

Distributed Query Processing in the Internet: Exploring Relation Replication and Network Characteristics

Chang-Hung Lee and Ming-Syan Chen
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, ROC

E-mail: { chlee@arbor.ee.ntu.edu.tw, mschen@cc.ee.ntu.edu.tw }

Abstract

We introduce the concept of network graph for distributed query processing. Semijoins and joins are termed contributive replicated semijoins and contributive replicated joins, respectively, when they are interleaved into a join sequence to reduce the amount of data transmission cost required in a network with replicated relations. Our solution procedure consists of three consecutive steps, namely relation selection, join sequence scheduling and merge processing. A simulator is developed to evaluate the performance of algorithms devised. Our results show that the approach of interleaving a join sequence with contributive replicated semijoins/joins is not only efficient in its execution but also effective in reducing the total amount of data transmission cost required to process distributed queries.

1. Introduction

Recently, as the number of Internet applications increases rapidly, there has been a growing demand for distributed database systems, where several databases for various applications are physically located in different sites. To save data communication cost, using mirror sites, among others, has become a favorable solution [2, 8]. However, it is noted that, instead of replicating databases at several network sites, it would be more economic to allocate only some relation replication in distributed network sites [9, 10, 15]. While having its own advantages, how to perform a join operation with relation replication efficiently is a difficult problem since the data transmission over the network is expensive and how to minimize the transmission cost for performing a join with relation replication is intrinsically hard to solve. Such a problem is even more important and difficult to resolve when a multi-join query is being carried out.

Moreover, conventional methodologies of query processing do not cope with some practical network issues

in the distributed relational database system. Specifically, even though many prior studies have developed several efficient algorithms for join or semijoin processing [4, 5, 13], little work has taken the network topology and the relation replication together into consideration [9]. For example, since most prior work does not consider relation replication, each transmission path from one relation to another is assumed to be a fixed path in the network graph, which is indeed a limitation in practice. Clearly, without considering the network characteristics, the solution quality for distributed query processing may degrade.

Consequently, to remedy the above deficiencies, we shall explore in this paper distributed query processing while taking relation replication and network characteristics into consideration. In general, optimizing large queries in distributed relational database is a complicated problem [5] and many forms of it are NP-hard [11]. The discrete domain of different schedules to answer a query is huge [1]. In view of this, our solution procedure for distributed query processing in this paper is decomposed into three consecutive steps, namely relation selection, join sequence scheduling and merge processing. However, as the effect of semijoins depends upon subsequent join and semijoin operations, the use of semijoins cannot be determined in isolation. The problem becomes even more complicated when relation replication and network characteristics are considered. In view of this fact, we develop in this paper the concepts of *contributive replicated semijoins* and *contributive replicated joins*, and utilize these operations to devise efficient algorithms for distributed query processing. Semijoins and joins are termed contributive replicated semijoins and contributive replicated joins, respectively, when they are interleaved into a join sequence to reduce the amount of data transmission cost required in a network with replicated relations. A simulator is developed to evaluate the performance of algorithms devised. It is shown by our results that the approach of interleaving a join sequence with contributive replicated semijoins or contributive replicated joins is not only efficient in its execution but also effective in reducing

the total amount of data transmission cost required when processing distributed queries.

The contributions of this paper are twofold. We not only devise innovative algorithms for distributed query processing by exploring relation replication and network characteristics, but also conduct performance studies on several important parameters to provide many insights into the problem. Note that the effect of relation replication and that of network characteristics are in fact entangled, thus justifying the necessity of our approach to considering these two factors together.

The rest of this article is organized as follows. The notation, definitions and assumptions are given in Section 2. In Section 3, we describe network-based distributed query processing. Simulation model and results are presented in Section 4. This paper concludes with Section 5.

2. Preliminaries

As in most previous works in distributed databases [1, 5, 9, 14], we assume that the cost for executing a query can mainly be expressed in terms of the total amount of inter-site data transmission cost required. Also, it is assumed that a query is in the form of conjunctions of *equi-join* predicates and all attributes are renamed in such a way that two join attributes have the same attribute name if and only if they have a join predicate between them.

For ease of discussion, G_Q denotes the query graph and $|G_Q|$ indicates the number of relations in a query, while G_N is the network graph of query and $|G_N|$ shows the number of network sites in a network graph. We use transmission coefficient $TC_{m \rightarrow n}$ to indicate the communication cost of transmitting one data unit from site S_m to site S_n . $|K|$ is used to denote the cardinality of a set K . Let w_A be the width of an attribute A and w_{R_i} be the width of a tuple in R_i . The size of the total amount of data in R_i can then be denoted by $w_{R_i}|R_i|$. For notational simplicity, $|A|$ is used to denote the cardinality of the domain of an attribute A . Define the selectivity $\rho_{i,a}$ of attribute A in R_i as $\frac{|R_i(A)|}{|A|}$, where $R_i(A)$ is the set of distinct values for the attribute A in R_i . $R_i - A \rightarrow R_j$ means a semijoin from R_i to R_j on attribute A . To simplify the notation, $R_i \rightarrow R_j$ is used to mean a semijoin from R_i to R_j in the case that the semijoin attribute does not have to be specified. Also, the notation $R_i \Rightarrow R_j$ is used to mean that R_i is sent to the site of R_j and a join operation is performed with R_j there. We use R_i' to denote the resulting relation after some reducers (joins or semijoins) are applied to an original relation R_i . The transmission coefficient $TC_{m \rightarrow n}$ is used to serve as the statistical average value of the relative bandwidth in each network edge and we define an *effectual semijoin* as follows.

