Using Parallel Semi-Join Reduction to Minimize Distributed Query Response Time

Xuemin Lin*, Maria E. Orlowska†, and Xiaofang Zhou‡

Abstract

A one-shot semi-join reduction approach was recently proposed to execute all semi-joins on the same relation simultaneously such that the relation only needs to be scanned once. The one-shot semi-join reduction approach was applied to reducing distributed query response time under the assumption that one copy of each referenced relation has been chosen prior to the production of an execution plan. The estimations of both semi-join reduction effect and local join cost, employed in previous work, were restricted to a special case. In this paper, we extend the previous work in three ways: 1) remove the requirement for copy selection before the production of a semi-join reduction program, 2) allow the choice of redundant copies for the execution of semi-joins, 3) employ a general cost model which covers a large class of possible estimations of semi-join reduction effect and local join cost. Then, an algorithm to produce an optimal parallel one-shot semi-join reduction program for minimizing response time shall be presented, addressing the above three aspects.

1 Introduction

Query processing in distributed relational database systems has been extensively studied for more than one decade. To efficiently execute a distributed query, emphasis on the reduction of transmission cost for data movements among different sites has been made. In recent investigations of distributed query processing, the following assumptions [9] are usually adopted for the underlying distributed systems: a) the transmission cost for data shipping is more expensive than the local processing cost, and b) local processing cost cannot be entirely neglected. We shall make these assumptions in this paper.

"Semi-join" is a major method used to reduce transmission cost. A *semi-join* [2] on a relation R_i with another relation R_j, denoted as $R_i \ltimes R_j$, is defined as the projection of $R_i \bowtie R_j$ on R_i. Consider a join $R_i \bowtie R_j$ where R_i and R_j are located at different sites. Let the *result site*, where the join result is required, be the site with R_j. A *semi-join reduction* approach to process this join is as follows:

> The join attributes in R_j, say R'_j, are sent to the site where R_i is located to perform a semi-join there, and then $R_i \ltimes R'_j$ is sent to the result site to perform the join.

As $(R_i \ltimes R'_j) \bowtie R_j \equiv R_i \bowtie R_j$, the transmission cost can be reduced by this semi-join if the sum of the data volumes of R'_j and $R_i \ltimes R'_j$ is smaller than that of R_i. (In this case, we say that R_i is *semi-join reduced by* R_j.) However, the sum of the data volumes of R'_j and $R_i \ltimes R'_j$ might be larger

*Department of Computer Science, The University of Western Australia, Nedlands, WA 6907, Australia, e-mail: lxue@cs.uwa.oz.au

†Department of Computer Science, The University of Queensland, QLD 4072, Australia, Email: maria@cs.uq.oz.au

‡CSIRO Division of Information Technology, GPO Box 664, Canberra, ACT 2601, Australia, Email: zxf@cbr.dit.csiro.au

0-7803-2018-2/95/$4.00 © 1995 IEEE

than that of R_i. Thus, the problem in processing distributed queries by a semi-join reduction approach is to determine a most favourable sem-join reduction program for reducing transmission cost.

A number of significant results in processing distributed queries by a semi-join reduction approach have been achieved [3,10,11]. Most of them assume that local costs are entirely negligible, and then favor a sequential semi-join reduction, that is, using the result of a semi-join to further reduce the size of a relation by another semi-join.

Recent performance studies [9] show that when local processing expense is not negligible, a sequential semi-join reduction approach may sometimes lead to inefficiencies, such as *loss of parallelism, processing overhead, loss of global semi-join optimization, and inaccurate semi-join reduction estimation.*

The *one-shot* semi-join approach was proposed in [10] to reduce the semi-join processing cost and to explore inter-operation parallelism in semi-join executions. It suggests executing a set of semi-joins on the same relation at the same time. That is, if there is more than one relation required to perform semi-join with the same relation R_i, then R_i is scanned once only to perform these semi-joins together. It encourages the semi-joins to be performed on different relations in parallel. The one-shot semi-join reduction approach was then applied to reducing the response time for distributed query processing by considering data transmission time together with local processing time. As in most prior work, the authors made the assumptions that in the presence of redundant data placement, 1) a pre-selected copy of each relation referenced by a query should be determined before producing a semi-join reduction program, and 2) all those pre-selected copies are located at different sites. Further, they considered only a special form of the estimations of semi-join reduction effects and local join costs (see Section 2).

