve response time. In [33], two relations are dynamically partitioned into fragments for natural join. Complete processing of a query referencing more than two relations may need the replication of all other referenced relations in all processors or repeated partitioning of the relations. In [17], one of the relations referenced by a query is partitioned into equal-sized fragments, each fragment is sent to a computer processor, and all other relations are replicated in all computer processors. Since different computer processors may have different processing speeds and/or different access methods for accessing required data, equal-sized fragments may not balance the load of computer processors. This is considered in [45] to partition a relation into unequal-sized fragments for load balancing. However, in [45], a particular set of cost functions is used to develop the partition strategy. In this paper, we generalize the partition strategy for more general cost functions. Sensitivity analyses are provided to show the feasibility of the strategy in fast local networks. Furthermore, a necessary and sufficient condition for a semijoin to be capable of reducing response time when used in conjunction with the partition algorithm is given.

The organization of this paper is as follows. In Section II, the partition problem is described. The general partition strategy is given in Section III. In Section IV, the complexity of the partition strategy is discussed. In Sec-

tion V, sensitivity analyses are given and discussed. We then give a necessary and sufficient condition for using a semijoin in partition strategy in Section VI. The conclusion is given in Section VII.

II. Partition Problem

Before we describe the partition problem, an example is given to illustrate how a fragment and replicate strategy achieves parallel processing and improves response time.

Example 2.1: Let a query reference two relations R_1 and R_2, which are both fragmented at sites S_1 and S_2 as shown below. F_{ij} is the fragment of R_i at site j. For simplicity, we assume that both sites have equal processing speeds.

Distribution of Relations for Example 2.1

Relation	Total Tuples	S_1	S_2
R_1	11 000	F_{11}(6000)	F_{12}(5000)
R_2	5000	F_{21}(3000)	F_{22}(2000)

If the query is processed only at site S_1, the response time is the sum of the times taken to bring the remote fragments F_{12} and F_{22} to site S_1 and to join $F_{11} \cup F_{12}$ with $F_{21} \cup F_{22}$. This is time consuming, since there is no parallel processing and there is significant data transfer. Similarly, if the query is processed only at site S_2 then the response time will also be high.

Now, if R_1 is left fragmented, then the processing can be done at both sites. Site S_1 will need to get fragment F_{22} from site S_2 and site S_2 will need to get fragment F_{21} from site S_1, so that R_2 is replicated at both sites. After the data transfer, both sites can work in parallel. Site S_1 will perform the join between F_{11} and R_2, while site S_2 will perform the join between F_{12} and R_2. The response time will be the maximum of the times spent at the two sites. The time spent at site S_1 can be shown to be smaller than that of the earlier strategy because the fragment F_{12} does not need to be transferred and processed at S_1. Similarly, the time at site S_2 can be shown to be smaller than single site processing. Thus, the response time if R_1 is left fragmented will be smaller than that in which either all processing takes place in site S_1 or all processing takes place in site S_2. □

Example 2.1 shows that fragment and replicate strategy can be used to improve response time. But if none of the relations referenced by a query is already fragmented, at least one relation should be partitioned into fragments in order to use the strategy. We use Example 2.2 to illustrate the situation.

Example 2.2: Let a query reference two relations R_1 and R_2, which are not fragmented and distributed among 3 sites as follows.

Distribution of Relations for Example 2.2

Relation	Total Tuples	S_1	S_2	S_3
R_1	11 000	R_1		
R_2	10 000		R_2	R_2

Suppose that S_3 is slightly faster than S_2 but much faster than S_1. A possible strategy without partitioning any relation is to send R_1 from S_1 to S_3 and process the query at site S_3. This strategy processes the whole relations R_1 and R_2 at site S_3. Suppose we partition R_1 into two fragments with the fragment containing 5000 tuples and that containing 6000 tuples sent to S_2 and S_3, respectively. Since only portions of R_1 are processed at S_2 and S_3, the times incurred at S_2 and S_3 are smaller than those of the earlier strategy. Processing at sites S_2 and S_3 take place in parallel. Thus, partitioning a relation for parallel processing could be a useful strategy. □

Example 2.2 shows that R_1 is partitioned into 2 fragments of unequal sizes and the fragments are sent to different sites for processing. In general, we need to determine which relation to be partitioned, how it is partitioned, what are the processing sites, etc. We describe the partition problem in the rest of this section.

Definition: A relation R_i is horizontally partitioned into d disjoint fragments (subsets of the relation) $\{F_{ij}\}$, $1 \leq j \leq d$, if $R_i = U_{1 \leq j \leq d} F_{ij}$ and $F_{ij} \cap F_{ik} = \varnothing$ if $j \neq k$.

Suppose that $\{R_1, \cdots, R_r\}$ are the relations referenced by a query Q and none of them is fragmented. One of them, say R, will be chosen to be horizontally partitioned into d disjoint fragments, for some integer d. Then, the fragments will be assigned to a set of d sites, called processing sites, with each fragment assigned to one site. The other relations, if not present at each of the d sites, will be sent to those sites, so that after the sending, each of the d sites will have a fragment of R and a copy of each of the other relations. A relation may have several copies with each copy situated at a site. If there exists a copy at a processing site, this copy is used at this site. At each processing site, a subquery, which is the same as the query Q but referencing the fragment of R that is assigned to this site and all other relations referenced by the query Q, is executed. The answer to the query Q is the union of the results to the subqueries at the d sites. The problem is to decide 1) the relation to be partitioned, i.e., R, 2) the number of fragments the relation is to be partitioned, i.e., d, 3) the size of each of the d fragments, 4) the set of sites where the fragments are to be assigned (i.e., processing sites), and 5) the copy of the partitioned relation to be used, if multiple copies exist, such that the response time (in terms of both local processing cost and communication cost) is minimized.

Let the copy of relation R at site s be chosen to be partitioned into d fragments. We say this is a d-partition of relation R and the partition is denoted by $PT(PS(R, d, s))$, where $PS(R, d, s)$ is the set of processing sites, each of which is assigned one fragment of R. The fragment that is assigned to site p due to this d-partition is denoted by $F(R, d, s, p)$. (When there is no ambiguity, arguments R, d, s, and/or p may be dropped, e.g., $F(p)$ may be used to denote the fragment assigned to p.) At each processing site p, a subquery $q(R_1, \cdots, F(R, d, s, p), \cdots, R_r)$ which references the fragment $F(R, d, s, p)$ and the other relations will be processed. The answer to the query Q is

the union of the answers to the subqueries. We use Cost $(q(R_1, \cdots, F(R, d, s, p), \cdots, R_r), p)$ or simply Cost $(F(p))$ to denote the total cost (time) to process the subquery at site p. This cost includes the delay due to the partitioning of the relation R at site s, the communication delay of transmitting the fragment F and all other relations referenced by the query which are not at site p to site p, and local processing cost for the subquery at site p. The response time to process query Q is $RT(Q(R), d, s) = \max_{p \in PS(R,d,s)} \{\text{Cost } (F(p))\}$. We are interested in minimizing $RT(Q(R), d, s)$.

The advantage to partition a relation into fragments and process each subquery at a different site is that each subquery can be processed in parallel and the size of data involved in processing is smaller. Since only one relation is partitioned, the gain due to this partition may not be significant when the number of relations referenced by a query is large. However, in practice, most queries reference only a few relations. In those situations, the partition strategy is useful. More detailed information on situations where the partitioning strategy is beneficial will be given in Section V.

III. Partition Strategy

A. Cost Model

The total cost of processing a subquery at a site p consists of the cost of partitioning a relation at site s, the data communication cost and the local processing cost for the subquery at site p. These costs are described as follows.

Data communication cost is assumed to be a monotonically increasing (nondecreasing) in the amount of data transferred. It is assumed to be the same between any pair of sites. These assumptions are consistent with those used in earlier query processing algorithms.

The local processing cost for processing the subquery at a site depends on the processing speed of the site, whether the join is supported by fast access paths such as indexes, and the sizes of the relations participating in the join. The cost is assumed to be monotonically increasing in the size of each of the relations to be processed and monotonically decreasing in the processing speed of a site. The cost to process an operation involving a relation with fast access paths is less than that without fast access paths.

Partitioning a relation consists of retrieving the relation from secondary memory, dividing it into fragments, and assigning buffers for the data to be sent to other sites. The partitioned relation needs to be transferred from secondary memory to the buffers only once. Dividing the relation into fragments can be performed by scanning the relation once. Thus, the partitioning cost is assumed to be monotonically increasing in the size of the relation to be partitioned and monotonically decreasing in the processing speed of the site where the relation is to be partitioned, but is independent of the number of partitions.

The total cost of processing a subquery at a site, Cost may be the sum of the partitioning cost, PC, the data communication cost, DC, and the local processing cost, LC, if the three processes do not overlap, i.e,

$$\text{Cost} = PC + DC + LC. \tag{1}$$

In the situations where the processes overlap (the partitioning process is likely to overlap partly with the data communication process, which is likely to overlap partly with the joining process), we may express

$$\text{Cost} = \beta PC + \alpha DC + LC \quad \text{for some } 0 < \alpha, \beta < 1. \tag{2}$$

In other words, $(1 - \beta)PC$ and $(1 - \alpha)DC$ are the time overlapped between the partitioning process and the data communication process and that between the data communication process and the joining process. Since the local processing cost for joining is likely to be dominant, it is left to be the same in the above equations (i.e., LC). Since data communication cost, partitioning cost, and local processing cost are monotonically increasing in relation sizes and total cost can be assumed to be monotonically increasing in each of the three costs by (1) or (2), total cost is monotonically increasing in relation sizes.