Definition 1 (Effectual semijoin): A semijoin, $R_1(S_1) - B \rightarrow R_2(S_2)$, is called *effectual*, if its cost of sending $R_1(B)$, i.e., $TC_{1 \rightarrow 2}(w_B|R_1(B)|) = w_B|B|\rho_{1,b}$,

is smaller than its benefit, i.e., $TC_{2 \rightarrow 1}(w_{R_2}|R_2| - w_{R_2}|R_2|\rho_{1,b}) = w_{R_2}|R_2|(1 - \rho_{1,b})$, where R_1 and R_2 are located at sites S_1 and S_2 respectively, and $w_{R_2}|R_2|$ and $w_{R_2}|R_2|\rho_{1,b}$ represent, respectively, the sizes of R_2 before and after the semijoin. Thus, $w_B|R_1(B)|TC_{1 \rightarrow 2}$ is used to denote the cost of a semijoin $R_1 - B \rightarrow R_2$.

In this paper, we use $w_{R_1}|R_1| \times TC_{1 \rightarrow 2}$ to denote the cost of a join $R_1 \Rightarrow R_2$. Furthermore, we assume that the values of attributes are uniformly distributed over all tuples in a relation and that the values of one attribute are independent from each other. The cardinalities of the resulting relations from join operations can thus be estimated according to the formula in [3]. Also, all tuples are assumed to have the same size. Results on the effect of data skew can be found in [7, 12].

We shall introduce the concepts of contributive replicated semijoin and contributive replicated join which are in essence new reducers to perform a join sequence. By interleaving these reducers into a join sequence, the data transmission cost required can be significantly reduced.

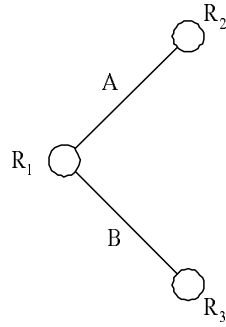
Definition 2 (Contributive replicated semijoin): A semijoin, $R_i(S_{replicated}) - A \rightarrow R_j(S_{target})$, is said to be *contributive replicated*, if its communication cost of sending $R_i(A)$, i.e., $TC_{replicated \rightarrow target} \times (w_A|R_i(A)| = w_A|A|\rho_{i,a})$, is smaller than the cost of original semijoin, i.e., $R_i(S_{original}) - A \rightarrow R_j(S_{target})$, where $R_i(S_{replicated})$, $R_i(S_{original})$, and $R_j(S_{target})$ mean that R_i , R_i and R_j are located at sites $S_{replicated}$, $S_{original}$ and S_{target} respectively.

Definition 3 (Contributive replicated join): A join, $R_i(S_{replicated}) \Rightarrow R_j(S_{target})$, is said to be *contributive replicated*, if its communication cost of sending R_i , i.e., $TC_{replicated \rightarrow target} \times (w_{R_i}|R_i|)$, is smaller than the cost of the original join operation, i.e., $R_i(S_{original}) \Rightarrow R_j(S_{target})$, while the information in $R_i(S_{replicated})$ should be the same as in the $R_i(S_{original})$.

As will be shown later, contributive replicated semijoin and contributive replicated join are very useful in reducing the data transmission cost required for distributed query processing. Note that the processing time in each computing host may vary, and its system dependent optimization is a challenging issue itself [1] and is beyond the scope of this paper.

3. Network-Based Dist. Query Processing

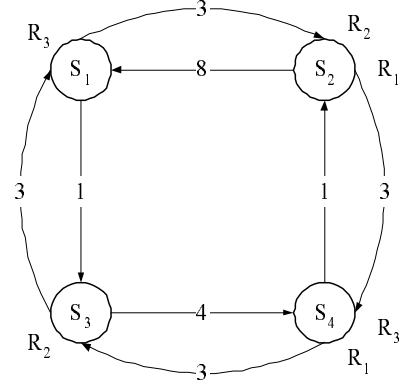
In [11], it is proved that five classes of distributed query optimization problems, i.e., local optimization of semijoins, join sequence optimization, relation semijoins on broadcasting networks, single-reducer tree queries, and full-reducer tree queries, which cover the majority of distributed query optimization problems in the query processing, are NP-hard. The query processing we study in this paper with the consideration of network characteristics and relation replication is therefore also NP-hard by reduction [6],



(a) An illustrative query

Relation R_i	Attribute X	Selectivity	Tuples $ R_i $
R ₁	A	0.7	1000
	B	0.5	
R ₂	A	0.6	1200
R ₃	B	0.6	1000
where $ A =1200, B =800$			

(b) Profile of the query in (a)



(a) An illustrative network graph

from/to	S ₁	S ₂	S ₃	S ₄
S ₁	0	3	1	5
S ₂	8	0	6	3
S ₃	3	5	0	4
S ₄	6	1	3	0

(b) Shortest path among network sites

Figure 1. An illustrative query and its profile

Figure 2. An illustrative network graph and its shortest path among network sites

thereby justifying the need of applying heuristic approaches to solving this problem.