In this paper, we shall extend the approach in [10] in three ways:

1. remove the requirement for copy selection before the production of a semi-join reduction program, and make a copy selection while producing a semi-join reduction program,
2. allow the choice of redundant copies for the execution of semi-joins,
3. employ a general cost model which covers a large class of possible estimations of semi-join reduction effects and local join costs.

These three extensions are necessarily. Note that data replication is desirable in distributed database systems. The necessities of considering aspects 1 and 2 are illustrated as follows.

Example 1. Suppose that a given network consists of 3 sites 1, 2, 3. Relations R_1, R_2, and R_3 are located at site 1, relations R_1, R_4, R_5 are located at site 2. A join $\bowtie_{i=1}^{i=5} R_i$ is required at site 3. Further suppose that R_1 has join attributes with R_2, R_3, R_4, and R_5. It is obvious that the choice of the copy of R_1 from site 1 will favor the semi-joins on R_1 with R_2 and R_3 (i.e., no transmission cost is required for these semi-joins), while the choice of the copy of R_1 from site 2 will favor the semi-joins on R_1 with R_4 and R_5. The different semi-join reduction programs based on different choices of a copy of R_1 may be obtained, which have the different "optimal" values using the algorithm in [10]. Further, if we use redundant copies of R_1 in the execution of semi-joins, there is no transmission cost for processing the semi-joins respectively on R_2, R_3, R_4, and R_5 with R_1. □

The significance of considering aspect 3 should be clear, since different join methods have different cost functions.

In regard of the above three aditional aspects, in this paper we shall present a polynomial time bounded algorithm, which can output an optimal semi-join reduction program for processing a distributed query such that the response time, with respect to our general cost model, will be minimized.

The rest of the paper is organized as follows. Section 2 presents a modification, with respect to the above three aspects, of the one-shot semi-join approach, and presents our cost model. In Section 3, we present our optimal algorithm together with the correctness proof. This is followed by conclusions and remarks.

2 One-shot Semi-join vs Cost Model

2.1 One-shot semi-join

Let $\bowtie_{i=1}^{m} R_i$ be a multiple join to be processed. A one-shot semi-join reduction approach consists of the following five phases [10].

1. *Initial local processing*: All selections and projections are processed in parallel at each site.

2. *Projection Phase*: All referenced relations are scanned at most once in parallel to generate all the necessary join attributes by projections. All these results are hashed at the time they are generated.

3. *Transmission Phase*: The results of the projection phase are transmitted in parallel to the corresponding sites for the execution of semi-joins.

4. *Reduction Phase*: Scan each relation R_i to process all these semi-joins. Since all the results of the projection phase are hashed, each tuple in R_i needs only to be scanned once to check by hashing.

5. *Final Join Phase*: Send all semi-join reduced relations to the result site to perform the join.

Obviously, in the case that the multiple copies of a relation exist, only one of them needs to be semi-join reduced and then sent to the result site to perform the final join. This can save the total transmission cost, and avoid any extra work in the final join phase. Meanwhile, redundant copies of the join attributes of a relation may be chosen to execute semi-joins on other relations, in order to save transmission cost of data shipping.

2.2 Cost Model

Our goal is to minimize the distributed query response time with respect to the above five phases. Particularly, we concentrate on minimizing the response time after the initial processing and projection phases, since the costs of another three phases - data transmission, join and semi-join - are much higher than that of the first two phases.

A precise cost model of distributed query response time in the above last three phases may include the considerations of job scheduling at a site (in case there are several referred relations located at the same site) and transmission contention. Complete optimization with respect to each of those problems is known to be NP-hard [4,11]. Thus, empirical cost models, which exclude some negligible factors to achieve a good approximation of a precise cost model, should be adopted to avoid computing a computationally intractable problem.

We consider, as follows, both communication and computation costs in our cost model. Note that in our discussion, a given network is an arbitrary one.

In the transmission phase, the referenced relations at each site are scanned (at most once) in parallel to generate all columns required by a semi-join reduction program. The values in these columns are hashed, and then sent in parallel to other sites, where the data are used to perform a semi-join reduction. We assume that the transmission delay due to network contention is negligible (this is possible in the environment where a network is connected through a high bandwidth, and multiple channels are available). We use $t_{i,k}^{j,l}$ to denote the time for the transmission of join attributes from site l where R_j is located, for processing the semi-join $R_i \ltimes R_j$, to site k where R_i is located.