In addition to the monotonically increasing property, we assume that the relation to be partitioned is infinitely divisible, and the cost is a continuous function of the relation size. However, in practice, the relation can only be partitioned into fragments having integral number of tuples. Since the practical solution with integral number of tuples and the theoretical solution with possibly nonintegral number of tuples differ by negligible number of tuples and the total cost is a continuous function of relation size, the theoretical solution is a good approximation.

Our analysis to be presented in later sections only requires that the total cost function is continuous, has the monotonically increasing property and the relation to be partitioned can be infinitely divisible. Thus, our result is applicable to a reasonably large class of cost functions.

Our solution is outlined as follows. In Section III-A, we will determine the processing sites and the number of fragments to yield the minimum response time for a given copy of a relation to be partitioned. Then, in Section III-B, based on the result obtained in Section III-A we will determine which copy of a relation should be partitioned to yield the best response time. The above method can be repeated for every relation to determine which relation and which copy should be partitioned. The complete algorithm is given in Section III-C.

B. Determine the Optimal Processing Sites

Given a relation and its copy to be partitioned, we want to determine 1) the processing sites (and therefore the number of partitions) and 2) how the relation is to be partitioned and assigned to the processing sites, such that the response time is minimized.

We first define a quantity which characterizes whether a site should be used as a processing site for a query Q. The total cost to process a subquery at a site p is denoted by Cost $(F(p))$. We define the *partition weight* at site p,

denoted by $PW(R, s, p)$, to be $\text{limit}_{F(p)\to 0}$ Cost $(F(p))$, where $F(p)$ is the fragment of R assigned to site p. Thus, the partition weight is the overhead for partitioning R and the cost for transferring and processing other unfragmented relations for the processing of the subquery at the site. Intuitively, the partition weight is the minimum total cost that a site can complete a subquery no matter how small a partitioned fragment, say in practice an artificial fragment having a tuple, may be assigned to this site. Thus, if site p is to be used as a processing site, it will incur a total cost greater than its partition weight. Since the partitioning cost is assumed to be monotonically increasing in the size of the partitioned relation and monotonically decreasing in the processing speed of the site where the relation is to be partitioned, but is independent of the number of partitioned fragments, we have $PW(R, s, p) \le PW(R, v, p)$ if $Sp(s) \ge Sp(v)$ (assuming there exists copies of R at both sites s and v), where $Sp(x)$ is the processing speed of site x.

Proposition 1 below shows that an optimal way to assign fragments to a given set of processing sites is in such a way that each processing site will have the same total cost. Thus, it remains to determine the processing sites. We will show in Propositions 2 and 3 that 1) if a site is not used as a processing site, then those sites with partition weights higher than or equal to that of the site should not be used as processing sites and 2) if a site is used as a processing site, then those sites with partition weights smaller than or equal to that of the site should be used as processing sites. Based on results 1) and 2) all sites are arranged in ascending order of partition weight. Consider site t; if site t is not a processing site, then sites $(t + 1)$ to the last site need not be considered; otherwise, site 1 to site t and possibly the next few sites will be processing sites. A binary search technique will be used to determine t such that site 1 to site t (and no other sites) are the processing sites.

In the following proposition, the copy of a relation R at site s is partitioned into d fragments and each fragment is assigned to a site of the set of processing sites PS.

Proposition 1: Let RT and RT_1 be the response times obtained by the d-partitions $PT(PS)$ and $PT_1(PS)$ of relation R, respectively. If the d-partition $PT(PS)$ satisfies Cost $(F(p))$ = Cost $(F(j))$ for every pair of processing sites $p, j \in PS$, then $RT \le RT_1$.

Proof: Let Cost $(F(p))$ (Cost $(F_1(p))$) be the total cost by assigning a fragment $F(p)(F_1(p))$ to site p by the d-partition $PT(PS)(PT_1(PS))$.

It is impossible that the size of $F_1(p)$ is smaller than the size of $F(p)$ for all p, $p \in PS$, since

$$\sum_{p\in PS} |F_1(p)| = \sum_{p\in PS} |F(p)| = |R|,$$

where $|X|$ denotes the size of relation or fragment X. Therefore, there exists at least one site j with $F_1(j) \ge F(j)$. We have Cost $(F_1(j))$ $\ge$ Cost $(F(j))$ since $F_1(j) \ge F(j)$. Thus, we have

$$RT_1 = \max_{p\in PS} \{\text{Cost}(F_1(p))\} \ge \text{Cost}(F_1(j)) \ge \text{Cost}(F(j)) = RT,$$

since Cost $(F(j))$ = Cost $(F(p))$ for every pair $p, j \in PS$. □

Proposition 1 says that the best partition of a relation is to make every site complete its processing at the same time.

Definition: OS is *an optimal set of processing sites* for the copy of relation R at site s if the copy is partitioned into fragments and assigned for processing at each of the sites in OS to yield the optimal response time. Each site t in OS is *an optimal processing site*.

In Lemma 1, we show that if the partition weight of a site j is smaller than the response time obtained by a d-partition and j is not one of the processing sites used by this d-partition, then there always exists a $(d + 1)$-partition, including j as a processing site, that will yield a smaller response time. This result will be used to prove that if the partition weight of a site j is smaller than that of an optimal processing site, then site j should also be an optimal processing site (see Proposition 2).

Lemma 1: Let RT be the response time obtained by the d-partition $PT(PS)$ of relation R by using the copy at site s.

If $PW(R, s, j) < RT$, and j is not in PS, then there exists a $(d + 1)$-partition, $PT(PS_1)$, such that $RT_1 < RT$, where RT_1 is the response time obtained by including site j as a processing site.

Proof: Let $PS_1 = PS \cup \{j\}$. Since $PW(R, s, j)$ is smaller than RT, we can assign a fragment $F_1(j)$ to site j such that the fragment is obtained by taking a small piece of data of size Delta (p) from each fragment $F(p)$ in the original d-partition and Cost $(F_1(j)) < RT$. More specifically, the new fragment $F_1(p)$ at site p has size $|F_1(p)| = |F(p)| -$ Delta (p) for every site p, $p \in PS$, where $F(p)$ is the original fragment at site p by the d-partition $PT(PS)$, and Delta $(p) > 0$ and $\Sigma_{p\in PS}$ Delta $(p) = |F_1(j)|$.

We have Cost $(F_1(p)) <$ Cost $(F(p))$ for all p, $p \in PS$, since $F_1(p) < F(p)$. Therefore, $RT_1 = \max_{p\in PS_1}$ Cost $(F_1(p)) < RT$. □

Note: It is assumed in the proof that the function Cost is a continuous function and therefore whenever the inequality is satisfied for site j, there is always a small fragment that can be assigned to the site while preserving the inequality. However, in practice, if the inequality is marginally satisfied, the site will not be used and the solution will differ from the "optimal solution" by at most a small amount.

Proposition 2: If site p is an optimal processing site and site j satisfies $PW(R, s, j) \le PW(R, s, p)$, then site j is also an optimal processing site.

Proof: We know that Cost $(F(p)) > PW(R, s, p)$ for $F(p) > 0$.

Let the optimal response time be OP. Then, since site

p is an optimal site,

$$OP > PW(R, s, p) \geq PW(R, s, j).$$

Suppose that j is not a processing site. By Lemma 1, we can find a $(d + 1)$-partition which includes site j as a processing site and yields a smaller response time. This contradicts that site p is a processing site with optimal response time. □

Proposition 3: If site p is not an optimal processing site and site j satisfies $PW(R, s, j) \geq PW(R, s, p)$, then site j is not an optimal processing site, either.

Proof: We have $PW(R, s, j) \geq PW(R, s, p)$. Suppose that site j is an optimal processing site. By Proposition 2, site p is also an optimal processing site, a contradiction. □

Suppose that R is the relation to be partitioned. After the sites are arranged in ascending order of partition weight, the best solution is obtained by assigning the fragments of R using Proposition 1 to the first d sites, for some integer d. We now seek to determine d.

One simple-minded way is to compute the response time for a given d by assigning the fragments of R to the first d sites using Proposition 1. Then, d is increased by 1 and the process is repeated until an increase in the number of sites yields an increase in the response time or the total number of sites m is reached. For a given d, finding the response time for the first d sites takes $O(d)$ time (assume that the Cost can be computed in constant time at every site). Since the process may be repeated up to m times, the time complexity is at most $O(m^2)$.

A more efficient process is as follows. We first consider the middle site, the $(m/2)$th site. If this is not an optimal processing site (determined by Proposition 4 below), then it is sufficient to consider the first $(m/2 - 1)$ sites. We then repeat the process by checking the $(m/4)$th site. If the $(m/2)$th site is an optimal processing site, then the $(3m/4)$th site will be examined. In other words, a binary search is performed on the set of sites. Suppose that the process of determining whether the ith site is an optimal processing site can be determined in $O(i)$ time (this is dependent on cost model), the time complexity for determining the optimal number of sites is at most $O(m \log m)$, since the binary search process is an $O(\log m)$ process. It remains to determine whether a given site is an optimal processing site.

