IMPROVING SEMI-JOIN EVALUATION IN QUERY PROCESSING

Ekow J. Otoo[1]
Doron Rotem[2]
Nicola Santoro[1]

SCS-TR-78

July 1985

[1]School of Computer Science, Carleton University, Ottawa, Ont. Canada K1S 5B6

[2]Lawrence Berkeley Laboratories, University of California, Berkeley, Calif.

[3]School of Computer Science, Carleton University, Ottawa, Ontario, Canada K1S 5B6

Improving Semi-join Evaluation in Distributed Query Processing

Ekow J. Otoo [1], Doron Rotem[2] and Nicola Santoro[1]

Abstract

The semi-join operation constitutes the fundamental operation for reducing the volume of data transferred between sites in a distributed query processing. We present a method called "augmented semi-join" evaluation technique that minimizes the volume of data exchanged between sites in query processing algorithms that relying on semi-join computations. The technique performs some initial preprocessing of the common join-attribute values of relations stored at different sites in such a way that each site can determine the tuples of the various relations that participate in the join computation for all subsequent queries that compute joins. Let $|\sigma_Q|$ denote the total volume of intersite data transferred to achieve complete reduction of the relations referenced in a query Q by using a semi-join expression σ_Q. Let $|\tilde{\sigma}_Q|$ denote the volume of data exchanged in achieving the same reduction with an augmented semi-join evaluation. Then we have that $|\tilde{\sigma}_Q| \leq 0.5|\sigma_Q|$ always. For some special queries e.g., simple queries, the query can be completely evaluated without semi-join computations, i.e., $|\tilde{\sigma}_Q| = 0$.

1. Introduction

The cost of processing queries in a distributed database system is proportional to the total volume of intersite data transfers. As such, distributed query optimization strategies that minimize either total response time or total communication times strive to minimize the amount of data exchanged between sites. We present in this paper an approach to reducing the data transfers required in processing distributed query. The technique can be incorporated in any distributed query optimization strategy to reduce both total response time and total query evaluation cost. The environment being considered is that of a distributed relational database management system in which relations are stored in geographically dispersed locations and queries initiated at a single site access the distributed relations.

Consider a collection of distributed relations. Given a query at a particular site, a strategy for processing the query involves first, removing redundant tuples from each of the relations referenced

[1]School of Computer Science, Carleton University, Ottawa, Ont. Canada, K1S 5B6.

[2]Lawrence Berkeley Laboratories, University of California, Berkeley, California.

in the query that reside at site distinct from the request site, by local processing, and then transmitting the reduced relations to one designated site. The response to the query is finally computed at this designated site from which the response is transmitted to the request site, if this is different. The semi-join operation [2,4,7,8,9,14], has so far been the primary data reducing operation used.

Queries that access data in a distributed relational database system consist predominantly of the relational operators : select, project, and the join [11,12,22]. Consequently, a number of early works on query optimization techniques consider only these three basic relational operators [1,2,8,15,17]. We use the term *join* generally to mean the equi-join except when the join attribute names are the same in which case we mean the natural-join.

The selection and projection, without duplicate retention, generally reduce the size of the operand relations. The application of these operators as early as possible helps to reduce the size of intermediate relations during query processing. This means that the primary concern in reducing the amount of data involved to evaluate a query expression is that of reducing the data required in the join computations. This is evident in the distributed query processing techniques presented in [1,3,7,6,12,14,17,23]. Although the principal data reducing operation is the semi-join, its efficient application is however limited to tree queries [2,4,7,9,23,24].

Computing the semi-join operation requires the shipment of join attribute values from one site to another. Such data movement may not always be cost beneficial to the overall query processing cost. Given a tree query we propose a technique for evaluating semi-joins that requires transfers of join attribute values only if it is guaranteed to reduce the volume of data at the destination site. We introduce the main idea through a simple example.

Let R(A,B) and S(B.C) be two relation schemes located at two separate sites. The semi-join of R by S, denoted $R \ltimes S$, is the join of R and S projected on the attributes of R. Suppose R is located at a site 1, and S at a site 2, and assume a query at the site 1 requests the join of R and S. One way of computing the join using semi-joins is to ship the B attribute values of R, $\pi_B(R)$, from site 1 to site 2. A reduced relation $S' = S \bowtie \pi_B(R)$ can then be derived and transmitted to site 1. Hopefully the reduction in the cost of transmitting only the relevant tuples of S more than compensates for the cost of transmitting the join attribute values from site 1 to site 2. Computing the join of R ans S through the use of semi-joins as above, is not always guaranteed to be cost beneficial. What is more, the cost of transmitting the projection $\pi_B(R)$, will be incurred each time such a query is evaluated. We will call the above method of computing the semi-join a *conventional* *semi-join* evaluation technique. Detecting the candidate tuples of relation S that can form a join with tuples of relation R at site 1 is essentially a problem of determining the tuple in S whose join domain values intersect the join domain values of R.