In this paper, the procedure for distributed query processing is decomposed into three consecutive steps, i.e., namely relation selection, join sequence scheduling and merge processing. For ease of exposition, an illustrative example is described below to explore each proposed scheme. Without loss of generality, the query we use is as follows.

select A, B from R_1, R_2, R_3 where $R_1.A = R_2.A$ and $R_1.B = R_3.B$.

Consider a company which has three fragments of databases, $R_1, R_2,$ and $R_3,$ located in four sites, $S_1, S_2, S_3,$ and $S_4.$ For example, R_1 resides in sites S_2 and $S_4,$ and on the other hand S_2 contains both R_1 and $R_2.$ The corresponding query graph and its profile are given in Figure 1a and Figure 1b, respectively. Specifically, as shown in Figure 2a, a network graph illustrates the distribution of relations in network sites with relation replication considered. The transmission cost among network sites, i.e., $TC_{m \rightarrow n},$ is given in each network edge. For instance, the value of “3” in the arrow from S_1 to S_2 means that the transmission cost of sending one data unit from S_1 to S_2 is 3 units, i.e., $TC_{1 \rightarrow 2} = 3.$ Note that, the asymmetry feature between upload path and download path is also considered in this network topology since $TC_{m \rightarrow n}$ and $TC_{n \rightarrow m}$ may differ from each other. With a shortest path algorithm [6], each minimum cost among network sites is generated in Figure 2b.

The transmission cost from S_m to $S_m,$ $TC_{m \rightarrow m},$ is given as “0”. In this example, S_1 is assumed to be the final site $S_{final},$ where the final results are needed.

3.1. Relation Selection

In the first step, i.e., relation selection, our main purpose is to search appropriate relations from the existing network graph based on the relationship among relations and network sites.

3.1.1 Independent Relation Selection (IS)

The scheme of Independent relation Selection (abbreviated as *IS*) explores how to select each relation required from the most appropriate network site, from which the cost of sending data to the final site is the minimal. For example, as shown in Figure 2a, R_1 is allocated at sites S_2 and $S_4.$ It can be seen that $TC_{4 \rightarrow 1} = 6$ is smaller than $TC_{2 \rightarrow 1} = 8$ from Figure 2b. $R_1(S_4)$ is thus selected. Similarly, $R_2(S_3)$ is then selected since $TC_{3 \rightarrow 1} = 3 < TC_{2 \rightarrow 1} = 8.$ Finally $R_3(S_1)$ is selected since S_1 is the final site. Scheme *IS* can be formally described as follows. For better readability, a few abbreviations are utilized in *IS.* $cost(R_i(S_m), S_{final})$ represents the transmission cost of sending R_i from S_m to

the final site S_{final} , which needs the final result of query. S_{RSP} indicates the set of relation-site pairs selected.

Scheme IS : Independent relation Selection

1. begin
2. $S_{RSP} := \phi$;
3. foreach R_i in G_Q do
4. begin
5. foreach $R_i(S_m)$ in G_N do
6. begin
7. select $R_i(S_m)$ such that $cost(R_i(S_m), S_{final}) = \min\{cost(R_i(S_m), S_{final})\}$;
8. $S_{RSP} := S_{RSP} \cup \{R_i(S_m)\}$;
9. end
10. end
11. end

Clearly, since the transmission cost of sending data to the final site is the only decisive parameter, the relation which has lower transmission coefficient $TC_{m \rightarrow final}$ will be selected under scheme IS.

3.1.2 Greedy Relation Selection (GS)

Note that IS described above does not take the intrinsic relationship among relations into consideration. However, as shown below, this relationship, if properly exploited, can lead to prominent performance improvement. In view of this, an alternative scheme, i.e., Greedy relation Selection (abbreviatedly as GS), is devised as follows. Scheme GS is greedy in nature. The first selected site $R_3(S_1)$ is the one which incurs the minimal transmission cost of sending its data to the final site. Next, we select $R_1(S_4)$ as the second relation because the transmission cost of sending data from $R_1(S_4)$ to $R_3(S_1)$ is the smallest one among all legitimate considerations. The sequentially selected relation-site pairs are the ones which incur the minimal communication cost to any selected relation-site pair. With the minimum cost of a combination of semijoin/join between $R_2(S_2)$ and $R_1(S_4)$ (i.e., $R_1(S_4) \rightarrow R_2(S_2)$ and $R_2(S_2) \Rightarrow R_1(S_4)$), $R_2(S_2)$ is thus the third selected relation. In general, scheme GS can be described as follows. Let $cost(R_i(S_m), R_j(S_n), SMJ)$ be the transmission cost of merging $R_i(S_m)$ to $R_j(S_n)$ with the join operation SMJ. S_{RSP} indicates the set of relation-site pairs selected. Note that the relationship among relations is taken as an important parameter in GS.

