Distributed and Parallel Databases, 3, 69–83 (1995)

An Efficient Processing of a Chain Join with the Minimum Communication Cost in Distributed Database Systems

XUEMIN LIN* AND MARIA E. ORLOWSKA {LXUE,MARIA}@CS.UQ.OZ.AU
Department of Computer Science, The University of Queensland, QLD 4072, Australia

Received June 17, 1993; Revised June 30, 1994

Editor: Peter Apers

Abstract. This paper investigates the optimization problem when executing a join in a distributed database environment. The minimization of the communication cost for sending data through links has been adopted as an optimization criterion. We explore in this paper the approach of judiciously using join operations as reducers in distributed query processing. In general, this problem is computationally intractable. A restriction of the execution of a join in a pre-defined combinatorial order leads to a possible solution in polynomial time. An algorithm for a chain query computation has been proposed in [21]. The time complexity of the algorithm is $O(m^2n^2 + m^3n)$, where n is the number of sites in the network, and m is the number of relations (fragments) involved in the join. In this paper, we firstly present a proof of the intuitively well understood fact—that the "eigenorder" of a "chain" join will be the best pre-defined combinatorial order to implement the algorithm in [21]. Secondly, we show a sufficient and necessary condition for a chain query with the eigenordering to be a "simple" query. For the process of the class of simple queries, we show a significant reduction of the time complexity from $O(m^2n^2 + m^3n)$ to $O(mn + m^2)$. It is encouraging that, in practice, the most frequent queries belong to the category of simple queries.

Keywords: Distributed databases, query processing optimization, communication cost

1. Introduction

Distributed query optimization has been studied for various environments, but the most significant theoretical results have been achieved in purely relational systems.

Obviously, any outcome of applying a method to query execution depends on distributed database design or more specifically on data fragmentation and its allocation to the network. When fragmentation and allocation are decided, one may apply different strategies for the execution of queries as well as the propagation of updates to achieve overall system performance. Clearly, there are complex dependencies between data allocation and the ways in which the system validate queries and updates.

In this paper, we consider only some aspects of this complex structure. We concentrate on distributed query processing to achieve the minimum overall system communication cost. (We do not consider the system's response time for an individual mode.)

By communication cost we mean the cost of data shipping between different individual computers.

*Current address: Department of Computer Science, The University of Western Australia, WA 6009, Australia.

Traditionally, the approach of using semi-joins as reducers to process a distributed query have received a great deal of attention. As pointed out in [32], this approach consists of the following three phases: (1) a *local processing phase* which involves all local processing such as selections and projections, (2) a *semijoin reduction phase* where a sequence of semijoins is used to reduce the size of relations, and then, lessen the total communication cost required, and (3) a *final processing phase* in which all resulting relations are sent to the *result* site where the final query processing is performed. The problem of minimizing the communication cost for distributed query processing by a semi-join reduction approach has been shown as NP-hard [13, 28]. In the meantime, this problem, restricted to a "chain join", can be solved in polynomial time in the environment where only *uniform* networks—networks in which the communication between each pair of sites is the same—are considered. Numerous algorithms and heuristics are proposed to solve this problem [1, 3, 4, 6, 7, 13, 18, 32, 33].

Another approach in processing a distributed query is based on using join operations as reducers. We call it a *join based reduction approach* (a formal description may be found in Section 2). A join based reduction approach in processing distributed queries to minimize the communication cost, after a semijoin reduction, is investigated in [8]. The authors proposed an application of a join based reduction approach in the final processing phase, as described above, to further reduce communication (transmission) cost through applying join operations in sending relations (possibly semijoin reduced) to the result site. Moreover, in observing the inherent computational intractability of the optimization problem, they provide an efficient heuristic for a case where only uniform networks are considered, but general queries are employed. Further, [21] points out that this optimization problem, restricted to a case in which chain queries and general networks are considered, can be solved in polynomial time. The claimed time complexity of the optimal algorithm in [21] is $O(m^2n^2 + m^3n)$. [21] suggests directly applying join operations as reducers without implementing a semijoin reduction phase, but with only a local processing phase such as projections and selections, in view of the fact that semijoins will potentially increase local processing cost. The optimization problem—minimizing the communication cost—for distributed query processing by applying a join based reduction approach is essentially the same, in spite of whether or not a semijoin reduction has been applied previously. So the algorithm in [21] can be immediately applied to the environment in [8].

In this paper, we investigate a join based reduction approach for processing a distributed query. We are motivated by two factors as follows:

1. Local processing costs can be significant [32], while additional local operations may be generated when semijoins are employed.

2. If it is imperative to minimize the communication cost, then judiciously applying join operations as reducers may further reduce the communication cost [8], in addition to semijoins.