The semi-join reduction phase cannot start until all join attributes arrive. As all arrived values are hashed already, the local relations which are to be reduced need to be scanned only once to perform semi-joins by hashing. When there is only one table at each site, say R_i, the time of this phase is dominated by the time for scanning the whole table of R_i, as pointed out in [10]. Thus, the time of semi-join reduction on relation R_i at site k is estimated by the time taken to scan R_i at site k, while other factors, such as hashing and probing, are negligible. This time is represented as $a_{i,k}$, which depends on the size of R_i and the computing facilities at site k.

When there is more than one referenced relation located at the same site, a delay on the response time of the semi-join reduction phase may happen due to a possible resource contention. Assume that this possible resource contention is handled by a simple first-in-first-serve policy, that is, a semi-join can be immediately executed once all the join attributes arrive. In our cost model, we employ the following resource contention situation:

All scanning programs at a site start at the same time concurrently.

Then we use the response time for scanning a relation R_i at site k, in this resource contention situation, to be the response time of an implementation of the one-shot semi-join reduction on R_i at site k. Therefore, the response time of an implementation of the one-shot semi-join reduction on R_i at site k can also be expressed as $a_{i,k}$ - a constant if R_i and site k are given. Note that this cost estimation can be regarded as the worst case with respect to a resource contention. The reasons for choosing this estimation are as follows.

- It is possible that a set of semi-joins on a set of referenced relations run together to cause a resource contention situation. A precise estimation of a response time with respect to an arbitrary resource contention situation is difficult. If such an estimation is done each time before a set of semi-joins is executed, too much processing cost may be incurred.
- This employed resource contention situation is possible.
- The semi-join cost is practically not a dominant factor in the one-shot semi-join approach for processing a distributed query. Also, in most cases, the number of referenced relations in a distributed join may be bounded by a small integer. As well, a suitable database placement, with respect to the storage capacity at each site, will prevent the number of relations located at a single site from being too large [1,6]. Thus, it is unwise for us to focus on this to cause the whole optimization problem computationally intractable.

Let $N_{i,k}$ denote the number of tuples left in R_i after an implementation of the one-shot semi-join

reduction on R_i at site k. The transmission time of the reduced R_i from site k to the result site can be expressed as $b_{i,k}N_{i,k} + c_k$, where c_k is the set up time, $b_{i,k}$ is the transmission time of one tuple of R_i from site k to the result site.

Note that in general, the above transmission time should be expressed as $b_{i,k,l}N_{i,k} + c_{k,l}$, where l represents a result site l. However, for a given multiple join with a given result site l, the above abbreviation $b_{i,k}N_{i,k} + c_k$ will not destroy the generality of the problem.

In the final join phase, semi-join reduced relations are sent to the result site to perform the join at that site. Clearly, the cost in this phase depends on many system factors, such as I/O bandwidth, memory size and local join algorithms. In general, the cost of performing a given final join $\bowtie_{i=1}^{m} R_i$ at the result site can be expected to be measured by the sizes of the semi-join reduced relations in average case. That is:

$$g(N_{1,k_1}, N_{2,k_2}, ..., N_{m,k_m}), \qquad (1)$$

where each N_{i,k_i} represents the number of tuples left in each R_i, and $g(x_1, x_2, ...x_m)$ is a *non-decreasing function* [8] with respect to each variable x_i for $1 \leq i \leq m$. Note that generally, g should also depend on the tuple size of each relation and the computing facilities at the result site, but the above abbreviation (1) will not destroy the generality of the problem for a given join and a given result site.

Note that (1) is only an approximate estimation for local join cost. A precise estimation for local join cost may require exponential size of information, such as *join selectivity* with respect to each sub-join. This will result in too much processing cost to handle the information. Thus, we use the above cost model with respect to an average case.

When applied on a specific system, a particular form of g can be given by the system administrator. After a specific g is given in a specific system, a tailored optimal solution can be derived immediately from our algorithm. A worst case based cost model with respect to the sizes of the reduced relations, which is a special case of (1), is used in [10]. This model suggests that the time needed for processing the join is proportional to the product of the relation cardinalities, that is,

$$C \prod_{i=1}^{i=m} N_{i,k_i} \qquad (2)$$

where k_i is the site at which R_i is sent to the result site. Such a loose estimation may potentially degrade the quality of a produced execution plan for a distributed query process.