Scheme GS : Greedy relation Selection

1. begin
2. $S_{RSP} := \phi$;
3. select $R_i(S_m) \in G_N$ such that $cost(R_i(S_m), S_{final}) = \min_{R_i(S_m) \in G_N} \{cost(R_i(S_m), S_{final})\}$;
4. $S_{RSP} := S_{RSP} \cup \{R_i(S_m)\}$;
5. foreach $R_i \notin S_{RSP}$ do
6. begin
7. if $(R_i \Rightarrow R_j \leq R \rightarrow R_i + R_i! \Rightarrow R_j)$
then $SMJ := R_i \Rightarrow R_j$;
8. else $(SMJ := R_j \rightarrow R_i + R_i! \Rightarrow R_j)$;
/* effectual semijoin */
9. select $R_i(S_m) \notin S_{RSP}$ in G_N

such that $cost(R_i(S_m), R_j(S_n), SMJ) = \min_{R_i(S_m) \notin S_{RSP}, R_j(S_n) \in S_{RSP}} \{cost(R_i(S_m), R_j(S_n), SMJ)\}$;

10. $S_{RSP} := S_{RSP} \cup \{R_i(S_m)\}$;
11. end
12. end

In GS, not only is the transmission path taken as a parameter for relation selection, but also the intrinsic relationship among relations is utilized to make the decision for relation selection. In addition, the notion of effectual semijoin is used to evaluate the usefulness of semijoins. Consequently, with relations selected, the join sequence can next be determined.

3.2. Join Sequence Scheduling (JSS)

After all the relation-site pairs, such as $R_1(S_4)$, $R_2(S_3)$ and $R_3(S_1)$ from IS scheme, are selected in the first step, we shall determine the join sequence for later merge processing. With the method of interleaving joins with semijoins, the step of Join Sequence Scheduling, abbreviatedly as JSS, is devised. First, the relation which needs the minimal communication cost of sending its data to the final site is selected as the one to send merged results to the final site. $R_3(S_1)$ is thus selected since R_3 is located at the final site S_1 . Next, it can be seen that a combination of semijoin $R_3(S_1) \rightarrow R_1(S_4)$ and join $R_1(S_4) \Rightarrow R_3(S_1)$ is the operation with the minimal cost to merge relation $R_1(S_4)$ to $R_3(S_1)$. Then, the second relation selected is $R_1(S_4)$. Finally, with $R_2(S_3) \Rightarrow R_1(S_4)$, $R_2(S_3)$ is the final relation to be selected into the join sequence scheduling. As a result, the join sequence of the selected relations obtained by IS scheme is $Join(Join(R_2(S_3), R_1(S_4)), R_3(S_1))$, where $Join(R_i(S_m), R_j(S_n))$ indicates the join result of joining $R_i(S_m)$ with $R_j(S_n)$. According to the operations above, the algorithmic form of scheme JSS can be described as follows, where $S_{JSS}(i)$ means the i -th step of the join sequence, and S_{JSS} is the set of relation-site pairs selected by scheme JSS. S_{RSP} is the set of relation-site pairs selected from the step of relation selection (i.e., by scheme IS and scheme GS).

Scheme JSS : Join Sequence Scheduling

1. begin
2. $S_{JSS}(\bullet) := \phi$, $S_{JSS} := \phi$ and $i = 0$;
3. select $R_i(S_m) \in S_{RSP}$ such that
 $cost(R_i(S_m), S_{final}) = \min_{R_i(S_m) \in S_{RSP}} \{cost(R_i(S_m), S_{final})\}$;
4. $S_{RSP} := S_{RSP} - \{R_i(S_m)\}$;
5. $S_{JSS} := S_{JSS} \cup \{R_i(S_m)\}$;
6. foreach $R_i(S_m) \in S_{RSP}$ do
7. begin
8. if $(R_i \Rightarrow R_j \leq R_j \rightarrow R_i + R_i! \Rightarrow R_j)$
then $SMJ := R_i \Rightarrow R_j$;
9. else $(SMJ := R_j \rightarrow R_i + R_i! \Rightarrow R_j)$;
/* effectual semijoin */
10. select $R_i(S_m) \in S_{RSP}$ and $R_j(S_n) \in S_{JSS}$

such that $cost(R_i(S_m), R_j(S_n), SMJ)$
 $= \min_{\substack{R_i(S_m) \in S_{RSP}, \\ R_j(S_n) \in S_{SJS}}} \{cost(R_i(S_m), R_j(S_n), SMJ)\};$

11. $S_{RSP} := S_{RSP} - \{R_i(S_m)\};$
12. $S_{SJS} := S_{SJS} \cup \{R_i(S_m)\};$
13. $S_{JS}(i) := (R_i(S_m), R_j(S_n))$ and $i = i + 1;$
14. end
15. end

Thus, according to the S_{RSP} (i.e., $\{R_1(S_4), R_2(S_3), R_3(S_1)\}$) generated in IS, we have $S_{JS}(0) = (R_1(S_4), R_3(S_1))$ and $S_{JS}(1) = (R_2(S_3), R_1(S_4))$ in the set of $S_{JS}(\bullet)$. These operations will be utilized in merge processing.