Suppose the given site is the ith site. Even if this given site is assigned a fragment of size zero (i.e., the site is not a processing site), the partition weight (i.e., the minimum cost) incurred at this site is $T = PW(R, s, i)$. Suppose all tuples of R are assigned to the preceding $(i - 1)$ sites (the parts of R assigned to the sites need not be disjoint) such that each of the $(i - 1)$ sites has response time T. The next proposition states that site i is not an optimal processing site if and only if the sum of the sizes of the fragment of R assigned to the first $(i - 1)$ sites is greater than or equal to the size of R.

Proposition 4: Suppose size (j) is the size of the part F_j of R assigned to site j such that the total cost of site j, Cost $(F_j) = PW(R, s, i)$ for each site $1 \leq j \leq i - 1$. Then site i is not an optimal processing site if and only if

$$\text{size} = \sum_{j=1}^{i-1} \text{size}(j) \geq \text{size}(R) \tag{3}$$

where size (R) is the size of relation R.

Proof: Suppose (3) is satisfied. Then, we can take away a small portion Del (j) from the part assigned to site j, $1 \leq j \leq i - 1$, such that the sum of the sizes of the parts assigned to the first $(i - 1)$ sites is exactly size (R), i.e., the part assigned to site j is changed to F_j' having size$'(j)$ = size (j) − Del (j) with

$$\sum_{j=1}^{i-1} \text{Del}(j) = \text{size} - \text{size}(R).$$

The new assignment is clearly a partition of R, with new total cost equal to Cost $(F_j') \leq$ Cost $(F_j) = PW(R, s, i)$ which is strictly less than the total cost of site i for any assignment of a fragment of nonzero size.

Suppose that (3) is not satisfied. Let a partition of R consist of assigning a fragment F_j'' of R of size size$''(j)$ to site j, $1 \leq j \leq i - 1$. There must exist a site u such that size$''(u) >$ size (u), otherwise,

$$\sum_{j=1}^{i-1} \text{size}''(j) \leq \sum_{j=1}^{i-1} \text{size}(j) < \text{size}(R),$$

implying that R is not partitioned.

Let w be the site with the largest cost, i.e., Cost $(F_w'') \geq$ Cost (F_j''). Cost $(F_w'') \geq$ Cost $(F_u'') >$ Cost $(F_u) = PW(R, s, i)$. Thus, we can reallocate part of the fragment assigned to w to site i to obtain a decrease in response time. (If there is another site v with the same cost as w, then we should do the reallocation to site v as well.) This shows that any partitioning of R using (no more than) the first $(i - 1)$ sites is not optimal. Since optimal response time is obtained by using the first t sites for some t (by Propositions 2 and 3) and $t > i - 1$, site i is an optimal processing site. □

The process of determining the last processing site is given as follows.

```
BINSEARCH(low, high, result)
/* The relation to be partitioned is R; the copy of R to
be partitioned is at site s. The optimal set of processing
sites is 1 to result. */
If (low > high) then result = high
else {
   MID = (low + high)/2; MID1 = MID − 1;
   For i := 1 to MID1
   {
      Assign part F_i' of R with size (i) to site i such
      that
      Cost (F_i') = PW(R, s, MID)}
   If Σ_{i=1}^{MID1} size (i) ≥ size (R)
   then BINSEARCH(low, MID − 1, result) /*search
   left half*/
```

```
    else BINSEARCH (MID + 1, high, result) /*search
    right half*/
}
```
□

Given a relation and its copy to be partitioned, algorithm BINSEARCH determines the optimal processing sites and hence the number of fragments into which the relation should be partitioned.

A result for scheduling jobs, allowing preemption but without considering set-up time was given in [29], [30]. In our problem, the partition weight may be considered as the set-up time for a job. Whether our problem can be couched as a job-scheduling problem with a known solution remains to be seen.

C. Determining the Copy of the Relation to Be Partitioned

When there is more than one copy of the relation to be partitioned, we need to determine which copy should be used, to obtain the best response time. Suppose that we use the copy CO_u at site u that has the fastest processing speed among the sites that have a copy of the relation. Let the set of the optimal processing sites and the optimal response time for this copy be PS_u and OP_u, respectively.

Suppose there is a copy of the relation to be partitioned, CO_v at site v. Proposition 5 below states that if the partition weight of site v using the copy CO_v is greater than or equal to OP_u, then CO_v cannot yield a better response time than OP_u and therefore should not be chosen for partitioning.

Lemma 2 below states, if the set of the optimal processing sites obtained for CO_v is not a subset of the union of PS_u and $\{v\}$, then the response time obtained by using CO_v cannot be less than OP_u. As a result, we do not need to consider any site outside PS_u and v as a processing site for copy CO_v. Furthermore, site v should be a processing site for copy CO_v, otherwise, no better response time can be obtained. Thus, Lemma 2 can be used to restrict the processing sites, if a copy other than CO_u is used.

Lemma 2: Let RT_u and RT_v be the optimal response times obtained by the optimal partitions $PT(PS_u)$ and $PT(PS_v)$ of relation R, respectively, by using the copies CO_u and CO_v of R at sites u and v, respectively. If 1) the processing speed $Sp(u)$ of site u is faster than or equal to the processing speed $Sp(v)$ of site v and 2)-a) PS_v is not a subset of the union of PS_u and $\{v\}$, or b) v is not in PS_v, then $RT_u \leq RT_v$.

Proof: First, we prove for the case involving conditions 1) and 2)-a). Since PS_v is not a subset of the union of PS_u and $\{v\}$, there must exist a site j, $j \neq v$, such that j is in PS_v but is not in PS_u. By Lemma 1, since j is not in PS_u, the partition weight $PW(R, u, j) \geq RT_u$. Therefore,

$$
\begin{aligned}
RT_v &= \text{Cost}\left(F(v, j)\right) \text{ (by Proposition 1)} \\
&\geq PW(R, v, j) \text{ (by the definition of } PW) \\
&\geq PW(R, u, j) \text{ (since } Sp(u) \geq Sp(v) \text{ and } v \neq j) \\
&\geq RT_u,
\end{aligned}
$$

where $F(v, j)$ is the fragment assigned to site j by partition $PT(PS_v)$.

Now, we prove for the case involving conditions 1) and 2)-b). Suppose we use the copy CO_u, partition it exactly the same way as we partition CO_v and assign the fragments to PS_v exactly the same way as for copy CO_v, i.e., $|F(u, j)| = |F(v, j)|$ for every $j \in PS_v$. Since $Sp(u) \geq Sp(v)$ and $v \neq j$, we have $PW(R, u, j) \leq PW(R, v, j)$ and hence Cost $(F(u, j)) \leq$ Cost $(F(v, j))$, for every $j \in PS_v$. The response time obtained for this partition is $RT'_u = \max_{j \in PS_v}\{\text{Cost}\,(F(u, j)\} \leq RT_v$. But RT_u is the optimal response time obtained for copy CO_u. Thus, $RT_u \leq RT'_u \leq RT_v$. □

In the proof of Lemma 2 and the following Proposition 5, we make use of the assumptions that partitioning cost is monotonically increasing in the size of the partitioned relation, monotonically decreasing in the processing speed of the site where the relation is to be partitioned, but is independent of the number of partitioned fragments and how the relation is accessed. If different access paths and other factors are taken into consideration for determining the partitioning cost, we can modify Lemma 2 and Proposition 5 as given in the Appendix.

Proposition 5: Let RT_u and RT_v be the optimal response times obtained by the optimal partitions $PT(PS_u)$ and $PT(PS_v)$ of relation R, respectively, by using the copies CO_u and CO_v of R at sites u and v, respectively. If $Sp(u) \geq Sp(v)$ and the partition weight $PW(R, v, v) \geq RT_u$, then $RT_u \leq RT_v$.

Proof: If v is not in PS_v, then this is a case of Lemma 2. If v is in PS_v, then $RT_v \geq PW(R, v, v) \geq RT_u$. □

When there is more than one copy of the relation to be partitioned, one method to choose which copy to be partitioned is to first use the copy at the site that has the fastest processing speed among the sites that have a copy of the relation, then discard some unnecessary copies by applying Proposition 5. The remaining copies are processed in descending order of processing speed of their residing sites. For each such copy, the processing sites will be restricted to be the intersection of the processing sites of the previous copies, plus the site where the copy resides, since, by Lemma 2, no copy can yield a better response time by using any site outside the processing sites of a previous copy except possibly the site having the copy to be partitioned. Furthermore, the site having the copy to be partitioned should be included in the processing sites. We then compare all such cases and the copy yielding the least response time is chosen. Thus, given a relation to be partitioned, we can determine the optimal copy to be partitioned and the optimal processing sites.

A good heuristic, to determine which copy to be partitioned, is to choose the copy CO_u at the site u that has the fastest processing speed among the sites that contain a copy of the relation to be partitioned. Suppose that CO_u is chosen and the response time is obtained for this copy. Let the size of the fragment assigned to site j, where a copy CO_j of the relation exists, be F_j. (If no copy of the relation can yield a smaller partition weight for the site

using its copy at the site to be partitioned, than the response time obtained using CO_u, then, by Proposition 5, the optimal solution for the relation is obtained.) If we choose CO_j to be partitioned, we incur some additional time at each processing site due to the partition delay by a slower processing site j, but we save transmission cost for site j because F_j is already at site j. If the copy CO_j is chosen to be partitioned, we can reduce the response time by at most the difference of the above two costs. In fast local area networks, the transmission cost is supposed to be small in comparison with processing cost. Thus, the copy at the site with the fastest processing speed should be a good choice.