Suppose that a join $\bowtie_{i=1}^{i=m} R_i$ is required at a site. Let r_i^j denote the attribute set from R_j for the execution of $R_i \ltimes R_j$, and $0 \leq \rho_i^j \leq 1$ denote the selectivity of $R_i \ltimes R_j$, that is, after the execution of $R_i \ltimes R_j$, the number of tuples left in R_i is $\rho_i^j |R_i|$ where $|R_i|$ denotes the number of tuples in R_i. For $1 \leq i \leq m$, $s_2(i) = \{j : 1 \leq j \leq m, j \neq i, r_i^j \neq \emptyset\}$.

A precise estimation of the size of a join (including semi-join) is usually difficult to be obtained. The research in this area still attracts a great attention [5]. In order to ensure that our algorithm may be applied to any reasonable estimation of a semi-join reduction, in this paper we give a general description of a semi-join reduction, and then provide a generally optimal algorithm with respect to this general description.

A *semi-join reduction function* ρ is a mapping from 2^X to the set of relational numbers which are in the interval $[0, 1]$, where X is a set and 2^X denotes the set of all the subsets of X, such that:

- $\rho(\emptyset) = 1$.
- $\rho(X_1) \leq \rho(X_2)$ when $X_1 \supseteq X_2$

Since more semi-joins on a relation will never increase the size of the left relation, a semi-join reduction function can be used to generally describe a possible estimation of a semi-join reduction effect:

> For $1 \leq i \leq m$, let ρ_i be a *semi-join* reduction function with its variable domain $2^{s_2(i)}$. After the execution of a set of semi-joins $\{R_i \ltimes R_j : j \in \bar{s}_2(i)\}$, the number of tuples left in R_i is then given by $\rho_i(\bar{s}_2(i))|R_i|$, where $\bar{s}_2(i)$ is a subset of $s_2(i)$. (Note that $\rho_i(\{j\})$ is the selectivity of $R_i \ltimes R_j$.)

Previously, the independency of semi-join reduction effects was employed [10], that is, after the execution of a set of semi-joins $\{R_i \ltimes R_j : j \in \bar{s}_2(i)\}$, the number of the tuples left in R_i is $\prod_{j \in \bar{s}_2(i)} \rho_i^j |R_i|$. This is covered by our general description. The advantage of using our general description is:

> Once a more precise estimation (covered by our general description) appears, say not necessarily only an estimation based on independent reduction effects, then it can be immediately employed in our algorithm.

For $1 \leq i \leq m$, $s_1(i)$ denotes the sites where a copy of R_i is located. Let S denote the Cartesian Product on all $s_1(i)$, that is, $S = s_1(1) \times s_1(2) \times ... \times s_1(m)$. The response time of a processing $\bowtie_{i=1}^{i=m} R_i$ by one-shot semi-join reduction approach can be represented by:

$$\max_{1 \leq i \leq m} \left(\max_{j \in s_2(i)} \{t_{i,k_i}^{j,l_j}\} + a_{i,k_i} b_{i,k_i} \rho_i(s_2(i))|R_i| + c_{k_i} \right) + g(\rho_1(s_2(1))|R_1|, \rho_2(s_2(2))|R_2|, ..., \rho_m(s_2(m))|R_m|) \quad (3)$$

for $(k_1, k_2, ..., k_m) \in S$, $\bar{s}_2(i) \in 2^{s_2(i)}$ for each i, and $l_j \in s_1(j)$ for each $j \in \bar{s}_2(i)$. In the next section, we show how to determine an execution plan to minimize the response time.

3 Algorithm

From the discussions in the last section, it follows that an execution plan of a distributed query $\bowtie_{i=1}^{i=m} R_i$ corresponds to a combination of the following three selections:

Sel 1: a selection of a site k_i from $s_1(i)$ for each referenced relation R_i,

Sel 2: a selection of a subset $\bar{s}_2(i)$ of $s_2(i)$ with respect to a given copy of R_i which is located at site k_i,

Sel 3: for each $j \in \bar{s}_2(i)$, a selection of a site $l_j \in s_1(j)$ with respect to a given site k_i where R_i is located

Sel 1 corresponds to a selection of a copy for each relation referenced by the multiple join. The selected copies in Sel 1 will possibly be first sem-join reduced, and then will be sent to the result site for the execution of the join. A choice of a set of semi-joins on a given copy of a relation is stated in Sel 2. To execute a semi-join $R_i \ltimes R_j$ at site k_i, Sel 3 will provide a choice of a copy of the join attribute set (between R_j and R_i) of R_j (if there are several copies of R_j), which is required to be sent to site k_i. Note that redundant copies of a R_j may be chosen in Sel 3 for the production of join attribute sets with respect to the execution of several semi-joins at different sites (see Example 1).