3.3. Merge Processing

After the join sequence is decided, the merge processing is performed. The merge processing can be taken as the revised and polished step for the step of JSS. The contributive replicated semijoins and contributive replicated joins will be interleaved into a join sequence in the step of merge processing. Two alternative schemes, normal merge processing (abbreviatedly as *NM*) and merge relation replication (abbreviatedly as *MR*), are devised.

3.3.1 Normal Merge Processing (NM)

Normal Merge processing (*NM*) interleaves the join sequence obtained with semijoins in the merge processing. As long as an effectual semijoin is found in each step of merge processing, that semijoin is employed as a reducer. Therefore, the sequence of the merge processing for this example in *JSS* would be: $R_2(S_3) \Rightarrow R_1(S_4) : 4,800$, $R_3(S_1) \rightarrow R_1(S_4) : 2,400$, and $R_1(S_2) \Rightarrow R_3(S_1) : 7,200$. The total amount of transmission cost is 14,400. The algorithmic form of *NM* is described below, where $|G_Q|$ indicates the number of relations in the query graph G_Q . Joins and effectual semijoins are two merging operations. The minimum-cost operation will be selected from these two operations in each step of merging processing until the query result is developed. After each merging processing, the profile of query graph is updated accordingly.

Scheme NM : Normal Merge processing

1. begin
2. for $k = |G_Q| - 1$ to 0;
3. begin
4. $(R_i(S_m), R_j(S_n)) := S_{JS}(k);$
5. $k = k - 1;$
6. if $(R_i \Rightarrow R_j \leq R_j \rightarrow R_i + R_i' \Rightarrow R_j)$
then $SMJ := R_i \Rightarrow R_j;$ /*joins*/
7. else $(SMJ := R_j \rightarrow R_i + R_i' \Rightarrow R_j);$
/*effectual semijoins*/
8. merge $R_i(S_m)$ and $R_j(S_n)$ to $R_j(S_n)$ with SMJ
and update the profile of query graph accordingly;
9. end
10. end

3.3.2 Merge relation Replication (MR)

Note that replicated relations are not utilized in *NM*. To remedy this, instead of directly merging all relations selected, scheme *MR* (standing for Merge relation Replication) takes the replicated relations on the unselected sites into consideration for merge processing. Therefore, before each merge processing is executed, all replicated relations are examined. When contributive replicated joins or contributive replicated semijoins are identified, new $R_i(S_{replicated})$ replaces prior $R_i(S_{original})$. Thus, the merge sequence of the previous example in Section 3.3.1 becomes the following: $R_2(S_2) \Rightarrow R_1(S_4) : 3,600$, $R_3(S_4) \Rightarrow R_1(S_4) : 0$, and $R_1(S_4) \Rightarrow R_3(S_1) : 7,200$. Thus, the total transmission cost is 10,800. Instead of using $R_2(S_3) \Rightarrow R_1(S_4)$ and $R_3(S_1) \rightarrow R_1(S_4)$ in scheme *NM*, a contributive replicated join $R_2(S_2) \Rightarrow R_1(S_4)$ and a contributive replicated semijoin $R_3(S_4) \rightarrow R_1(S_4)$ are utilized in scheme *MR* to save 25%, i.e., $\frac{14400-10800}{14400}$, of the transmission cost for merging processing, showing the very advantage of scheme *MR*. A formal description of *MR* is described below, where S_{MR} is the set of merged relations. In Scheme *MR*, $cost_1$ indicates the transmission cost of merging two relations with the join method. The $cost_2$ indicates the one of using the combination of joins/effectual semijoins. The $cost_3$ denotes the one of utilizing the contributive replicated join to merge. The $cost_4$ contains the contributive replicated semijoin in the merging processing. In each step of merging two relations, scheme *MR* selects the minimum-cost operation from $cost_1$, $cost_2$, $cost_3$, and $cost_4$. After each merging processing, the profile of query graph is updated accordingly until the final result is developed.

Scheme MR : Merge relation Replication

1. begin
2. for $k = |G_Q| - 1$ to 0;
3. begin
4. $S_{MR} := \phi;$
5. $(R_i(S_m), R_j(S_n)) := S_{JS}(k);$
6. $k = k - 1;$
7. $cost_1 = cost(R_i(S_m) \Rightarrow R_j(S_n));$ /*joins*/
8. $cost_2 = cost(R_j(S_n) \rightarrow R_i(S_m))$
 $+ cost(R_i(S_m) \Rightarrow R_j(S_n));$
/*combination of joins/effectual semijoins*/
9. $cost_3 = \min_{S_{replicated-m} \neq S_m, R_i \notin S_{MR}} \{cost(R_i(S_{replicated-m}) \Rightarrow R_j(S_n))\};$
/*contributive replicated joins*/
10. $cost_4 = \min_{S_{replicated-n} \neq S_n} \{cost(R_j(S_{replicated-n}) \rightarrow R_i(S_m))$
 $+ cost(R_i(S_m) \Rightarrow R_j(S_n))\};$
/*contributive replicated semijoins*/
11. $cost = \min\{cost_1, cost_2, cost_3, cost_4\};$
12. switch (cost)
13. {
14. case $cost_1$ do $R_i(S_m) \Rightarrow R_j(S_n);$
15. case $cost_2$ do $R_j(S_n) \rightarrow R_i(S_m)$
and $R_i(S_m) \Rightarrow R_j(S_n);$
16. case $cost_3$ do $R_i(S_{replicated-m}) \Rightarrow R_j(S_n);$