We choose only one copy of a relation to partition. Fragmenting more than one copy and distributing one fragment from one copy and another fragment from another copy may save some cost. However, that approach is not practical, because a fragment in our environment is not defined by a predicate. Only the number of tuples of the fragment is of significance. Arbitrarily choosing some tuples from a copy to form a fragment and some tuples from another copy to form another fragment, etc., may not give a partition of a relation.

D. A Partition Algorithm

In previous subsections, a method is provided to determine the optimal copy to be partitioned and the optimal processing sites for a given relation. The above method is applied to each relation referenced by the query and the relation yielding the least response time is chosen. This forms an algorithm to find an optimal partition strategy. Let $R(Q)$ be the set of relations referenced by query Q. Let $RS(Q)$ be the set of referenced sites which contain at least a copy of some relation referenced by Q, and $NRS(Q)$ be the set of nonreferenced sites which contain no relations referenced by Q. The set of sites $S = RS(Q) \cup NRS(Q)$, i.e., the union of the referenced and nonreferenced sites of Q. Among the nonreferenced sites, $NRS(Q)$, let t be the site having the fastest processing speed.

In the following algorithm PARTITION, the best single site processing strategy is determined at step 1). At step 2), the partition strategy is applied to each relation to determine the copy to be partitioned, the processing sites, and the response time. All the relations are considered and the one yielding the best response time is chosen to be partitioned if the strategy using it is better than the single site strategy obtained at step 1). The chosen relation, copy, and processing sites are identified by R, COPY, and PS, respectively.

We use CS_v to denote an ordered list (set) of sites which are arranged in the ascending order of their partition weight. We also use first (CS_v), last (CS_v), and $v(CS_v)$ to denote the first, last, and the position of the site containing CO_v in the ordered list of sites CS_v.

Algorithm Partition:

1) Estimate the total time if the query is processed in a single site k, $k \in RS(Q) \cup \{t\}$.

Set Bound = the best estimated response time obtained at some site p.

Processing site $PS = \{p\}$.

2) For every relation R_i, $1 \le i \le n$, referenced by the query Q.

2.1) Arrange all sites p in ascending order of partition weight $PW(R_i, s, p)$, where s is the site having the fastest processing speed among the sites that contain a copy of R_i.

2.2) Discard those sites p with partition weight greater than or equal to Bound.

2.3) Use binary search to determine the optimal processing sites for the copy of R_i at site s.

low = 1; high = the number of sites remained after step (2.2).

BINSEARCH(low, high, result)

2.4) Compute response time RT using the processing sites 1 to result, denoted by $CS_0 = \{1 \cdots \text{result}\}$.

2.5) Order the remaining copies of R_i that can yield smaller partition weight than RT for their residing sites using the copies for partitioning in descending order of processing speeds of their residing sites. (Let r_i be the number of these remaining copies.) Let the arranged copies be CO_v, $1 \le v \le r_i$.

```
For v = 1 to r_i
    { if (PW(R_i, site (CO_v), site (CO_v)) < RT)/
    * By Proposition 5, if the partition weight is not
    less than RT, it cannot yield a response time
    better than RT */
    { CS_v = CS_{v-1} ∪ {site (CO_v)}
      low = v(CS_v); high = last (CS_v)
      BINSEARCH(low, high, result_v)
      Compute the response time RT_v for the processing sites 1 to result_v.
      If (RT_v < RT)
      { RT = RT_v; COPY_i = CO_v;
        PS_i = {1 ··· result_v}
      }
      CS_v = CS_{v-1} ∩ {1 ··· result_v} /*Only
      those sites in the previous processing sites will
      be considered for the next copy*/
    }
}
2.6) If (RT < Bound)
    { Bound = RT
      R = R_i /* partitioned relation */
      COPY = COPY_i /* chosen copy*/
      PS = PS_i /* processing sites*/
    }
```

□

E. Determine the Sizes of the Fragments

Given the relation R, and its copy at site s, to be partitioned and the optimal processing sites, we determine the sizes of the fragments to be assigned to the sites as follows. By Proposition 1, the optimal assignment is to make each processing site have the same total cost. Thus,

for each optimal processing site i,

$$\text{Cost}\,(F(i)) = c$$

where c is the total cost (and hence the response time) to be determined and $F(i)$ is the fragment of R to be assigned to site i. Therefore,

$$F(i) = \text{Cost}^{-1}\,(c). \tag{4}$$

Since $\Sigma_i\,|F(i)| = |R|$, we can solve for c by substituting (4) into it and therefore $F(i)$ can be solved. Suppose that we can solve (4) for $F(i)$ as a function of c in constant time. We need to repeat this for d fragments. Thus, c can be solved in $O(d)$ if the relation is partitioned into d fragments and the sizes of fragments can also be computed in $O(d)$.

IV. Time Complexity for Partition Algorithm

The time complexity to estimate the Cost $(F(p))$ depends on the cost functions used. Since we do not assume any cost function, we will use sq to denote the time to estimate the total cost of a subquery. Thus, the following complexity analysis will be adjusted according to the cost model given.

In the algorithm PARTITION, it repeats for every referenced site and one nonreferenced site to obtain the best response time for Step 1). Thus, it takes $O(|RS(Q)|\,sq)$. At Step 2.1), it will take $O(m(pw + \log m))$ to arrange sites in ascending order of partition weight, where m is the number of sites, pw is the time to estimate partition weight. The sites with partition weight greater than or equal to Bound can be determined in $O(\log m)$ in Step 2.2). Let the number of sites after Step 2.2) be m'. It takes $O(m' \log m')$ for the BINSEARCH algorithm for Step 2.3) (by the assumption that the sizes of fragments can be solved in $O(m')$). At Step 2.4), it takes $O(m')$. Thus, the time complexity for Steps 2.1)–2.4) is $O(m(pw + \log m))$. Suppose that there are v_i copies of relation R_i. At Step 2.5), Steps 2.1)–2.4) are repeated at most v_i times with $O(v_i m(pw + \log m))$. We will repeat this process for every relation. Thus, it needs $(N_d m(pw + \log m) + msq)$ for the algorithm PARTITION, where N_d is the total number of the copies of all the relations.

V. Sensitivity Analyses

In order to study 1) whether an optimal partitioning strategy has significant improvement over the best single site processing strategy in which all data are sent to one site and processed at this single site to answer a query and 2) how the performance of the partition algorithm is affected by different parameters, the following simulations are conducted.

In the simulations, the total cost at each processing site is the sum of the partitioning cost, data transmission cost and the local processing cost at that site as stated in (1) in Section II. The case where the three processes overlap is given by (2) and may be visualized as follows. Comparing (1) and (2), we observe that the partitioning cost component and data communication component in (2) are smaller than the corresponding components in (1). In other words, if the partitioning process and the data communication process overlap, then this can be interpreted as the partitioning process has a smaller cost. Similarly, if the data communication process and the local processing process overlap, then we can interpret this as the data communication cost is smaller. Thus, the case where the partitioning process, the data communication process, and the local processing process overlap can be simulated by assigning smaller cost components for data communication and partitioning. The simulation is carried out by having the following parameters to distinguish different simulation environments.

The first parameter is A, which is used to denote the ratio of joining cost to partition cost. The higher the constant A, the smaller the partition cost in comparison with the joining cost and/or more overlap between the partition and the transmission processes. Two values, 10 and 100, are simulated for A in this simulation. When A is 100, the partition delay is considered negligible. (When A is 10, partition delay is still less than joining cost.)

The second parameter is B, which is used to denote the relative cost between joining cost and transmission cost. The larger the value of B, the smaller the transmission cost in comparison with local processing cost (i.e., the faster the transmission speed in comparison with local processing speed) and/or more overlap between the transmission and local processing processes. In the simulation, two values, 1 and 10, are considered for B. 1 denotes that transmission cost and local processing cost are comparable, while 10 denotes that transmission cost is much smaller.

Different processing speeds at different sites are considered in our simulations. One set of processing speeds are randomly generated between 1 and 100. This is used to find out how significant difference of processing speeds at different sites affects performance. Another set of processing speeds are randomly generated between 80 and 100. This is used to simulate sites having rather uniform processing speeds.

Each relation referenced by a query is randomly assigned a size, randomly assigned to sites and randomly assigned or not assigned a fast access path (index) for joining attribute (the cost to process a join involving a relation with a fast access path for joining attribute is less than that without a fast access path). The number of relations ranges from 2 to 6. The number of sites ranges from 2 to 11.

The response time Y_s obtained by the best single site processing is compared to the optimal response time Y_p, obtained by the partition strategy. The improvement factors is obtained by computing $100\% * (Y_s - Y_p)/Y_p$. The improvement factors shown in Figs. 1–3 are the average results over 1000 runs.