A trivial exhaustive search on the search space, as described above for the minimization of the value of (3), is not feasible, because the search space usually has an exponential size. Thus, we first present a reduction on the search space such that the reduced search space is polynomially bounded.

3.1 Reduction on Search Space

In this subsection, we present a reduction on the search space from Sel 3 - Sel 1. Obviously, we need only consider those semi-joins $R_i \ltimes R_j$ whose selectivities ρ_i^j are not 1. Thus in our later discussions, we restricted our interests in a selection of $\bar{s}_2(i)$ from the following set for each i:

$$\{j : 1 \leq j \leq m, j \neq i, r_i^j \neq \emptyset, \rho_i^j < 1\}.$$

Hereby, we re-define $s_2(i)$ as the above set.

First, one can immediately verify the following Lemma, which shows that Sel 3 can be solved easily. A t_{i,k_i}^{j,l_j} in the minimum value of (3) has a restriction on the choice of l_j:

Lemma 1 *Suppose that R_i, k_i, and R_j are given. To minimize (3), l_j should be a site in $s_1(j)$, such that t_{i,k_i}^{j,l_j} is minimized.*

This means that if r_i^j in R_j is required by the site k_i to process the semi-join $R_i \ltimes R_j$ at k_i, only such a site l_j in $s_1(j)$ that the value of t_{i,k_i}^{j,l_j} is minimized shall be considered in the optimal solution for minimizing (3). In case there are several such sites, we just randomly choose one. We use $f(i,k_i,j)$ to denote such a site. Clearly, $f(i,k_i,j)$ may be determined in time $O(n_j)$ for given i,k_i,j where n_j is the cardinality of $s_1(j)$.

In the optimal execution plan for minimizing (3), a choice of each $\bar{s}_2(i)$ may be limited as follows for a given i and its associated site $k_i \in s_1(i)$. Let t_{i,k_i}^j denote $t_{i,k_i}^{j,f(i,k_i,j)}$. We can show that the optimal execution plan must have the following property.

Lemma 2 *Let $R_i \ltimes R_j$ be executed in an execution plan P which has the minimum value of (3) for processing the join. Then this execution plan either*

- *includes the implementation of the semi-joins on R_i with those $R_{\bar{j}}$ for $t_{i,k_i}^{\bar{j}} \leq t_{i,k_i}^j$ and $\bar{j} \in s_2(i)$, or*
- *can be extended to another execution plan $\bar{P}$ by including the semi-joins on R_i with those $R_{\bar{j}}$ for $t_{i,k_i}^{\bar{j}} \leq t_{i,k_i}^j$ and $\bar{j} \in s_2(i)$ such that the response time for $\bar{P}$ is the same as that for P.*

Proof: Note that in our cost model, the response time for an implementation of the one-shot semi-join reduction on R_i is expressed approximately by the scanning time of R_i, no matter how many join attributes from the other relations are involved. Meanwhile, the execution of more semi-joins on R_i will never lead to an increase in the number of tuples left in R_i. From these and together with the response time expression by (3), the Lemma follows immediately . □

Given R_i and $k_i \in s_1(i)$, $\{t_{i,k_i}^j : j \in s_2(i)\}$ can be organized into the non-decreasing ordered set $D_{i,k_i} = \{d_{i,k_i}(p) : 1 \leq p \leq |s_2(i)|\}$, such that, say $t_{i,k_i}^{j_p} = d_{i,k_i}(p)$ and $t_{i,k_i}^{j_q} = d_{i,k_i}(q)$, $t_{i,k_i}^{j_p} \leq t_{i,k_i}^{j_q}$ if $p \leq q$. Let:

$$sm_{i,k_i}(p) = \{j : t_{i,k_i}^j \leq d_{i,k_i}(p)\} \text{ for } 1 \leq p \leq m \qquad (4)$$

$sm_{i,k_i}(0) = \emptyset$. From Lemma 2, it implies that in our later discussion, we need only to consider one of $sm_{i,k_i}(p)$ for $0 \leq p \leq |s_2(i)|$ as a choice of $\bar{s}_2(i)$ from $s_2(i)$, in our construction of an optimal execution plan. Note that $sm_{i,k_i}(p) \subseteq sm_{i,k_i}(q)$ if $p \leq q$.