17. case cost₄ do $R_j(S_{replicated-n}) \rightarrow R_i(S_m)$
and $R_i(S_m) \Rightarrow R_j(S_n)$;
18. }
19. $S_{MR} := S_{MR} \cup \{R_j\}$ and update the profile
of query graph accordingly;
20. end
21. end

3.4. Solution Algorithms Constructed

Since there are two alternative schemes in the first step and two alternative methods in the third step, four solution algorithms are possibly developed to deal with distributed query processing. For the space limitation, three selected algorithms, namely IS_{NM} , IS_{MR} , and GS_{MR} , will be explored, where IS_{NM} can be viewed as an extension from the conventional query processing.

Algorithm IS_{NM} is to independently select each relation from the network topology and query, i.e., using scheme IS in the step of relation selection. Also, replicated relations are not taken into consideration, i.e., using scheme NM in the step of merge processing. JSS is used as the method to decide the join sequence. The resulting execution sequence is as follows: $R_2(S_3) \Rightarrow R_1(S_4) : 4,800$, $R_3(S_1) \rightarrow R_1(S_4) : 2,400$, and $R_1(S_4) \Rightarrow R_3(S_1) : 7,200$. Thus, the total transmission cost of employing IS_{NM} is 14,400. Algorithm IS_{NM} is formally described below.

Algorithm IS_{NM} : Independent Selection and Normal Merge

- Step 1: Using IS scheme to select the relations required for the query processing.
- Step 2: Deciding the join sequence with JSS scheme.
- Step 3: Merging the relations required with NM scheme.

Algorithm IS_{MR} is similar to algorithm IS_{NM} except that scheme MR is used in the step of merge processing. The resulting execution sequence will become the following: $R_2(S_2) \Rightarrow R_1(S_4) : 3,600$, $R_3(S_4) \rightarrow R_1(S_4) : 0$, and $R_1(S_4) \Rightarrow R_3(S_1) : 7,200$. Thus, the total transmission cost of employing IS_{MR} is 10,800. Comparing IS_{MR} with IS_{NM} , we find that two merging steps in IS_{MR} , $R_2(S_2) \Rightarrow R_1(S_4)$ and $R_3(S_4) \rightarrow R_1(S_4)$, are different from $R_2(S_3) \Rightarrow R_1(S_4)$ and $R_3(S_1) \rightarrow R_1(S_4)$ in IS_{NM} . By utilizing the replicated relations on the unselected sites as contributive replicated semijoin/join, about 25%, i.e., $\frac{14,400-10,800}{14,400}$, of the transmission cost is saved. Algorithm IS_{MR} is outlined as follows.

Algorithm GS_{MR} is similar to algorithm IS_{MR} except that scheme GS is used in the step of relation selection. Under algorithm GS_{MR} , the resulting execution sequence becomes: $R_1(S_2) \rightarrow R_2(S_2) : 0$, $R_2(S_2) \Rightarrow R_1(S_4) : 2,520$, $R_3(S_4) \rightarrow R_1(S_4) : 0$, and $R_1(S_4) \Rightarrow R_3(S_1) : 7,200$. Therefore, the total communication cost is 9,720. Comparing the execution results in IS_{NM} , about 32.5%, i.e., $\frac{14,400-9,720}{14,400}$, of the cost is saved by these contributive replicated semijoins and the greedy selection.

4. Simulation Model and Results

In this section, the simulation model is introduced in Section 4.1. Evaluating average value of transmission coefficients, the density of relation replication, the density of network graph, and the network size are shown in Section 4.2.

4.1. Simulation Model

To show the execution of those three algorithms, IS_{NM} , IS_{MR} , and GS_{MR} , unless mentioned otherwise, the network graph with 25 network sites $|G_N| = 25$ is utilized to evaluate several parameters of our simulation model. With a uniform distribution, the transmission coefficient $TC_{m \rightarrow n}$ of each edge of the network graph is randomly determined by a given range (0, 10) and its average value $|TC| = 5$. The simulation program was coded in C++, and input queries were generated as follows. The average number of relations in a query was pre-determined as $|G_Q| = 5$. The occurrence of an edge between two relations in the query graph was determined according to a given probability, denoted by p_{QG} . Without loss of generality, only queries with connected query graphs were deemed valid and used for our study. To make the simulation be feasibly conducted, the average number of tuples in a relation is selected from 2,500 to 3,500 and the cardinalities of attributes are also selected from 2,500 to 3,500 accordingly. As such, the join selectivities can reflect the reality.