Our proposal for reducing the data transfer in the preceding example involves retaining, for each join domain of each relation at any site, some information on those relations at other sites, that will subsequently form a join with it. We explore two methods of retaining such extra information, with respect to the storage requirement, preprocessing cost, update cost and the reducing effect on semi-join evaluation.

In the first method, the relations at each site are augmented with tables of 0/1 bit vectors. For each distinct join attribute A_j, the values are stored as a table of vectors $\langle x_1, b_2, \ldots, b_K \rangle$, where x_1 denotes a value of attribute A_j, and each $b_j, j = 2, 3, \ldots K$, is a 0/1 bit. The bit b_j is set to 1 if a relation R_{i_j} at another site has a value x_1 in its join attribute. In the second method, the values of any join attribute are retained as a collection of disjoint subsets that may be directly addressed. Consider two relations R(A, B) at site 1 and S(B,C) at site 2. We store at site 1 the two subsets $r^{\cap} = \pi_B(R) \cap \pi_B(S)$ and $r^{-} = \pi_B(R) - \pi_B(S)$. Similarly at site 2, we store $s^{\cap} = \pi_B(S) \cap \pi_B(R)$ and $s^{-} = \pi_B(S) - \pi_B(R)$. To compute the join of R and S at site 1, we only need to send to site 2, a message consisting of the query. The semi-join $S \ltimes R$ can be evaluated at site 2 giving $S' = S \bowtie s^{\cap}$. The reduced relation S', is then shipped to site 1. In either of the two methods of storing the augmented information the table construction, and the disjoint subset partitioning may be performed only once as setup activity, except for subsequent updates to the relations.

Using either methods, all subsequent semi-join computation, in the example above , is done with no join attribute transfers. Since the relations at each site had to be augmented with some information, we call this technique the *augmented semi-join* method. This paper extends the augmented semi-join method to handle queries that compute joins over multiple relations and over multiple attributes. The main results in this paper are essentially the following. We show that

1. Simple queries [1,7,12], in a distributed database can be processed without transferring join attribute values.
2. In any query optimization strategy utilizing conventional semi-joins, the amount of data transferred is at least twice that required in an augmented semi-join the evaluation.

The outline of the rest of this paper is as follows. In section 2, we define the terminology and notation, and state the assumptions relevant to our subsequent discussions. The different methods for storing the relevant information are described in section 3. The application of the augmented join domain values in minimizing the data transfers required in a semi-join computation is discussed in section 4. In section 5 we discuss the reducing power of the augmented semi-join approach. We present a summary of the main ideas in section 6.

2. Some Basic Relational Database Concepts

2.1 Relational terminology.

We assume that the reader is familiar with the relational data model [11,12,19,22]. We only present a summary of the basic concepts here. A domain D, is a set of values. An attribute A, is a named domain of a relation. A relation scheme denoted by $\mathbf{R}_i(A_1, A_2, \ldots, A_k)$ is a description of a relation consisting of a relation name $\mathbf{R}$ and a list of attributes $A_1, A_2, \ldots, A_k$. An instance R, of a relation scheme $\mathbf{R}$, is the current value of the relation.

A relation instance is a subset of the cartesian product of the domains of the attributes in the relation scheme, i.e., $R(A_1, A_2, \ldots, A_k) \subseteq D_1 \times D_2 \times \ldots \times D_k$. A tuple $t \in R$, is an element (or row) of a relation. The attribute A of the relation R will at times be denoted by R.A. Similarly the corresponding value of attribute A in a tuple t is denoted by $t.A$. A database scheme S, is a collection of relation schemes $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_n$. A database state S, is a collection of relation instances $R_1, R_2, \ldots R_n$. A database is said to be distributed if instances of the relations are resident at separate geographical locations, each under the control of their respective local database management system.