We adopt the same environment as that in [21]. We focus specifically on the execution of chain queries in a distributed relational database, assuming that algebraic modifications, such as selections and projections, have been applied prior to the join computation. We have placed our study in a static environment. This means that the network of individual

machines is fixed and its topology, including the cost of data unit transmission through every link, is fixed and expressed as a constant. We assume that data for the period of our execution of the database requests, have been allocated to the individual sites. Then we show that for a large class (category) of queries, the time complexity $O(m^2n^2 + m^3n)$ in [21] can be significantly reduced. We propose a pre-processing phase to the algorithm in [21], which can be executed in linear time. The aim of this additional computation is to classify a multi-join, without a size reduction in any intermediate result, to the specified category called "simple" queries (see Section 2 for the definition). This initial procedure is based on a sufficient and necessary condition of a join with a pre-defined combinatorial order to belong to the category of simple queries. This initial procedure, together with the "simple" strategy (see Section 2 for the definition) to process a simple query, leads to a reduction of the time complexity of a join compution, from $O(m^2n^2 + m^3n)$ to $O(mn + m^2)$. Applied to the environment in [8], it also leads to computing an optimal strategy for processing a simple query in $O(m^2)$ (because only uniform networks are considered there).

Additionally, we present some considerations of different orderings of relations in a chain query. We prove that the "eigenorder" is optimal independently from strategies appointed for a join based reduction approach.

Most of the research in distributed query processing optimization considers only the communication cost. In a general environment, additional consideration of the local processing cost exacerbates the intractability of the optimization problem. We have reached interesting conclusions regarding this issue, however, this is only applicable to a special case of a distributed environment. It is as follows:

If a simple strategy is applied along with a query processing optimization algorithm (for a uniprocessor environment) in a communication network where each individual processor (site) is identical, then a simple chain query can be processed with the minimal total cost including local processing cost.

The rest of the paper is organized as follows. Section 2 provides basic notation, the problem specification, a formal definition of a join based reduction approach, and an overview of the related works. Section 3 provides evidence of the optimality of the eigenorder of a chain join. It follows that the problem of minimizing the communication cost for a chain join belongs to the polynomial time solvable class of problems, though the general optimization problem is computationally intractable [14, 21]. We then provide a necessary and sufficient condition under which a join, with a pre-defined combinatorial order, is rendered simple (defined in Section 2) in order to achieve the minimum communication cost. In Section 5, we show how to apply our result to the inclusion of local processing cost. Finally, we present conclusions.

2. Preliminaries

This section includes a background discussion with respect to networks and join computation representations. It also provides a brief overview of the algorithm in [21].

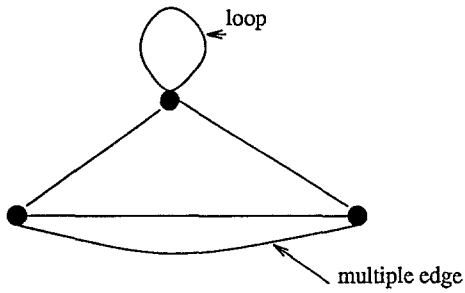

Figure 1. Loop and multiple edge.

2.1. *Network*

Let I denote the set of positive integers. A *network* $N = (V, E, p)$ consists of a connected graph (V, E) without *loops* and *multiple edges* [5] (see Figure 1) and a mapping p: $E \rightarrow I$, where each node in V represents an individual computer (a site of system), each edge represents a link, and p is assigned so that the communication cost of a unit data transmission through a link e is $p(e)$. The communication cost within a node is assumed to be zero. Here $G = (V, E)$ is called the *underlying graph* of N.

In this paper, we consider only the *metric networks* of which the underlying graph is *fully connected* (*complete*) [5], and for any triple u, v, w of distinct nodes:

$$p((u, v)) \leq p((u, w)) + p((w, v)).$$

It is obvious that to minimize the overall communication cost, the data transmission between two sites must be done through the *shortest path* (cheapest path in term of cost) between these two sites (nodes). However, the overall effectiveness of a strategy for sites communication depends heavily on methods adopted for updates propagation, as well as methods for the execution of complex queries.

The transformation of an arbitrary network to a "metric version" is a simple task which always can be accomplished by finding the length of the shortest path for any pair of sites (nodes). Thus, we consider only a metric network.

For the remainder of the paper, "metric network" is abbreviated to "network".

2.2. *A representation of join computation*

Without loss of generality of the optimization problem, and also for simplification of the formal notation, we express queries by the specification of relations involved in the query. The results, presented in this paper, are applicable to any query, but in our formalization we consider only *natural joins* [27].

Further on, by F_i we denote the *relation schema* [27] of a relation f_i.

The intersection of two relations is *empty* if there are no common attributes.

A *join* $f_1 \bowtie f_2 \bowtie f_3 \bowtie \cdots \bowtie f_m$ is a *chain* if there is a *permutation* $p = (i_1, i_2, \ldots, i_m)$ of $(1, 2, \ldots, m)$ such that for $1 \leq j \leq m-1$, $F_{i_j} \cap F_{i_{j+1}} \neq \emptyset$, and for $1 \leq j, k \leq n$ and $|j-k| \geq 2$, $F_{i_j} \cap F_{i_k} = \emptyset$. The permutation $(i_1, i_2, \ldots, i_m)$ is called the *eigenorder* of the chain join. Note that the eigenorder of a chain join is unique up to a reversion.