In addition, relation-site distributions are randomly generated by the simulation program with a uniform distribution according to a given density of relation replication in a network graph, denoted by p_{RSD} . For example, when p_{RSD} is selected to be 0.5, it means that one relation, say R_i , is located at one network site, say S_m , with the probability of 0.5. In our experiments, each execution cost is the result of the average from 5,000 query executions. For each processing, three scheduling algorithms, i.e., IS_{NM} , IS_{MR} and GS_{MR} , are performed to execute the query. When two relations not having join predicates are to be joined together, a Cartesian product is performed. From our simulation, since relative performance of these schemes is not sensitive to the density of the query graph, we use $P_{QG} = 0.5$ in our simulation. The occurrence of an edge between two sites in the network graph was determined according to a given probability, denoted by p_{NG} . As before, only connected network graphs were deemed valid and used for our study.

To exhibit the benefit of relation replication, the reduction ratio $r_{IS} = \frac{|Cost(IS_{MR}) - Cost(IS_{NM})|}{Cost(IS_{NM})}$ is used to compare IS_{MR} and IS_{NM} . Similarly, $r_{GS} = \frac{|Cost(GS_{MR}) - Cost(IS_{NM})|}{Cost(IS_{NM})}$ is used to compare GS_{MR} and IS_{NM} .

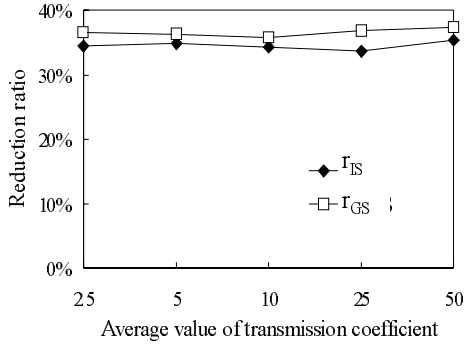
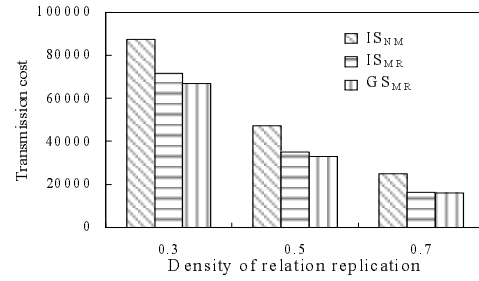


Figure 3. Performance evaluation of the average value of transmission coefficient |TC|

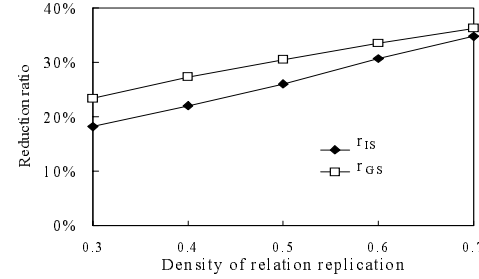
4.2. Performance Evaluation

Figure 3 shows that the relative performances of our proposed algorithms are not sensitive to the cost range of the network edge. Therefore, in the following experimental study, the use of $|TC| = 5$ will not affect the relative merits of the schemes evaluated. Furthermore, Figure 4a depicts that total communication cost of each proposed algorithm decreases with the growth of the density of relation replication p_{RSD} in network graphs. Clearly, more alternatives of relation selection will lead to a lower communication cost required. In addition, a larger p_{RSD} tends to favor the use of contributive replicated semijoins and/or contributive replicated joins, which will in turn increase the values of r_{IS} and r_{GS} . This fact is validated by the results in Figure 4b.

Performance study of relation replication for various network graphs is conducted in Figure 5. It can be seen that r_{IS} and r_{GS} in Figure 5b grow with the value of p_{NG} while total communication cost of each proposed algorithm decreases in Figure 5a. It shows that more alternative transmitting paths lead to a lower communication cost required. This means that the benefit of relation replication increases with the growth of the density of network graph. In addition, with the growth of $|G_N|$, the execution costs decrease as shown in Figure 6a, leading to more prominent performance improvement. The performance improvement, however, will be saturated as the number of network sites $|G_N|$ reaches g_{sat} , where the value of g_{sat} can be approximated to $\frac{|G_Q|}{P_{NG} \times P_{RSD}} = 20$ empirically in Figure 6b. Intuitively, even though the relations required are more widely distributed in the network, those replicated relations located far away from the final site will be of little impact to reduce the transmission cost. That is to say, if we want to further decrease the transmission cost of distributed query processing with replicated relations, p_{NG} , p_{RSD} , and $|G_N|$ should be increased to overcome such a saturation point.