From Lemmas 1 and 2, the problem in the determination of an execution plan with the minimum response time, has been reduced to finding a k_i in $s_1(i)$ for $1 \leq i \leq m$, and a corresponding $sm_{i,k_i}(p_i)$ for $0 \leq p_i \leq |s_2(i)|$ to minimize:

$$\max_{1 \leq i \leq m} \{ \max_{j \in sm_{i,k_i}(p_i)} (t_{i,k_i}^j) + a_{i,k_i} + b_{i,k_i} \rho_i(sm_{i,k_i}(p_i))|R_i| + c_{k_i}) + g(\rho_1(sm_{1,k_1}(p_1))|R_1|, \ldots, \rho_m(sm_{m,k_m}(p_m))|R_m|) \qquad (5)$$

To further reduce the search space, we first introduce the following notation. Let (x_1, y_1) and (x_2, y_2) be two points in a 2-dimensional space. We say that (x_1, y_1) *dominates* (x_2, y_2) if $x_1 \geq x_2$ and $y_1 \geq y_2$. Suppose that for $1 \leq i \leq m$ and $0 \leq p_i \leq |s_2(i)|$,

$$h(sm_{i,k_i}(p_i)) = \{ \max_{j \in sm_{i,k_i}(p_i)} \{t^j_{i,k_i}\} + a_{i,k_i} + b_{i,k_i} \rho_i(sm_{i,k_i}(p_i))|R_i| + c_{k_i} \}.$$

and

$$v(sm_{i,k_i}(p_i)) = \rho_i(sm_{i,k_i}(p_i))|R_i|.$$

From the fact that g is nondecreasing with respect to each variable, we can verify the following Lemmas.

Lemma 3 *Suppose that given a R_i, and $k_i, \bar{k}_i \in s_1(i)$ where k_i and $\bar{k}_i$ are not necessarily equivalent, $(h(sm_{i,k_i}(p_i)), v(sm_{i,k_i}(p_i)))$ dominates $(h(sm_{i,\bar{k}_i}(\bar{p}_i)), v(sm_{i,\bar{k}_i}(\bar{p}_i)))$. Let $sm_{i,k_i}(p_i)$ be selected in an execution plan P. Then the execution plan $\bar{P}$, obtained by the replacement of $sm_{i,k_i}(p_i)$ in P with $sm_{i,\bar{k}_i}(\bar{p}_i)$, leads to the response time smaller than (or equal to) the response time by using P.*

Proof: By (5), the fact that g is nondecreasing, and the property of a semi-join reduction function, one may immediately verify this Lemma. □

Lemma 4 *Let an execution plan P contain $sm_{i,k_i}(p_i)$ for $1 \leq i \leq m$, and T denote $max_{1 \leq i \leq m}\{h(sm_{i,k_i}(p_i))\}$. Further suppose that there exists a $sm_{l,\bar{k}_l}(\bar{p}_l)$ for a relation R_l where $1 \leq l \leq m$ and for a site $\bar{k}_l$ in $s_1(l)$, such that $h(sm_{l,\bar{k}_l}(\bar{p}_l)) \leq T$ and $v(sm_{l,\bar{k}_l}(\bar{p}_l)) \leq v(sm_{l,k_l}(p_l))$. Then we may replace $sm_{l,k_l}(p_l)$ in P by $sm_{l,\bar{k}_l}(\bar{p}_l)$ to obtain another execution plan $\bar{P}$ such that the value of (5) with respect to $\bar{P}$ is not greater than that with respect to P.*

Proof: Again, one may immediately verify this Lemma by (5) and the fact that g is nondecreasing. □

Suppose that for $1 \leq i \leq m$ and $k_i \in s_1(i)$, all $sm_{i,k_i}(p_i)$ have been computed. Then we can use the following procedure PROC to determine an optimal execution plan according to Lemma 3 and Lemma 4.