First, we study the effect of the number of sites and the number of relations on the improvement factor. As shown in Figs. 1–3, the improvement factor increases rapidly initially as the number of sites increases, but tapers off as

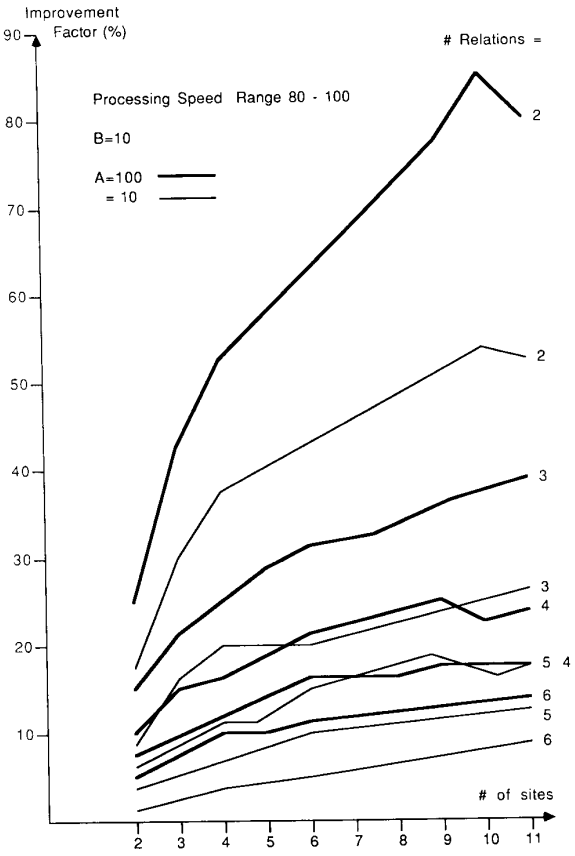


Fig. 1. The effect of the partition cost.

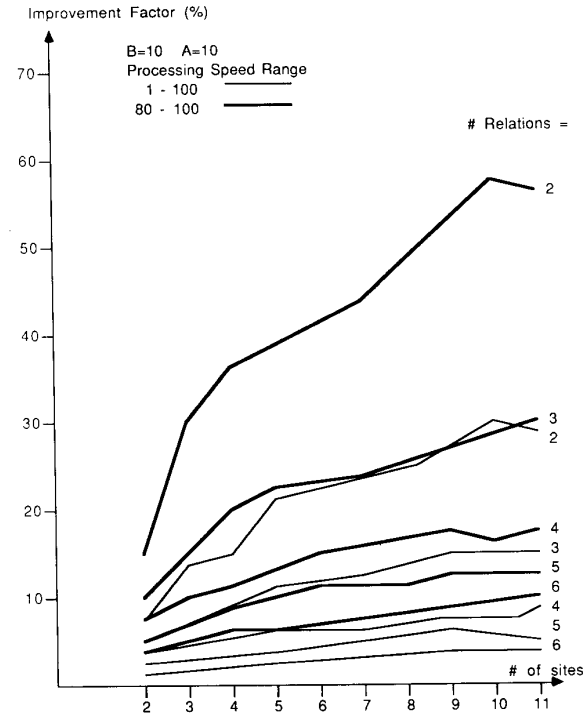


Fig. 2. The effect of the distribution of processing speeds.

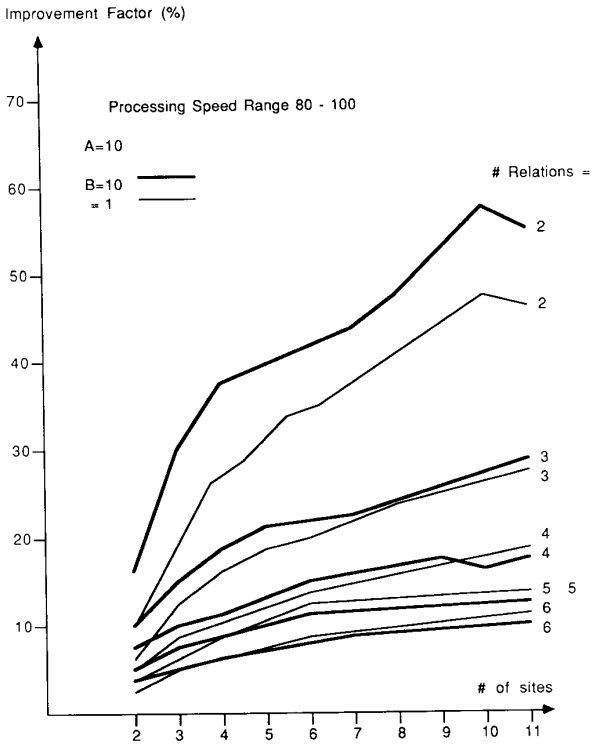


Fig. 3. The effect of transmission speed.

the number of sites continues to increase. This may not be obvious, but it is not surprising, either. When the number of partitioned fragments increases, there is more parallelism. However, when the number of partitioned fragments increases, it incurs more data communication cost due to sending data to more sites. Furthermore, each processing site needs to process not only a fragment of the partitioned relation but also all other relations referenced by the query. So, even if the number of processing sites were to approach infinity, the fragments would be reduced to sizes close to zero, but the other relations still need to be processed. Thus, the benefit of having additional sites after a certain number of sites is marginal. Furthermore, for those sites which originally do not contain any data referenced by the query, local processing costs are high as there is no fast access path to support efficient processing. This result suggests that it is sufficient to consider just a few sites for the partition strategy.

The improvement factor decreases as the number of relations increases. Recall that in a partition strategy, only one relation is partitioned and the gain in local processing cost is due mainly to this partition. Thus, when the number of relations increases, the gain in local processing cost relative to the total cost due to processing all other relations decreases.

Fig. 1 shows the results of two sets of simulations. One is for low partition cost, i.e., $A = 100$ (solid line) and another is for high partition cost, i.e., $A = 10$ (dotted line). When the partition cost is smaller (i.e., $A = 100$)

in comparison with the processing cost, the improvement factor is higher.

In Fig. 2, we study the effect of the distribution of processing speeds among sites on the performance of the partition strategy. When the distribution of processing speeds is more uniform (i.e., processing speeds range from 80 to 100), the improvement factor is higher. This is apparent. If there are significant differences in processing speeds of sites, it is preferable to send data to those few sites with faster processing speeds. However, this lowers the degree of parallelism.

In Fig. 3, we study the effect of the relationship between transmission cost and local processing cost on the performance of the partition strategy. We compare two cases, $B = 1$ (local processing cost and transmission cost are comparable) and $B = 10$ (local processing cost is dominant). There is not much difference between the two cases when the number of relations is large (e.g., 4 in this simulation). However, for a small number of relations, there is much more improvement when transmission cost is lower. Clearly, when transmission cost is comparable to local processing cost, improvement of the partition strategy over single site processing may not be possible. This is unlikely to occur in local area networks where data communication cost is not as significant as local processing cost [27].

By these simulation results, the partition strategy is shown to be feasible in fast local networks. The factors that favor the partition strategy are 1) small number of relations, 2) small partition cost (in comparison to joining cost), 3) small difference among processing speeds of sites, and 4) fast transmission speed.

VI. Incorporating the Semijoin Operation

After the relation to be partitioned and the processing sites have been determined, we want to decide whether response time can be reduced further by performing some semijoins. A semijoin sj: R_i-$A \rightarrow R_k$ is the selection of the tuples of R_k whose values under attribute A are in $R_i.A$. For example, a query references three relations, R_1, R_2, and R_3. There is a join between R_1 and R_2 on attribute A, and a join between R_2 and R_3 on attribute B. Suppose that R_1 is to be partitioned. We may decide whether one of the following semijoins, R_1-$A \rightarrow R_2$, R_2-$B \rightarrow R_3$, and R_3-$B \rightarrow R_2$, should be performed to improve response time. We will not perform the semijoin to the partitioned relation. Usually, R_k is very much reduced in size by the semijoin. Since joining of various relations takes place in the last phase of the partitioning algorithm, and the joining cost is monotonically increasing in relation size, it may be worthwhile to reduce some of the relations through semijoins before distributing relations and performing the joins.

If a semijoin is performed, the partition weight and hence the total cost of a site will be changed. Let $RT(*)$ be the response time obtained before performing semijoin sj and $F(p, *)$ be the fragment assigned to site p. After semijoin sj is performed, the original total cost of site p, Cost $(F(p, *))$, is changed to Cost $(F(p, *), sj)$ (semijoin cost is included). The profit obtained at site p due to performing semijoin sj is defined to be $RT(*)$ − Cost $(F(p, *), sj)$ and is denoted by $PF(p, sj)$. For a processing site p, we have $RT(*)$ − Cost $(F(p, *), sj)$ = Cost $(F(p, *)$ − Cost $(F(p, *), sj)$ (since $RT(*)$ = Cost $(F(p, *))$, see Proposition 1). If the total cost of a site becomes less than the original response time, there is a positive profit at the site; otherwise, there is a negative profit. If the profits of all processing sites are positive, then response time can be reduced by performing this semijoin. If the profits of all processing sites are negative, response time will be increased if this semijoin is performed. It is also possible that some processing sites have positive profits but other processing sites have negative profits. We want to decide whether we can redistribute the partitioned relation, i.e., assign less (or do not assign) data of the partitioned relation to those sites with negative profit and assign more to those sites with positive profit, such that response time can be reduced.