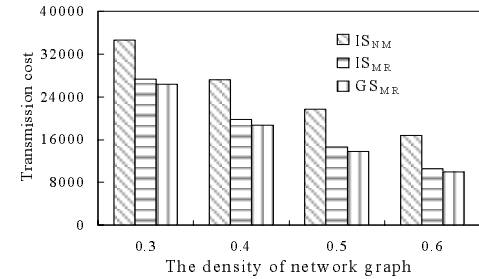


(a) The transmission cost of IS_{NM} , IS_{MR} and GS_{MR}

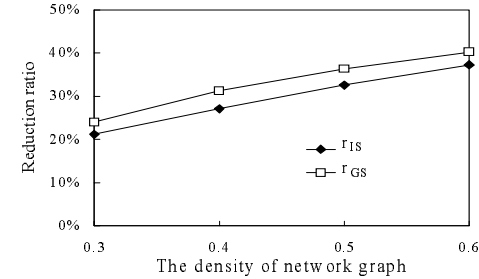


(b) The reduction ratio of r_{IS} and r_{GS}

Figure 4. Performance evaluation with various p_{RSD}

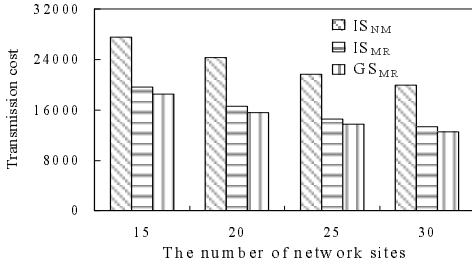


(a) The transmission cost of IS_{NM} , IS_{MR} and GS_{MR}

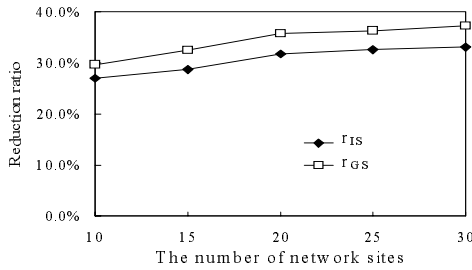


(b) The reduction ratio of r_{IS} and r_{GS}

Figure 5. Performance evaluation with various density of network graph



(a) The transmission cost of IS_{NM}, IS_{MR}, and GS_{MR}



(b) The reduction ratio of r_{IS} and r_{GS}

Figure 6. Performance evaluation with various network sites $|G_N|$

5. Conclusion

We studied in this paper the subject of exploring the effects of relation replication and network characteristics to optimize the execution of distributed query processing. Our solution procedure was composed of three consecutive steps, namely relation selection, join sequence scheduling and merge processing. Our results showed that the approach of interleaving a join sequence with contributive replicated semijoins/joins is not only efficient but also effective in reducing the total amount of data transmission cost required to process distributed queries.

6. Acknowledgment

The authors are supported in part by the Ministry of Education Project No. 89-E-FA06-2-4-7 and the National Science Council, Project No. NSC 89-2219-E-002-028 and NSC 89-2218-E-002-028, Taiwan, Republic of China.

References

- [1] G. Bojan and M. M. Qutaibah. Combinatorial Optimization of Distributed Queries. *Transactions on Knowledge and Data Engineering*, pages 915–927, December 1995.
- [2] J. W. Byers, M. Luby, and M. Mitzenmacher. Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed up Downloads. *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, pages 275–283, Jan. 1999.
- [3] M.-S. Chen and P. S. Yu. Combining Join and Semijoin Operations for Distributed Query Processing. *IEEE Transactions on Knowledge and Data Engineering*, 5(3):534–542, June 1993.
- [4] M.-S. Chen and P. S. Yu. A Graph Theoretical Approach to Determine a Join Reducer Sequence in Distributed Query Processing. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):152–165, February 1994.
- [5] M.-S. Chen, P. S. Yu, and K.-L. Wu. Optimization of Parallel Execution for Multi-Join Queries. *IEEE Transactions on Knowledge and Data Engineering*, 8(3):416–428, June 1996.
- [6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts London, England, 1989.
- [7] K. A. Hua, Y. L. Lo, and H. Young. Considering Data Skew Factor in Multi-way Join Query Optimization for Parallel Execution. *VLDB Journal*, 2(3):303–330, July 1993.
- [8] G. S. Jung, Q. M. Malluhi, and W. G. Brown. A Scheme for High-performance Data Delivery in the Web Environment. *Parallel and Distributed Systems, 1998. Proceedings. 1998 International Conference*, pages 210–217, 1998.
- [9] D. Kossmann. The State of the Art in Distributed Query Processing. In *ACM Computing Survey*, September 2000.
- [10] A. B. Stephens, Y. Yesha, and K. E. Humenik. Optimal Allocation for Partially Replicated Database Systems on Ring Networks. *IEEE Transactions on Knowledge and Data Engineering*, pages 975–982, 1994.
- [11] C. Wang and M.-S. Chen. On the Complexity of Distributed Query Optimization. *IEEE Transactions on Knowledge and Data Engineering*, pages 650–662, 1996.
- [12] J. L. Wolf, D. M. Dias, and P. S. Yu. A Parallel Sort Merge Join Algorithm for Managing Data Skew. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):70–86, January 1993.
- [13] Z. Xie and J. Han. Join index hierarchies for supporting efficient navigations in object-oriented databases. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pages 522–533, Santiago, Chile, September 1994.
- [14] C. T. Yu and C. C. Chang. Distributed Query Processing. *ACM Computing Surveys*, 16(4):399–433, December 1984.
- [15] C. T. Yu and W. Meng. *Principle of Database Query Processing for Advanced Applications*. Morgan Kaufmann Publisher, Inc. San Francisco, California, 1997.