A *tree* t [5] is a *rooted-tree* if it is organized as follows (see Figure 2(b)):

- a node in t is the root, and
- each node v other than the root has one parent v_p, and v and v_p is connected by an edge in t, and
- there is a pointer type data structure such that each node other than the root is pointed to its parent.

Suppose that u is a node of t. A rooted-subtree t_u of t with the root u is the *induced rooted-subtree of* u *with respect to* t if t_u is the maximal rooted-subtree with the root u in t. For example in Figure 2(b), the rooted-subtree consisting of node 4, $(f_3, 3)$, and $(f_4, 4)$ is the induced rooted-subtree of node 4, while the rooted-subtree consisting of node 4 and $(f_3, 3)$ is not the induced rooted-subtree of 4 with respect to the rooted-tree in Figure 2(b). The induced rooted-subtree of a leaf is a leaf itself.

There are a number of ways to process a multi-join. Consider a join $f_1 \bowtie f_2 \bowtie f_3$. One can process it in three ways:

- firstly, $f_1 \bowtie f_2$ is carried out, and then $(f_1 \bowtie f_2) \bowtie f_3$ is implemented;
- we first implement $f_2 \bowtie f_3$, and then do $f_1 \bowtie (f_2 \bowtie f_3)$;
- $f_1 \bowtie f_3$ can also be first processed, and then $(f_1 \bowtie f_3) \bowtie f_2$ follows.

In a distributed database environment, we also need to consider which copy of a relation (if redundant data exists) should be used to process a query.

Given a multi-join $q = \bowtie_{i=1}^{i=m} f_i$, a *join based reduction approach* suggests using only $m-1$ join operations to get the result, that is, no redundant operations may be allowed. Each possible computation of a join, by a join based reduction approach, may be represented as a rooted-tree t (similar to a query tree [27]), called an *execution-tree*. In an execution-tree t of q, the root represents the join result site of q. Each leaf is represented by (f_i, j) where f_i is a relation and j is a site number where f_i is allocated, and each of the nodes, other than leaves, is represented by an integer i where i represents the i'th site in the network.

In an execution tree, each node other than the leaves represents a site which needs the result of the join of the relations in the induced rooted-subtree. An execution tree t of a join corresponds to a particular strategy of the computation of the join, by a join based reduction approach, as follows. Iteratively from leaves to the root, for each node i, the subresults produced by its children are sent to site i and then joined there. The total communication cost (data transmission cost) of the join computation represented by an execution tree t is called the *cost* of t.

For example in Figure 2, a network is illustrated in (a) where each link e has $p(e) = 1$. Suppose that the relation f_1 is located at site 1 and site 2, the relation f_2 is located at site

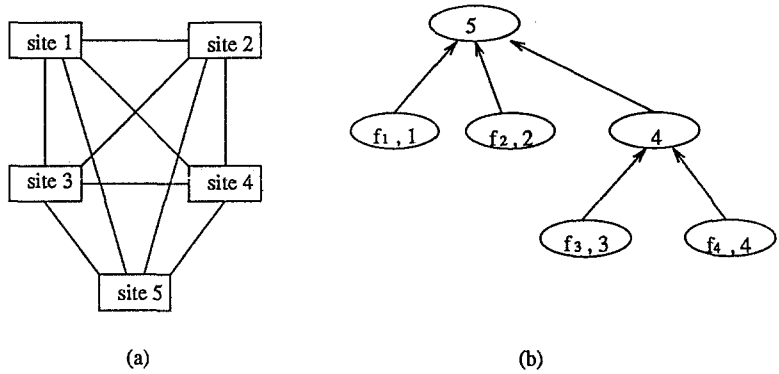

Figure 2. Network and execution tree.

2 and site 4, and the relations f_3 and f_4 are respectively located at site 3 and site 4. An execution tree for the join $f_1 \bowtie f_2 \bowtie f_3 \bowtie f_4$ with result site 5 is illustrated in Figure 2(b). This join is processed according to this tree as follows:

> Relation f_3 located at site 3 is sent to site 4, and is joined with f_4 there. The copy of f_1 located at site 1, the copy of f_2 located at site 2, and the result of join of $f_3 \bowtie f_4$ at site 4 are sent to site 5 to perform the join $f_1 \bowtie f_2 \bowtie f_3 \bowtie f_4$.

Assume that the size of f_1 is 10, the size of f_2 is 10, the size of f_3 is 10, and the size of $f_3 \bowtie f_4$ is 10. The cost of the above tree is 40.

Given a join $f_1 \bowtie f_2 \bowtie \cdots \bowtie f_m$, an *execution order* of this join is a permutation $(i_1, i_2, \ldots, i_m)$ of $(1, 2, \ldots, m)$. An execution tree t *preserves* an execution order $p = (i_1, i_2, \ldots, i_m)$ if:

(2.1) for each node in t and the induced rooted-subtree t' of the node, there are two integers l and L such that the relations stored at the leaf set of t' are f_{i_j} for $l \leq j \leq L$.