Let $RT(*)(RT(sj))$ be the response time obtained by the partition $PT(PS(*))(PT(PS(sj)))$ of R without (with) performing semijoin sj, R_i-$A \rightarrow R_k$, where $R_k \neq R$ and * denotes that no semijoin is performed. The semijoin is usually not performed on R, since only part of R which is the fragment of R assigned at a site, is processed at the site. Let $F(p, *)$ be the fragment assigned to site p by partition $PT(PS(*))$.

Suppose that a semijoin sj is to be performed, and the profit at site p, $PF(p, sj)$, is negative. This negative profit means that the total cost at site p will increase if sj is performed. Thus, in order to lower the total cost at site p to be below the original total cost (i.e., the original response time), we must assign a smaller fragment (or no fragment) of R (instead of $F(p, *)$) to site p. Let $F(p, sj)$ be the largest fragment that can be assigned to site p after semijoin sj is performed such that the new total cost Cost $(F(p, sj)$ (which is the total cost to process subquery at site p by using $F(p, sj)$ instead of $F(p, *)$ after semijoin sj is performed) is equal to the original response time (i.e., the original total cost Cost $(F(p, *))$ before performing the semijoin at site p).

We determine $F(p, sj)$ as follows. After performing a semijoin sj, the quantity $M = PW(p, sj)$, which is the partition weight of site p after semijoin sj is performed, is the minimum startup cost at site p even if a fragment of relation R of size close to zero is assigned to p for processing. If $M \geq RT(*)$, then there is no way we can assign any fragment $F(p, sj)$ to site p such that Cost $(F(p, sj))$ is less than $RT(*)$. Therefore, site p should be excluded from the set of processing sites if the semijoin sj is performed. Hence, $F(p, sj)$ should be zero. If $M < RT(*)$, then we can assign $F(p, sj)$ to site p such that Cost $(F(p, sj)) = RT(*)$. In the first situation, site p will not be used as a processing site because its new total cost is always larger than the original response time (i.e., the original total cost). In the second situation, $F(p, sj)$ is the largest fragment that we can assign to site p such

that the new total cost is the same as the original response time (before performing semijoin) at site p. Let $F'(p, sj) = F(p, sj) - F(p, *)$. Then $|F'(p, sj)|$ (note that $F'(p, sj)$ is negative here) is the smallest amount of data that we must redistribute from site p to other sites.

Suppose that the profit $PF(p, sj)$ is positive. This positive profit means that the total cost at site p will decrease if sj is performed. Thus, this site can receive more data than it was assigned. Let $F(p, sj)$ be the largest fragment that we can assign to site p such that the new total cost is equal to the original response time (i.e., the original total cost before performing semijoin sj at site p). Then, $F(p, sj)$ satisfies Cost $(F(p, sj)) = RT(*)$. *Since* $PF(p, sj) > 0$, $F(p, sj) > F(p, *)$. In this case, the largest amount that we can reallocate from other sites to site p in addition to $F(p, *)$ is $F'(p, sj) = F(p, sj) - F(p, *) > 0$.

In the above, we discuss redistribution among the processing sites. It is possible that some nonprocessing sites will have a lower partition weight after semijoin sj is performed. We may consider assigning a fragment to those sites. The size of fragments previously assigned to nonprocessing sites is zero. That is $F(p, *) = 0$ if p is not in $PS(*)$. We can use the same method discussed above to determine $F(p, sj)$ which is the amount we can assign to site p such that the new total cost is equal to the original response time.

The following proposition states a necessary and sufficient condition that a semijoin is useful to reduce response time.

Proposition 6: Let $F'(p, sj) = F(p, sj) - F(p, *)$, where $F(p, *)$ is the fragment size assigned to site p by optimal partition $PT(PS(*))$ of relation R and $F(p, sj)$ is the largest fragment that can be assigned to site p after the execution of semijoin sj such that the new total cost Cost $(F(p, sj))$ is the same as the original response time, i.e., $F(p, sj)$ satisfies Cost $(F(p, sj)) = RT(*)$, if $RT(*) > PW(p, sj)$, otherwise, $F(p, sj) = 0$.

$\Sigma_p F'(p, sj) \geq 0$ if and only if the semijoin sj yields a smaller response time, i.e., there exists a partition $PT(PS(sj))$ such that $RT(sj) \leq RT(*)$, where $RT(sj)$ and $RT(*)$ are the response times obtained by the partitions $PT(PS(sj))$ and $PT(PS(*))$, respectively.

Proof: Sufficiency: Consider the following two cases:

1) Suppose that $F'(p, sj) \geq 0$ for every site p, $p \in PS(*)$. Since $F'(p, sj) \geq 0$ for every site p, we have nonnegative profit for every site p. Therefore, we can make $PT(PS(sj)) = PT(PS(*))$ and have $RT(sj) \leq RT(*)$.

2) Suppose that there exists a site u, $u \in PS(*)$, with $F'(u, sj) < 0$. In this case, we have a negative profit at site u. We must reduce $F(u, *)$ by at least $|F'(u, sj)|$ such that the new total cost at site u will not exceed the total cost at site u before performing semijoin sj, and redistribute it to some other sites with positive profit.

Since $\Sigma_p F'(p, sj) \geq 0$, there must be a set of sites PJ with $F'(v, sj) > 0$ for v, $v \in PJ$, such that $\sigma_{v \in PJ} F'(v, sj) + F'(u, sj) \geq 0$. Therefore, we can redistribute a portion, $F^u(v, sj)$, of $|F'(u, sj)|$ to site v such that $F^u(v, sj) \leq F'(v, sj)$ for every site v, $v \in PJ$, and $\sigma_{v \in PJ} F^u(v, sj) = F'(u, sj)$. After redistribution, $F'(v, sj)$ is reduced to $F'(v, sj) - F^u(v, sj)$. We make $PS(sj) = PS(*) \cup PJ - \{u\}$ if $F(u, sj) = 0$, otherwise $PS(sj) = PS(*) \cup PJ$. We can repeat this process for every site with a negative profit. By this redistribution, the new total cost at site p, $p \in PS(sj)$ will not exceed the original response time obtained by $PT(PS(*))$. Thus, $RT(sj) \leq RT(*)$.

Necessity: We now show that if $\Sigma_p F'(p, sj) < 0$, then there does not exist any partition such that the response time obtained is less than or equal to the original response time (i.e., $RT(*)$). First, if a site p has $F(p, sj) = 0$, the total cost incurred at this site is not less than the original response time, even if we assign a fragment having zero size. Thus, those sites p with $F(p, sj) = 0$ can not be included in $PS(sj)$. Second, we now show that if $PS(sj)$ is the set of sites p with $F(p, sj) > 0$, the response time obtained by using all these processing sites is higher than the original response time. Since $\Sigma_p F'(p, sj) < 0$, $\Sigma_p F(p, sj) < \Sigma_p F(p, *)$. In other words, if $F(p, sj)$ is assigned to every site p with $F(p, sj) > 0$, a portion of size $R - \Sigma F(p, sj) > 0$ remains unassigned. But after the assignment of $F(p, sj)$, each such site p has total cost equal to the original response time $RT(*)$. Since the remained portion has to be assigned and any additional assignment to any site p causes the total time to go beyond the original response time, the desired result follows. □

Each possible semijoin is examined against the condition given in Proposition 6. If the condition is satisfied, then the semijoin, if executed, will yield a smaller response time than simply partitioning a relation applying the fragment and replicate approach.

VII. Conclusion

The fragment and replicate strategy permits parallel processing of a query. However, if no relation referenced by a query is fragmented, it is necessary to decide which relation is to be partitioned into fragments, which copy of the relation should be used, how the relation is to be partitioned and where the fragments are to be sent for processing. A general partition algorithm has been given to provide answers to the above questions.

Simulation results show that the partition strategy is feasible in fast local network environments. The results also show that the number of partitions does not need to be large. The factors that favor the partition strategy are 1) a small number of relations referenced by the given query, 2) a small partition cost (in comparison with joining cost), 3) fast transmission speed, and 4) rather uniform processing speeds among sites.

An incorporation of the use of semijoin in the partition strategy to improve response time is given. A necessary and sufficient condition that a semijoin yields improve-

ment when used in conjunction with the partition strategy is provided.

As shown in the simulation results, when the number of relations increases, the percentage of improvement decreases. This suggests that we may need to partition more than one relations to increase the improvement. This needs to be investigated in the future.

The following two situations have not been taken into consideration by our model. 1) The cost of assembling results of the subqueries executing in the processing sites is not included in our cost model. When a partial result is transferred to the query site, it is to be unioned with the partial results that have been assembled so far. The cost of performing the union may depend on the number of partial results (which is the same as the number of processing sites). 2) When a relation is partitioned into d fragments and sent across the network to the d processing sites, the cost of transmission (or network contention) may depend on d as well as the amount of data transfer. This is also not considered in our model.

In both situations, the cost of assembling the data and the cost of distributing the fragments go up as d increases. We can handle these situations by a slight modification of our algorithm. If d_1 and d_2 are numbers of processing sites, $d_1 < d_2$, such that the response time using d_1 processing sites is only "slightly higher" than that using d_2 processing sites, then we will use the former solution. We intend to explore this issue further in an implementation in the near future.

Appendix

Let Par (R, u) and Par (R, v) be the partitioning costs for partitioning the copies of R at sites u and v, respectively.