2.2 Relational Operators.

We assume that queries are expressed using primarily the three relational operators of select, project and join. Let $\mathcal{F}$ be a predicate formula involving constants, attribute values, the arithmetic comparison operators $\{=, \neq, <, \leq, >, \geq\}$, and logical connectives $\{\wedge, \vee, \neg\}$. The *selection*, $\sigma_{\mathcal{F}}(R)$ is the set of tuples $t \in R$ such that values of t satisfies $\mathcal{F}$. Let $X \subseteq \{A_1, A_2, \ldots A_k\}$. The *projection* of R on X, written $\pi_X(R)$, is the relation obtained by restricting the columns of R to those in X and eliminating duplicates. The *equi-join* of two relations $R_i(A, B)$ and $R_j(C, D)$ is defined as

$$R_i \underset{B=C}{\bowtie} R_j = \sigma_{B=C}(R_i \times R_j),$$

where $R_i \times R_j$ denote the cartesian product. An equi-join over all common attribute names in R_i and R_j with the duplicate columns eliminated is called a *natural-join*. In this case, we use the symbol $\bowtie$ to denote the ***natural-join***. The term "join" will be used to mean the natural-join in this paper unless stated otherwise, since any equi-join query can be transformed into a natural-join query by proper renaming of the attributes.

The semi-join of a relation R_i on A and R_j on B, written $R_i \ltimes R_j$, is the equi-join $R_i \underset{A\ B}{\bowtie} R_j$, projected back on the attributes of R_i. Similarly the natural semi-join of R_i by R_j is defined as the natural-join of R_i and R_j projected back onto the attributes of R_i.

2.3 Queries.

A query Q(S) on the database state $S = \{R_1, R_2, \ldots R_n\}$, is a function defined by the relational algebraic expression $Q(S) = \pi_{\mathcal{L}}\sigma_{\mathcal{F}}(\bowtie_{i_j} R_{i_j})$ for $i_j \in \{1, 2, \ldots, n\}$, $j = 1, 2, \ldots n$. The symbols π and σ denote the projection and selection operators respectively, and $\mathcal{L}$, called the target list, specifies the attributes to appear in the response. The select operator is inherently data reducing and will be assumed to be applied locally at zero cost, whenever possible, prior to data transfer. The projection onto the target list can always be applied at the result site. Hence in so far as minimizing the data transfers involved to answer a query, we need only consider the problem of computing the join over the set of distributed relations. In this respect a query can be considered simply as a conjunction of equi-join clauses : $Q(S) = \bigwedge_{i,j}(R_i.A_{ij} = R_j.A_{ij})$.

Two queries Q_1 and Q_2 are said to be equivalent if there response relations $Q_1(S)$ and $Q_2(S)$ are equal for all database states S. In [2] the concept of a query graph G_Q, is introduced. A query graph allows the characterization of queries as either tree queries or cyclic queries [2,4,7,8,13,16,24], according to whether the query graph G_Q is acyclic or cyclic. A query graph G_Q, corresponding to a query Q, is a labeled undirected graph $G_Q(\mathcal{V}_Q, \mathcal{E}_Q)$, where $\mathcal{V}_Q$ is the set of vertices and $\mathcal{E}_Q$ is the set of labeled edges. A vertex in $\mathcal{V}_Q$ is a relation R_i referenced in Q. Two vertices corresponding to R_i and R_j are connected by an edge $(i, j) \in \mathcal{E}_Q$ if there is a clause "$R_i.A = R_j.A$" in Q. The labels of the edge (i,j), is the union of all such A's. A query is called a tree query if either its query graph is a tree or it is equivalent to a query whose query graph is a tree, otherwise it is a cyclic query.

For some illustrative examples, consider the relations $R_1(A, B)$, $R_2(A, C)$, $R_3(A, B, C)$ and

$R_4(A, B, C, E)$. Suppose we have the following queries.

$$Q_1 : (R_1.A = R_2.A) \wedge (R_1.A = R_3.A) \wedge (R_1.A = R_4.A),$$
$$Q_2 : (R_1.A = R_2.A) \wedge (R_2.C = R_3.C) \wedge (R_3.B = R_4.B),$$
$$Q_3 : (R_1.A = R_2.A) \wedge (R_2.C = R_3.C) \wedge (R_3.C = R_4.C) \wedge (R_4.A = R_1.A),$$
$$Q_4 : (R_1.A = R_2.A) \wedge (R_2.C = R_3.C) \wedge (R_3.B = R_1.B).$$

The query trees corresponding to Q_1, Q_2, Q_3 and Q_4 are shown in Figures 1.1, 1.2, 1.3a and 1.4. Queries of the form of Q_1 are called simple queries [1,2,7,9,15,17]. Those of the form of Q_2 are called chain queries [2,7,9,16,23,24]. The query graphs in Figures 1.3a and 1.4 contain cycles but Q_3 is equivalent to a query whose graph is as shown in Figure 1.3b. Hence Q_3 is not a cyclic query. However, Q_4 is a true cyclic query.