A node u in an execution tree t *preserves* p if the induced rooted-subtree of u satisfies (2.1). For example, the execution tree in Figure 2(b) preserves the execution orders $(1, 2, 3, 4)$ and $(2, 1, 4, 3)$, but does not preserve $(1, 3, 2, 4)$.

It is clear that an execution tree preserves p if and only if all its nodes preserve p. Note that each leaf in an execution tree preserves any execution order.

Suppose that q is the join $f_1 \bowtie f_2 \bowtie \cdots \bowtie f_m$. Let the whole execution tree set of q be denoted by T_q, that is, $T_q = \{t: t \text{ is an execution tree of } q\}$. Note that each execution tree of $\bowtie_{i=1}^{i=m} f_i$ consists of only m leaves, because we do not consider additional operations. We note that in some cases, adding additional operations—for example, semi-joins [3]—may further reduce the communication cost. We do not consider these aspects of join computation in this work, since we can assume that profitable semijoins have been done prior, like the environment in [8].

2.3. *An overview of the related works*

It has been shown that the problem of finding an execution tree with the minimum cost from T_q is generally computationally intractable [14, 21]. Suppose that p is an execution order of a join $q = \bowtie_{i=1}^{i=m} f_i$, and T_q^p is a subset of T_q such that:

$$T_q^p = \{t: t \in T_q, t \text{ preserves } p\}.$$

[21] provides a polynomial time bounded algorithm $\theta(m^2n^2 + m^3n)$ (n is the number of sites in the network) for finding an execution tree, with the minimum cost, from T_q^p. For the algorithm in [21], the inputs are:

- a network,
- an execution order $p = (l_1, l_2, \ldots, l_m)$ of a join $q = \bowtie_{i=1}^{i=m} f_i$, and
- a matrix $S_{m\times m}^p = [a_{ij}]_{m\times m}$ such that $a_{ij} = |f_{l_i} \bowtie f_{l_{i+1}} \bowtie \cdots \bowtie f_{l_{i+j-1}}|$ for $1 \le i \le m-1$ and $2 \le j \le m-i+1$ and $a_{i1} = |f_{l_i}|$ for $1 \le i \le m$, otherwise a_{ij} is infinity.

Here each $|f_{l_i} \bowtie f_{l_{i+1}} \bowtie \cdots \bowtie f_{l_{i+j-1}}|$ denotes the data volume of $f_{l_i} \bowtie f_{l_{i+1}} \bowtie \cdots \bowtie f_{l_{i+j-1}}$, and each $|f_{l_i}|$ denotes the data volume of f_{l_i} as well. In the algorithm of [21], a *dynamic programming technique* [9] is used. It, iteratively, computes the best execution tree, preserving p, for processing each $\bowtie_{x=0}^{x=j} f_{l_{i+x}}$ at each site, where $1 \le i, i+j \le m$. That is, before computing the best execution tree of $\bowtie_{x=0}^{x=j} f_{l_{i+x}}$ at each site, one should first compute respectively the best execution tree for processing each $\bowtie_{x=c_1}^{x=c_2} f_{l_{i+x}}$ at each site (where $0 \le c_1, c_2 \le j$ and $c_2 - c_1 < j$), and then combine them to obtain the best execution tree, preserving p, for processing $\bowtie_{x=0}^{x=j} f_{l_{i+x}}$ at each site. The detailed code of this algorithm may be found in [21].

The authors of [8] investigate the problem of finding an execution tree with the minimum cost in T_q for a general join $q = \bowtie_{i=1}^{i=m} f_i$ in uniform networks. Because of the computational intractability of the problem, a heuristic algorithm is proposed in [8] to find an approximate solution to the problem.

Let $minT_q$ and $minT_q^p$ denote the costs of the execution trees with the minimum cost respectively in T_q and in T_q^p.

We, in this paper, first prove that:

$$minT_q = minT_q^p,$$

for the case where q is a chain multi-join and p is the eigenorder of q. This implies that we may use the algorithm in [21] to find a solution for $minT_q$ when q is a chain join.

To present our second and also the main result in this paper, we shall first give some notation.

An execution tree $t \in T_q$ is *normal* if:

(2.2) in t, for each node u and each child v of u, the site corresponding to u should be different to the site corresponding to v, except that v is a leaf.

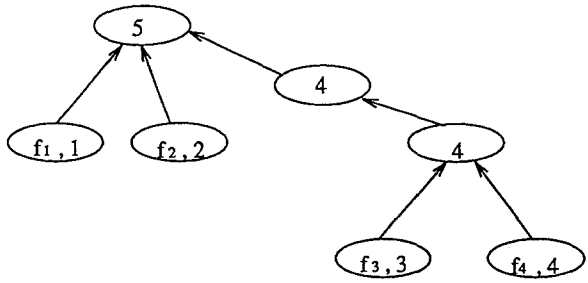

Figure 3. Non-normal execution tree.