Lemma A1: Let RT_u and RT_v be the optimal response times obtained by the optimal partitions $PT(PS_u)$ and $PT(PS_v)$ of relation R, respectively, by using the copies CO_u and CO_v of R at sites u and v, respectively. If 1) Par $(R, v) \geq$ Par (R, u) and 2)-a) PS_v is not a subset of the union of PS_u and $\{v\}$, or b) v is not in PS_v, then $RT_u \leq RT_v$.

Proof: We first prove for the case 1) and 2)-a). Since PS_v is not a subset of the union of PS_u and $\{v\}$, there must exist a site j, $j \neq v$, such that j is in PS_v but is not in PS_u. By Lemma 1, since j is not in PS_u, the partition weight $PW(R, u, j) \geq RT_u$.

The partition weight of a site is the time for partitioning R and the cost for transferring and processing other unfragmented relations for the processing of the subquery at the site. Therefore, when $j \neq v$ and $j \neq u$, we have $PW(R, v, j) =$ Par $(R, v) + C$ and $PW(R, u, j) =$ Par $(R, u) + C$, where C is the time for transferring and processing relations at site j, which is independent of the copy to be partitioned. We can easily obtain $PW(R, v, j) \geq PW(R, u, j)$ if Par $(R, v) \geq$ Par (R, u). The above result is also true for $j = u$. Therefore,

$$\begin{aligned} RT_v &= \text{Cost}\left(F(v, j)\right) \text{ (by Proposition 1)} \\ &\geq PW(R, v, j) \text{ (by the definition of } PW) \\ &\geq PW(R, u, j) \text{ (since Par } (R, v) \geq \text{Par } (R, u) \\ &\qquad \text{and } v \neq j) \\ &\geq RT_u, \end{aligned}$$

where $F(v, j)$ is the fragment assigned to site j by partition $PT(PS_v)$.

We now prove for case 1) and 2)-b). Suppose we use the same processing sites PS_v and partition $PT(PS_v)$ but partitioning copy CO_u, i.e., $|F(u, j)| = |F(v, j)|$, for every $j \in PS_v$. Since Par $(R, v) \geq$ Par (R, u) and $j \neq v$, we have $PW(R, u, j) \leq PW(R, v, j)$ and hence Cost $(F(u, j)) \leq$ Cost $(F(v, j))$. The response time obtained by the above partition is $RT'_u = max_{j \in PS_v} \leq RT_v$. But RT_u is the optimal response time obtained for copy CO_u. Thus, $RT_u \leq RT'_u \leq RT_v$. □

Proposition A1: Let RT_u and RT_v be the optimal response times obtained by the optimal partitions $PT(PS_u)$ and $PT(PS_v)$ of relation R, respectively, by using the copies CO_u and CO_v of R at sites u and v, respectively. If Par $(R, v) \geq$ Par (R, u) and the partition weight $PW(R, v, v) \geq RT_u$, then $RT_u \leq RT_v$.

Proof: if v is not in PS_v, then this is a case of Lemma A1. If v is in PS_v, then $RT_v \geq PW(R, v, v) \geq RT_u$. □

By the above Lemma and Proposition A1, the copies of a relation is ordered by the partitioning cost instead of the processing speeds when we want to determine the copy to be partitioned.

Acknowledgment

The authors would like to acknowledge and thank the anonymous referees for their valuable suggestions.

References

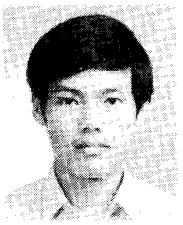
[1] P. Apers, A. Hevner, and S. B. Yao, "Optimization algorithm for distributed queries," *IEEE Trans. Software Eng.*, 1983.
[2] P. A. Bernstein and D.-M. W. Chiu, "Using semi-joins to solve relational queries," *J. ACM*, pp. 25-40, 1981.
[3] P. A. Bernstein and N. Goodman, "The theory of semi-join," CCA, Tech. Rep., Nov. 1979.
[4] P. A. Bernstein, N. Goodman, E. Wong, C. Reeve, and J. B. Rothnie, "Query processing in SDD-1: A system for distributed databases," *ACM TODS*, vol. 6, no. 4, pp. 602-625, Dec. 1981.
[5] D. Bitton, D. DeWitt, and C. Turbyfil, "Benchmarking database systems: A systematic approach," in *Proc. Conf. VLDB*, 1983.
[6] P. A. Black and W. S. Luk, "A new heuristic for generating semijoin programs for distributed query processing," in *Proc. IEEE COMPSAC*, 1982.
[7] D. Brill, M. Templeton, and C. Yu, "Distributed query processing strategies in Mermaid: A frontend to data management systems," *IEEE Data Eng.*, pp. 211-218, 1985.
[8] A. Chan, U. Dayal, S. Fox, N. Goodman, D. Ries, and D. Skeen, "Overview of an ADA compatible distributed database manager," in *Proc. ACM SIGMOD 83*, pp. 228-242.
[9] J. M. Chang, "A heuristic approach to distributed query processing," in *Proc. Conf. VLDB*, 1982.
[10] ——, "Query processing in a fragmented data base environment," Bell Lab., Tech. Rep., 1982.

[11] A. L. P. Chen and V. O. K. Li, "Deriving optimal semi-join programs for distributed query processing," in *Proc. IEEE INFOCOM*, San Francisco, CA, Apr. 1984.
[12] ——, "Optimizing star queries in a distributed database system," in *Proc. Conf. VLDB*, Singapore, Aug. 1984.
[13] ——, "Improvement algorithms for semi-join query processing programs in distributed database systems," *IEEE Trans. Comput.*, Nov. 1984.
[14] D.-M. Chiu, P. Bernstein, and Y. C. Ho, "Optimizing chain queries is a distributed database system," *SIAM J. Comput.*, Feb. 1984.
[15] D.-M. W. Chiu and Y. C. Ho, "A method for interpreting tree queries into optimal semi-join expressions," in *Proc. ACM SIGMOD*, 1980, pp. 169-178.
[16] W. W. Chu and P. Hurley, "Optimal query processing for distributed database systems," *IEEE Trans. Comput.*, vol. C-31, no. 9, pp. 835-850, Sept. 1982.
[17] R. Epstein, M. Stonebreaker, and E. Wong, "Distributed query processing in relational database systems," in *Proc. ACM SIGMOD* 1978, pp. 169-180.
[18] B. Gavish and A. Segev, "Set query optimization in distributed database systems," *ACM TODS*, vol. 11, no. 3, 1986.
[19] N. Goodman, P. A. Bernstein, E. Wong, C. Reeve, and J. B. Rothnie, "Query processing in a system for distributed databases (SDD-1)," CCA, Tech. Rep., 1979.
[20] N. Goodman and O. Shmueli, "Transforming cyclic schemes into trees," in *Proc. ACM SIGACT-SIGMOD Conf. Principles of Databases*, 1982.
[21] A. Hevner and S. B. Yao, "Query processing in distributed database systems," *IEEE Trans. Software Eng.*, vol. 5, no. 3, pp. 177-187, 1979.
[22] ——, "Querying distributed databases on local area networks," *Proc. IEEE*, vol. 75, no. 5, pp. 563-572, May 1987.
[23] M. Jarke and J. Koch, "Query optimization in database systems," *ACM Comput. Surveys*, June 1984.
[24] L. Kerschberg and S. B. Yao, "Optimal distributed query processing," Bell Lab., Holmdel, NJ, 1980.
[25] G. Lohman, C. Mohan, L. Hass, B. Lindsay, P. Selinger, and P. Wilms, "Query processing in R*," IBM, Res. Rep. RJ4272, Apr. 1984.
[26] W. C. Luk and L. Luk, "Optimizing query processing strategies in a distributed database system," Simon Fraser Univ., Burnaby, B. C., Canada.
[27] L. F. Mackert and G. M. Lohman, "R* optimizer validation and performance evaluation for distributed queries," in *Proc. Conf. VLDB*, Kyoto, Japan, 1986.
[28] L. F. Mackert and G. M. Lohman, "R* optimizer validation and performance evaluation for local queries," in *Proc. Conf. ACM SIGMOD*, 1986.
[29] R. McNaughton, "Scheduling with deadlines and loss functions," *Management Sci.*, vol. 6, no. 1, pp. 1-12, Oct. 1959.
[30] R. R. Muntz and E. G. Coffman, Jr., "Preemptive scheduling of real-time tasks on multiprocessor systems," *J. ACM*, vol. 17, no. 2, pp. 324-338, Apr. 1970.
[31] G. Pelagatti and F. A. Schreiber, "A model of an access strategy in a distributed database system," in *Proc. Conf. Database Architecture*, Venice, Italy, 1979.
[32] D. Sacca and G. Wiederhold, "Database partitioning in a cluster of processors," *ACM Trans. Database Syst.*, vol. 10, no. 1, pp. 29-56, Mar. 1985.
[33] G. M. Sacco, "Fragmentation: A technique for efficient query processing," *ACM TODS*, vol. 11, no. 2, pp. 113-133, June 1986.
[34] A. Segev, "Optimization of join operations in horizontally partitioned database systems," *ACM TODS*, vol. 11, no. 1, 1986.
[35] P. Selinger and M. Adiba, "Access path selection in distributed database systems," in *Proc. First Int. Conf. Distributed Data Bases*, Aberdeen, 1980.
[36] M. Templeton, D. Brill, A. Chen, S. Dao, and E. Lund, "Mermaid experiences with network operations," *IEEE Data Eng.*, 1986.
[37] J. D. Ullman, *Principles of Database System*, 2nd ed. Rockville, MD: Computer Science Press, 1982.
[38] Williams *et al.*, "R*: An overview of the architecture," in *Proc. 2nd Int. Conf. Databases*, 1982.
[39] E. Wong and R. H. Katz, "Distributing a database for parallelism," in *Proc. Conf. ACM SIGMOD*, 1983, pp. 23-29.
[40] E. Wong, "Retrieving dispersed data from SDD-1: A system for distributed databases," in *Proc. Berkeley Workshop Distributed Data Management and Computer Networks*, Berkeley, CA, 1977.
[41] S. B. Yao, "Optimization of query evaluation algorithms," *ACM TODS*, vol. 4, no. 2, pp. 133-155, June 1979.
[42] C. T. Yu and C. C. Chang, "Distributed query processing," *ACM Comput. Surveys*, vol. 16, no. 4, pp. 399-433, Dec. 1984.
[43] C. T. Yu, C. C. Chang, M. Templeton, D. Brill, and E. Lund, "On the design of a distributed query processing strategy," in *Proc. Conf. ACM SIGMOD*, 1983, pp. 30-39.
[44] ——, "Mermaid: An algorithm to process queries in a fragmented database environment," *IEEE Trans. Software Eng.*, pp. 795-810, Aug. 1985.
[45] C. T. Yu, K. C. Guh, D. Brill, and A. L. P. Chen, "Partitioning relation for parallel processing in fast local networks," in *Proc. Int. Conf. Parallel Processing*, 1986.
[46] C. T. Yu, K. C. Guh, C. C. Chang, C. H. Chen, M. Templeton, and D. Brill, "An algorithm to process queries in a fast distributed network," in *Proc. IEEE Real-Time Systems Symp.*, 1984, pp. 115-122.
[47] C. T. Yu, K. C. Guh, and A. L. P. Chen, "An integrated algorithm for distributed query processing," in *Proc. IFIP Conf. Distributed Processing*, Oct. 5-7, 1987.
[48] C. T. Yu, K. C. Guh, W. Zhang, M. Templeton, D. Brill, and A. Chen, "Algorithms to process distributed queries in fast local networks," *IEEE Trans. Comput.*, vol. C-36, no. 10, pp. 1153-1164, Oct. 1987.
[49] C. T. Yu, K. Lam, C. C. Chang, and S. K. Chang, "A promising approach to distributed query processing," in *Proc. Berkeley Workshop Distributed Data Managment and Computer Networks*, Berkeley, CA, Feb. 1982, pp. 363-390.
[50] C. T. Yu, L. Lilien, K. C. Guh, M. Templeton, D. Brill, and A. Chen, "Adaptive techniques for distributed query optimization," in *Proc. Int. Conf. Data Engineering*, Los Angeles, CA, Feb. 1986.
[51] C. T. Yu and M. Z. Ozsoyoglu, "An algorithm for tree-query membership of a distributed query," in *Proc. IEEE COMPSAC*, 1979, pp. 306-312.