PROC

Proc 1: For each i with $1 \leq i \leq m$, the elements in the following set

$$\{(h(sm_{i,k_i}(p_i)), v(sm_{i,k_i}(p_i))) : k_i \in s_1(i), 0 \leq p_i \leq |s_2(i)|\}$$

can be viewed as points in two dimensional space, and are organized into the nondecreasing ordered set $S(i)$ with respect to first coordinate. Then each $S(i)$, for $1 \leq i \leq m$, is scanned once to eliminate those elements in $S(i)$ which dominate at least another element in $S(i)$. Go to Proc 2. (Note the outputs of Proc 1 are nondecreasing ordered sets $S(i)$ for $1 \leq i \leq m$, such that in each $S(i)$, there is no element dominating another element.)

Proc 2: Suppose that after Proc 1, for each i with $1 \leq i \leq m$, $S(i)$ is represented by $\{(u_i(p), w_i(p)) : 1 \leq p \leq |S(i)|\}$, where each $(u_i(p), w_i(p)) = (h(sm_{i,k_i}(p_i)), v(sm_{i,k_i}(p_i)))$ for some $sm_{i,k_i}(p_i)$, and is linked to $sm_{i,k_i}(p_i)$, R_i and k_i. Here p denotes the ordering, that is, in each $S(i)$, $u_i(p) < u_i(q)$ if $p < q$. (Note $w_i(p) > w_i(q)$ if $p < q$, since no element in $S(i)$ dominates another.) Merge all $S(i)$ to a nondecreasing ordered set A with respect to the first coordinate. A is represented by

$$\{((u_i(p)(x), w_i(p)(x)) : \text{ for } 1 \leq i \leq m, (u_i(p), w_i(p)) \in S(i)\}.$$

Here x denotes the ordering, that is, for $1 \leq x \leq |A|$, $u_i(p)(x) \leq u_j(q)(y)$ if $x < y$ where i and j are not necessarily different, and p and q are not necessarily different for a pair of distinguished i and j. The link of each element $(u_i(p)(x), v_i(p)(x))$ in A is equal to the link of $(u_i(p), v_i(p))$ in $S(i)$ as described above. Go to Proc 3.

Proc 3: Let $C = \infty$, and for $1 \leq i \leq m$, let $u_i = \infty, w_i = \infty, temp1_i = \infty, temp2_i = \infty$. For $x = 1$ to $|A|$ do (scan each element in A):

- output the subscript (j) of $(u_j(p)(x), v_j(p)(x))$;
- $temp1_j := u_j(p)(x)$; $temp2_j := v_j(p)(x)$; (only re-value the corresponding $(temp1_j, temp2_j)$.)
- $C_1 = max_{1 \leq i \leq m}\{temp1_i\} + g(temp2_1, temp2_2, ..., temp2_m)$;
- if $C_1 < C$ then replace C with C_1, and replace $\{(u_i, w_i) : 1 \leq i \leq m\}$ with $\{(temp1_i, temp2_i) : 1 \leq i \leq m\}$.

Go to Proc 4.

Proc 4: output $P = \{(u_i, w_i) : 1 \leq i \leq m\}$.

Note that we assume $g = \infty$ if one of its variable is ∞. After procedure PROC, we can determine a $sem_{i,k_i}(p_i)$ for each (u_i, w_i) from the link established in Proc 2. We, then, obtain an execution plan $\bar{P}$.

3.2 Description of Algorithm

In this subsection, we present a detailed description of the algorithm OPT. The algorithm OPT outputs an execution plan for a distributed join process by the one-shot semi-join approach with a minimum response time (as presented in (3)). It consists of the following 5 steps:

Algorith OPT

Step 1: For each i, each $k_i \in s_1(i)$ and each $j \in s_2(i)$, find $f(i, k_i, j)$. Let $t^j_{i,k_i} = t^{j,f(i,k_i,j)}_{i,k_i}$. Go to Step 2.

Step 2: For each i and each $k_i \in s_1(i)$, compute all $sm_{i,k_i}(p)$ for $0 \leq p \leq |s_2(i)|$. Go to Step 3.

Step 3: Compute each

$$(h(sm_{i,k_i}(p_i)), v(sm_{i,k_i}(p_i))),$$

and link

$$(h(sm_{i,k_i}(p_i)), v(sm_{i,k_i}(p_i)))$$

to $sm_{i,k_i}(p_i)$. Go to Step 4.