For example, the execution tree in Figure 2 is normal, but the one in Figure 3 is not normal. It is clear that for each execution tree t in T_q, there exists a normal execution tree t' such that their costs are the same.

An execution tree is *simple* if it consists only of the root and leaves. Let

$$T_q^s = \{t\colon t \in T_q, t \text{ is simple}\}.$$

Note that for any execution order p of a join q, $T_q^s \subset T_q^p$. Since networks considered here are metric, to find an execution tree in T_q^s with the minimum cost we only need to do the following. If j is the query result site then:

> For each relation f_i involved in the join, find the closest copy of f_i to site j, and then send it to site j to perform the join.

The above computation takes $O(mn)$ time for a general network, and takes $O(m)$ time for a uniform network. Here, by $minT_q^s$ we mean the cost of an execution tree with the minimum cost in T_q^s. Note usually $minT_q^p \leq minT_q^s$. A query q with respect to an execution order p is *simple* if $minT_q^p = minT_q^s$.

Our second result in this paper (presented in Section 4) provides a sufficient and necessary condition for a join q under which, for a given execution order p and any data allocation, the following condition is satisfied:

$$minT_q^p = minT_q^s.$$

It leads to a significant reduction of the time complexity to produce a solution to $minT_q$ for a large class of queries q—chain queries.

3. Optimality of the eigenorder

In this section, we prove the following Theorem.

THEOREM 1 *Suppose that q is a chain join, and p is the eigenorder of q. Then $minT_q^p = minT_q$.*

From Theorem 1, it follows that using the eigenorder of a chain query q as one input of the algorithm in [21], we may find $minT_q$ in polynomial time. To prove Theorem 1, we need only to prove the following Lemma.

LEMMA 1 *Suppose that q is a chain join, p is the eigenorder of q, and $t \in T_q$. Then there is an execution tree in T_q^p whose cost is not greater than the cost of t.*

Lemma 2 is the key of the proof of Lemma 1. To present Lemma 2, we first provide some additional notation about a rooted-tree. Suppose that t_u is a rooted-tree such that the root u has only one child x, and the child set of x is $U = \{v_i: 1 \leq i \leq k\}$. A *2-way separation* $t(\{v_{i_1}, \ldots, v_{i_j}\}, y)$ of t_u is also a rooted-tree which has the same root u as t_u, and splits t_u into two parts by adding a child y to u such that the child set of y is $\{v_{i_1}, \ldots, v_{i_j}\}$ and the child set of x is changed to $U - \{v_{i_1}, \ldots, v_{i_j}\}$, while the other parts of the separation are the same as t_u (see Figure 4, for example).

LEMMA 2 *Suppose that t_u (see Figure 4(a)) is a rooted-subtree of an execution tree t such that in t_u, the root u of t_u has only one child x, the child set of x is $U = \{v_i: 1 \leq i \leq k\}$, and t_u contains the induced rooted-subtree t_{v_i} of each v_i with respect to t. Further suppose that for each v_i, the join result produced by t_{v_i} is r_i, and there are l nodes in U, say v_i for $1 \leq i \leq l$ such that*

$$(3.1)\quad |r_1 \bowtie r_2 \bowtie \cdots \bowtie r_k| \geq |r_1 \bowtie r_2 \bowtie \cdots \bowtie r_l| + |r_{l+1} \bowtie \cdots \bowtie r_k|.$$

Then the cost of the execution tree by the replacement of t_u by its 2-way separation $t(\{v_{l+1}, \ldots, v_m\}, y)$ is not greater than that of t, where y represents the same site (individual computer) as x.

Proof: Observing the condition (3.1), the intuition is that to send out the result of the join $r_1 \bowtie r_2 \bowtie \cdots \bowtie r_k$ is never better than to respectively send out the results of the joins $r_1 \bowtie r_2 \bowtie \cdots \bowtie r_l$ and $r_{l+1} \bowtie \cdots \bowtie r_k$. This leads to an immediate verification of the corresponding inequality about the tree costs. ■

If the relation schemata of two relations do not have common parts, then the join of these two relations is the Cartesian Product. Thus, the following Lemma is trivially true.

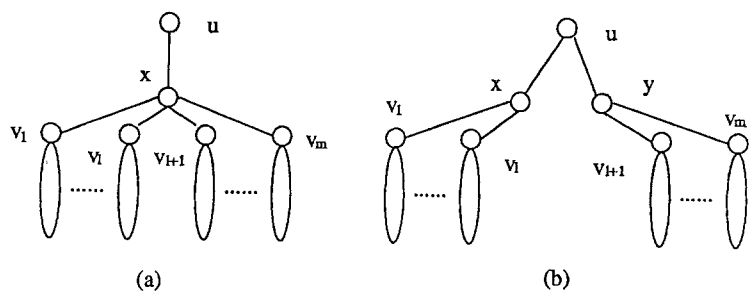

Figure 4. 2-way separation.