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.

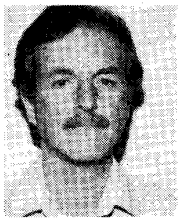
He is currently a Professor in the Department of Electrical Engineering and Computer Science at the University of Illinois at Chicago. He has published in various journals and conference proceedings, including *Journal of the ACM*, *Communications of the ACM*, *ACM Transactions on Data Base Systems*, *ACM Computing Surveys*, *Journal of Theoretical Computer Science*, *Journal of Computer & System Science*, *Information Processing & Management*, *Information Processing Letters*, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, IEEE TRANSACTIONS ON COMPUTERS, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, *Information Technology*, *Canadian Journal of Operation Research & Information Processing*, *ACM SIGMOD*, *VLDB*, *ACMSIGIR*, *IFIP*, *IEEE COMPSAC*, *IEEE Data Engineering*, and *ASIS*. He has served as a consultant for various corporations.

Dr. Yu has served as Chairman of the ACM Special Interest Group on Information Retrieval, an advisory committee member for the National Science Foundation, and Program Committee Chairman for the annual ACMSIGIR conference.



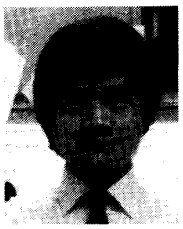
Keh-Chang Guh (S'85-M'86) received the B.S. and M.S. degrees in electrical engineering from National Cheng Kung University, Tainan, Taiwan, Republic of China, in 1977 and 1979, respectively, and the Ph.D. degree in electrical engineering and computer science from the University of Illinois at Chicago in 1986.

Since 1986, he has been an Assistant Professor in the Department of Electrical Engineering and Computer Science, University of Wisconsin-Milwaukee. His research interests include distributed database management systems and expert systems.



David Brill received the B.A. degree from the 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, Santa Monica, CA, 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.



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, in 1984.

He is currently a member of Technical Staff at Bell Communications Research, Piscataway, NJ, and Adjunct Assistant Professor in the Department of Electrical Engineering and Computer Science at Polytechnic University, Brooklyn, NY, where he teaches a course on compiler and formal lanuages. 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, and was a member of the ANSI/X3/SPARC/ Database Systems Study Group.

Expressions for Completely and Partly Unsuccessful Batched Search of Sequential and Tree-Structured Files

YANNIS MANOLOPOULOS AND J. (YANNIS) G. KOLLIAS

Abstract—**A number of previous studies derived expressions for batched searching of sequential and tree-structured files on the assumption that all the keys in the batch exist in the file, i.e., all the searches are successful. New formulas for batched searching of sequential and tree-structured files are derived, but the assumption made now is that either all or part of the keys in the batch do not exist in the file, i.e., the batched search is completely or partly unsuccessful.**

Index Terms—**Access strategy, batched searching, performance evaluation, physical database design, sequential and tree-structured files, successful and unsuccessful search.**

I. Introduction

THIS paper considers a file residing in a secondary storage device, which is physically partitioned into fixed size blocks (e.g., disks). A query based on a primary key value (e.g., social security number) is satisfied by one record (i.e., successful search) or the requested record does not exist in the file (i.e., unsuccessful search). The studies in [1], [2] present a number of file organization schemes (e.g., sequential, random, tree-structured, etc.) and estimate the cost of both successful and unsuccessful searches using these schemes. These costs are expressed by the required number of block accesses to satisfy the query. A query based on secondary key values (e.g., date of birth, sex, etc.) is satisfied by accessing a number of records located normally in more than one block of secondary memory. The blocks containing the records of interest are usually established by employing secondary indexing techniques [1], [2].

Let us assume that we have to satisfy k queries based on either primary or secondary key values. Shneiderman and Goodman [3] argued that the response time of satisfying the queries may be reduced if we consider them as a **batch** instead of satisfying them individually on a first-come-first-served basis. A number of studies considering batching appeared in the literature. Mainly, they report estimations on the number of blocks of secondary storage that have to be transferred in main memory for various environments. The assumptions made by all previous studies is that the records are retrieved under a replacement or a nonreplacement model. The replacement (nonreplacement) model assumes that the probability of locating a record in a specific block is not (is) reduced when a block has already been accessed.

Studies based on the replacement model assumption derived expressions for the expected value of block accesses required to satisfy a request for k keys using sequential [4] or random files [4]–[6]. Under the assumption of the nonreplacement model, expressions have also been derived for sequential [3], [4], [7]–[9], random [4], [10]–[13] and tree-structured files [3], [8], [14]. Table I lists the above mentioned studies according to the file organization and the model concerned.

In this paper we focus on batched searching of sequential and tree-structured files and, therefore, we start discussing the relative studies in more detail. In [3] approximate formulas are derived evaluating the gain due to batched searching of sequential files and of j-ary search trees. In [7] another approximate solution was given for the cost of batched searching in sequential file structures. Recently, [8] derived exact and approximate formulas for the cost of batched searching in both the sequential and tree-structured environments. The same problem for tree-structured files was also examined in [14], where an accurate formula for the gain was derived. We note that similar exact formulas were derived estimating the cost of batched searching in an array [15] and in a main memory database [16], as well as the cost of seeking in a disk system [17]–[19].

A common characteristic of all the previous studies is that they assume that all the records of the batch exist in the file, i.e., that the search is successful. In this paper the last assumption is dropped and the performance of completely or partly unsuccessful batched searching is examined. We say that a batched search is **completely unsuccessful** or **partly unsuccessful** when all the keys or some keys of the batch do not exist in the file respectively. Before proceeding further, we note that erroneous input and missing records from the file (possibly because file updates are performed off-line) are among the reasons which may cause completely and partly unsuccessful

Manuscript received August 31, 1987; revised September 30, 1988.

Y. Manolopoulos is with the Department of Electrical Engineering, Division of Electronics and Computer Engineering, Aristotelian University of Thessaloniki, 54006 Thessaloniki, Greece.

J. G. Kollias is with the Department of Electrical Engineering, Division of Computer Science, National Technical University of Athens, 15773 Zografou, Athens, Greece.

IEEE Log Number 8927390.