Step 4: Execute the procedure PROC as described in the last subsection. Go to Step 5.

Step 5: Track back through the links on the output of PROC to obtain an execution plan.

The correctness of the algorithm OPT can be obtained from Lemmas 1, 2, 3, and 4.

Theorem 1 *The algorithm OPT produces an execution plan for a distributed join such that the execution plan minimize the response time as expressed in (3).*

Proof: One may immediately verify this Theorem by Lemmas 1, 2, 3, and 4. □

3.3 Time Complexity of the Algorithm

Suppose that for $1 \leq i \leq m$, $|s_1(i)| = n_i$, $|s_2(i)| = m_i$, $M = max_{1 \leq i \leq m}\{m_i\}$, and $N =$

$max_{1 \le i \le m}\{n_i\}$. We assume that it takes constant time to compute the value of g and each ρ_i. In the algorithm OPT, Step 1 takes $O(mMN^2)$. In Step 2, sorting is the dominant cost. Totally, Step 2 takes $O(mNMlogM)$. Step 3 takes $O(mMN)$ time. In Step 4, sorting is also the dominant cost. Thus, Step 4 takes $O(mNM(logm + logN + logM)) = O(mNM(logm + logN))$. Step 5 takes $O(m)$ time. Hence the time complexity of the algorithm OPT is $O(mMN(N + logm))$. The overall space for the algorithm OPT takes $O(nMN^2)$ space. Note that in case that there is no redundant data placement, the complexity of the algorithm is the same as that in [10].

4 Remarks and Conclusions

In this paper, the approach in [10] has been extended to cover three additional aspects.

As we pointed out in Section 2, the response time minimization problem will be computationally intractable if we apply a precise cost model. This prevents us using a precise cost model, since spending exponential time to produce an execution plan for a join process in order to minimize response time is useless. The local processing cost model presented in this paper is an approximation model by the deletion of some non-dominant factors. Thus, we could expect that the optimization solution for the cost model presented in this paper is, in practice, a nearly optimal result to an NP-hard problem.

In the future, we would like to ease the restrictions in using a worst case based estimation model for a resource contention situation at the semi-join phase; that is, we would like to find a better way to estimate the response time while several semijoins run together to share a resource.

References

[1] Y. Bartal, A. Fiat and Y. Rabani, "Competitive Algorithms for Distributed Data Management", *24th Annual ACM Symposium on the Theory of Computing*, pp. 39-49, 1992.

[2] P. A. Bernstein and D. Chiu, "Using Semi-Joins to Solve Relational Queries", *Journal of ACM*, 28(1), 25-40, 1981.

[3] M.-S. Chen and P. S. Yu, "Interleaving a Join Sequence with Semijoins in Distributed Query Processing", *IEEE Transactions on Parallel and Distributed Systems*, 3(5), pp. 611-621, 1992.

[4] M. R. Garey and D. S. Johnson, *Computers and Intratability: a guide to the theory of NP-Completeness*, W. H. Freeman and Company, 1978.

[5] Y. E. Ioannidis and S. Christodoulakis, "On the Propagation of Errors in the Size of Join Results", *Proceedings of the 1991 SIGMOD International Conference on Management of Data*, pp. 268-277, 1991.

[6] X. Lin and M. Orlowska, "An Integer Linear Programming Approach to Data Allocation with the Minimum Total Communication Cost in Distributed Database Systems", to appear in *Information Sciences* (in press), 1995.

[7] D. Shasha and T. L. Wang, "Optimizing Equijoin Queries in Distributed Databases Where Relations are Hash Partitioned", *ACM Transactions on Database Systems*, 16(2), pp. 279-308, 1991.

[8] A. E. Taylor, *Advanced Calculus* , Ginn, 1955.

[9] M. Templeton, et al., "Mermaid-Experiences with network operation", *Proceedings of IEEE Data Engineering Conference*, 1986.

[10] C. P. Wang, A. L. P. Chen and S.-C. Shyu, "A Parallel Execution Method for Minimizing Distributed Query Response Time", *IEEE Transactions on Parallel and Distributed Systems*, 3(3), 325-333, 1992.

[11] C. T. Yu and C. C. Chang "Distributed Query Processing", *ACM Computing Surveys*, 16(4), 1984.