LEMMA 3 *Suppose that f_1 and f_2 are two relations, and $F_1 \cap F_2 = \emptyset$. Then $|f_1 \bowtie f_2| \geq |f_1| + |f_2|$.*

The proof of Lemma 1 is based on a sequential transformation from an execution tree in T_q to an execution tree in T_q^p, using Lemmas 2 and 3, such that the result of each transformation will decrease the number of nodes which do not preserve p, and will never increase the cost.

Proof of Lemma 1: If t is also in T_q^p, then the Lemma is certainly true. We prove this Lemma under the assumption that t is not in T_q^p, that is, t does not preserve p. Without loss of generality, we may assume that q is a join of m relations, and that the eigenorder p of q is $(1, 2, \ldots, m)$.

Note that t preserves p if and only if each node preserves p. Suppose that in t, there are l nodes, $V_p = \{u_i\colon 1 \leq i \leq l\}$, which do not preserve p.

Note that each leaf in t preserves p. It follows that in V_p, there is a node u_j such that in the induced rooted-subtree t_1 of u_j with respect to t, all the nodes of t_1 preserve p, except that u_j does not preserve p.

It is clear that u_j cannot be a leaf, nor the root of t—otherwise t preserves p. Suppose that the parent of u_j is u_0, and the child set of u_j is $\{v_i\colon 1 \leq i \leq k\}$. Let t_0 denote the rooted-subtree of t consisting of the root u_0 and t_1.

Because each v_i $(1 \leq i \leq k)$ preserves p, we may assume that the relations in the induced rooted-subtree of v_i are f_h for $L_i \leq h \leq R_i$ where L_i and R_i are integers. Without loss of generality, we may assume that for $1 \leq i \leq k-1$, $R_i + 1 \leq L_i + 1$. Further, suppose that there are K children v_{i_h} $(1 \leq h \leq K)$ of u_j such that $L_{i_{h+1}} > R_{i_h} + 1$. From the fact that q is a chain join, p is the eigenorder, and Lemma 3, it follows that for $1 \leq h \leq K$, say $L_{i_0} = L_1$, $|f_{L_{i_{h-1}}} \bowtie \cdots \bowtie f_{R_k}| \geq |f_{L_{i_{h-1}}} \bowtie \cdots \bowtie f_{R_{i_h}}| + |f_{L_{i_{h+1}}} \bowtie \cdots \bowtie f_{R_k}|$.

An iterative application of Lemma 2 implies that the cost of the execution tree t'—constructed as follows—is not greater than the cost of t. The execution tree t' is obtained from t by the replacement of t_0 by the rooted-tree t_0' with the root u_0 in which the children of u_0 are w_x for $1 \leq x \leq K$ representing the same site in the network, $w_x = u_j$, the children of each w_x are v_h for $i_{x-1} + 1 \leq h \leq i_x$, and the induced rooted-subtree of each v_h with respect to t' is the induced rooted-subtree of v_h with respect to t (see Figure 5).

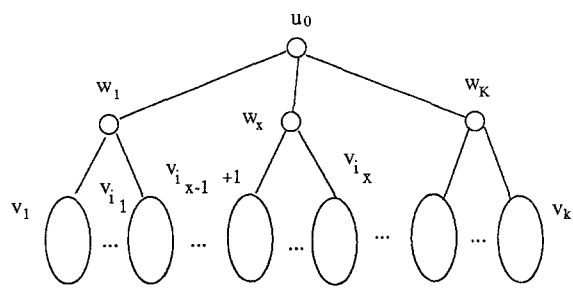

Figure 5. An illustration to the proof of Lemma 1.

Note that each w_x above preserves p. It follows that the set of nodes in t' which do not preserve p is a subset of $V_p - \{u_j\}$. Thus, one may continue the above procedure until each node preserves p. ■

4. An effective pre-process in the join computation algorithm

In this section, we show a necessary and sufficient condition for $minT_q^s = minT_q^p$.

THEOREM 2 *Suppose that $p = (1, 2, \ldots, m)$ is an execution order of a join of m relations, and the network N with at least two nodes. Then $minT_q^s = minT_q^p$ for any data allocation if and only if for $1 \leq i \leq m, 1 \leq j \leq m - i$:*

(4.1) $|f_i \bowtie f_{i+1} \bowtie \cdots \bowtie f_{i+j}| \geq \sum_{l=1}^{l=j} |f_{i+l}|$.

To prove Theorem 2, we first show the following Lemma.

LEMMA 4 *Suppose that $p = (1, 2, \ldots, m)$ is an execution order of a join q of m relations, and for $1 \leq i \leq m, 1 \leq j \leq m - i$, the condition (4.1) holds. Further suppose that in an execution tree $t \in T_q^p$, there is a non-root node u such that u has at least 2 children nodes, and in the induced rooted-subtree of u with respect to t, there are no other nodes which has more than 1 child. Then the cost of the rooted-tree t', obtained from t by the deletion of u and the attachment of the children of u to the parent of u, is not greater than the cost of t.*

Proof: From the fact that t preserves p, the fact that for $1 \leq i \leq n, 1 \leq j \leq m - i$, the condition (4.1) holds, and the fact that the networks considered in this paper are metric, it immediately follows that this Lemma holds. ■

Proof of Theorem 2: First we prove the "only if" part. Suppose that the "only if" part is not true. It implies that there is an i and an $j(j \leq m - i)$ such that

$$|\bowtie_{l=0}^{l=j} f_{i+l}| < \sum_{l=1}^{j} |f_{i+l}|,$$

and that $minT_q^p = minT_q^s$.

Let the data allocation L allocate the relations f_l $(i \leq l \leq i + j)$ on a site other than the result site of q, and allocate the other relations to the result site. Then, one may immediately verify that for this data allocation $minT_q^p < minT_q^s$. Thus, the "only if" part is true.

Now we prove the "if" part. We need only to prove that for each execution tree $t \in T_q^p$, there is an execution tree $\bar{t} \in T_q^s$ whose cost is not greater than the cost of t.

For an execution tree $t \in T_q^p$ which is not in T_q^s, one may obtain an execution $t' \in T_q^p$ by iterative applications of Lemma 4 such that in t', all the nodes, other than the root, have at most only one child, and the cost of t' is not greater than the cost of t. An execution tree $\bar{t}$ in T_q^s may be obtained by the deletion of all the nodes, other than the root and the leaves from t', and the attachment of the leaves to the root. From the fact

that the network is metric, it follows that the cost of $\bar{t}$ is not greater than the cost of t. ■

COROLLARY 1 *Suppose that* $p = (1, 2, \ldots, m)$ *is the eigenorder of a chain join of* m *relations, and for* $1 \leq i \leq n, 1 \leq j \leq m - i$, *the condition* (4.1) *holds. Then* $minT_q = minT_q^s$.

We now present a simple linear algorithm to check the condition (4.1) for the input matrix $S^p_{m\times m}$ as described in Section 2.3 with respect to an execution order p.

Algorithm 1. *CHECK*

1. Input: $S^q_{m\times m}$;
2. Output: ID : Boolean Value;
3. { $ID \leftarrow$ true;
4. **for** $i = 1$ **to** m **do**
5. { $N \leftarrow a_{i,1}$;
6. **for** $j = i + 1$ **to** m **do**
7. { $N \leftarrow N + a_{j,1}$;
8. **if** $N > a_{i,j-i+1}$ **then**
9. Stop this algorithm and $ID \leftarrow$ false } } }.

The algorithm outputs $ID = true$ if condition holds, otherwise $ID = false$.

It is clear that this algorithm may be executed in $O(m^2)$ time which is linear with respect to the size of the input $S^p_{m\times m}$. To find $minT_q^s$, it takes $O(mn)$ time (see Section 2.3), where m is the number of fragments in a join, and n is the number of sites of the network. We can expect that there are a large class of joins with a given execution order (for example, a chain join) in which the condition (4.1) holds [22]. This implies that with the application of the algorithm CHECK to the algorithm in [21], the time complexity $\theta(m^2n^2 + m^3n)$ may be reduced to $O(mn + m^2)$, in many cases, for general networks, and reduced to $O(m^2)$ for uniform networks. This is a significant reduction, since in real life, m is bounded by a small constant.

5. Total communication cost vs total local processing cost

The local processing cost, which includes local CPU and I/O costs, has been addressed in the problems of query processing optimization in either a uniprocessor environment [16, 24] or a parallel system environment [11, 17, 23]. The algorithm in [23] is optimal for parallel processing a chain join based on a hash approach; that is, the total cost including the local processing cost will be minimized. Consider that a uniprocessor environment is a special case of a parallel system. Thus, the algorithm in [23] may also process a chain join, with the minimum local processing cost in a single processor environment, by using a hash-join approach.

In most of the previous studies in distributed query processing optimization, the local processing cost has been ignored when forming the total cost function. Only the total communication cost has been addressed. Along with the development of network techniques, the local processing cost is also expected to play an important role in the optimization problem. The computational intractability of the problem to minimize the total cost for processing a distributed query is simply exacerbated by an inclusion of the local processing cost. However, we can show that in a distributed environment of homogeneous sites, the simple strategy for processing a distributed join and an optimal query processing algorithm in a uniprocess environment will constitute an optimal processing for a simple chain distributed join.

Given an execution plan EP of a distributed join q, let $TC_{EP,q}$ denote the total cost for processing q by EP, $CC_{EP,q}$ denote the total communication cost, and $LC_{EP,q}$ denote the total local processing cost. Thus, $TC_{EP,q} = CC_{EP,q} + LC_{EP,q}$.

For a simple chain join q_0, let the execution plan EP_0 consist of the simple strategy EP_s along with an optimal join processing algorithm EP_l in a uniprocessor environment (that is, by EP_l the computation cost of q in a uniprocessor is minimized). We have that

$$TC_{EP_0,q_0} = CC_{EP_0,q_0} + LC_{EP_0,q_0} = CC_{EP_s,q_0} + LC_{EP_l,q_0}.$$

Clearly, for any execution plan EP of q_0, we have $CC_{EP_s,q_0} \leq CC_{EP,q_0}$ according to our results in the last two sections. Moreover, in a distributed environment, where all the sites are the same, it is trivially true that $LC_{EP_l,q_0} \leq LC_{EP,q_0}$, and then EP_0 is the optimal execution plan to process a simple chain join with respect to the minimization of the total cost. (For example, if we use only a hash-join approach to compute a join at local sites, then the polynomial time algorithm in [23] may act as EP_l.)

6. Conclusion

In this paper, we proved that the eigenorder of a chain join is the optimal pre-defined order to implement the algorithm in [21]. This implies we may apply the algorithm in [21] to the environment in [8] to find an optimal solution for a chain multi-join. Then we obtained a significant reduction on the complexity of the algorithm in [21] for a large class of joins by including a pre-processing procedure to evaluate the category of the query.

Acknowledgment

The work of the first named author is partially supported by a GIRD grant. The authors would like to thank Dr. Marian W. Orlowski for a useful discussion. We are also grateful to the anonymous referees for their comments and suggestions.

References

1. P.M.G. Apers, A. Hevner, and S.B. Yao, "Optimization Algorithms for Distributed Queries," *IEEE Transactions on Software Engineering*, SE-9(1), pp. 57–68, 1983.

2. 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.

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

4. P.A. Bernstein, N. Goodman, E. Wong, C.L. Reeve, and J.B. Rothe, "Query Processing in a System for Distributed Database (SDD-1)," *ACM Transaction on Database Systems*, 6(4), pp. 602–625, 1981.

5. J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, The Macmillan, 1978.

6. D. Chiu, P.A. Bernstein, and Y. Ho, "Optimizing Chain Queries in a Distributed Database System," *SIAM Journal on Computing*, 13(1), pp. 116–134, 1984.

7. 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.

8. M.-S. Chen and P.S. Yu, "Using Join Operations as Reducers in Distributed Query Processing," *Databases in Parallel and Distributed Systems*, pp. 116–123, 1990.

9. T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, The MIT press, 1990.

10. C.J. Date, *An Introduction to Database System*, 2 Addision-Wesley, 1982.

11. S. Ganguly, W. Hasan, and R. Krishnamurthy, "Query Optimization for Parallel Execution," *SIGMOD Record*, 21(2), pp. 9–18, 1992.

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

13. A.R. Hevner and S.B. Yao, "Query Processing in Distributed Database Systems," *IEEE Transactions on Software Engineering*, SE-5(3), pp. 177–187, 1979.

14. T. Ibaraki and T. Kameda, "On the Optimal Nesting Order for Computing N-Relations Joins," *ACM Transactions Database Systems*, 9, pp. 482–502, 1984.

15. 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.

16. R. Krishnamurthy, H. Boral, and C. Zaniolo, "Optimization of Nonrecursive Queries," *Proceedings of VLDB 86*, pp. 128–137, 1986.

17. H. Lu, M.C. Shan, and K.L. Tan, "Optimization of Multi-Way Join Queries for Parallel Execution," *Proceedings of VLDB 91*, pp. 549–560, 1991.

18. S. Pramanik and D. Vineyard, "Optimizing Join Queries in Distributed Databases," *IEEE Transactions on Software Engineering*, 14(9), pp. 1319–1326, 1988.

19. D. Maier, *Theory of Relational Databases*, Computer Science Press, 1993.

20. Y. Mansour and B. Patt-Shamir, "Greedy Packet Scheduling on Shortest Paths," *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pp. 165–176, 1991.

21. M.W. Orlowski, "On Optimisation of Joins in Distributed Database Systems," *Future Databases 92*, World Scientific, pp. 106–114, 1992.

22. M.W. Orlowski, *Private Communication.*

23. 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.

24. A. Swami and A. Gupta, "Optimization of Large Join Queries: Combining Heuristics and Combinatorial Techniques," *Proceedings of SIGMOD 89*, pp. 367–376, 1989.

25. A.E. Taylor, *Advanced Calculus*, Ginn, 1955.

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

27. J.D. Ullman, *Principles of Database Systems*, Computer Science Press, Rockville, MD, 1982.

28. C.P. Wang, "The Complexity of Processing Tree Queries in Distributed Databases," *2nd IEEE Symposium on Parallel and Distributed Processing*, pp. 604–611, 1990.

29. C.P. Wang, V.O.K. Li, and A.L.P. Chen, "One-shot Semi-Join execution strategies for processing distributed queries," *7th IEEE Data Engineering Conference*, pp. 756–763, 1991.

30. 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), pp. 325–333, 1992.

31. E. Wong, "Dynamic Rematerialization: Processing Distributed Queries Using Redundant Data," *IEEE Transactions on Software Engineering*, SE-9(3), pp. 228–232, 1983.

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

33. C.T. Yu, Z.M. Ozsoyoglu and K. Lam, "Optimization of Distributed Tree Queries," *Journal of Computer and System Science*, 29, pp. 399–433, 1984.