Algorithms for detecting whether a given query is a tree query or a cyclic query are presented in [2,23]. We consider only tree queries with the understanding that a cyclic query will be reduced first into to a tree query using the transformation techniques in [16, 24] before processing. In general, a given arbitrary query may have its query graph disconnected. We assume that queries have query graphs that are connected since disconnected query graphs can be considered as a collection of separate connected query graphs. When relations are stored with replication, we assume that the the distinct copies of the relations to be used in the processing the queries are preselected.

2.4 Semi-join Expressions.

For a given query a semi-join expression, written $EXP(Q)$, is a set of relevant semi-joins required to reduce each of the referenced relations $R_i \in Q$, to the minimum set of tuples R_i', required for the correct evaluation of Q. The semi-join expression is defined recursively as follows. Let $EXP_i(Q)$ be a subexpression of $EXP(Q)$, then

1. $R_i \in EXP_i(Q)$ for i = 1,2, ..., n;
2. if $(i, j) \in \mathcal{E}_Q$ then for any $E_i \in EXP_i(Q)$ and $E_j \in EXP_j(Q)$, $(E_i \ltimes E_j) \in EXP_i(Q)$ and $(E_j \ltimes E_i) \in EXP_j(Q)$;
3. $EXP(Q) = \bigcup_{i=1}^{n} EXP_i(Q)$.

Each relation in an expression E is a relation scheme $\mathbf{R}_i$. The result ϱ, of evaluating the expression E (i.e., $\varrho = eval(E)$), is an assignment of each relation instance R_i corresponding to $\mathbf{R}_i \in E$, to the expression. Each R_i is reduced, if possible, to R_i' through semi-join operations.

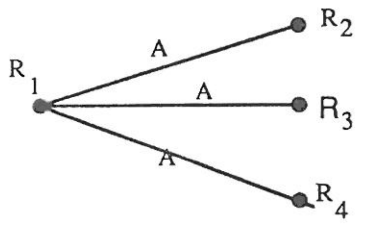


Figure 1.1: A simple query graph

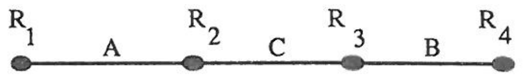


Figure 1.2 : A chain query graph

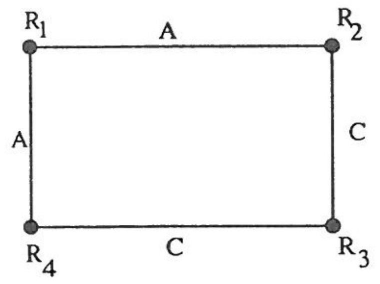


Figure 1.3a An apparent cyclic query

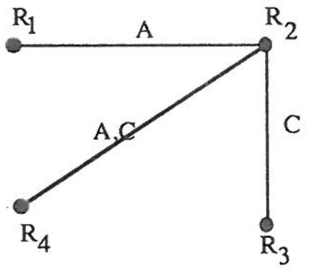


Figure 1.3b :
A tree query graph equivalent to that of Figure 1.3a

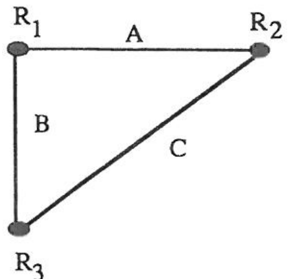


Figure 1.4 : A cyclic query graph

The motivation for the formulation of the semi-join expression $EXP(Q)$ with respect to a query Q is the following. Given a query Q which requires the computation of a join $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$ at a site i. The result of evaluating the expression $EXP(Q)$, is the derivation of the reduced relation R_i', at the respective sites. These reduced relations are necessary and sufficient for the correct computation of the join when shipped to the result site i.

2.5 Cost Measure.

The cost measure of interest in this paper is the volume of data transferred between sites to generate full reduction of the relevant relations required to compute the joins in a query. Since the cost of evaluating the query is proportional to the volume of data transferred, and consequently proportional to the volume of join attributes exchanged between sites for the semi-join computation, an evaluation of the semi-join expressions with minimum data transfers constitutes one aspect of an optimal distributed query processing strategy. The order of evaluating semi-joins in query processing, is generally called a strategy. Various criteria and heuristics have been proposed for deriving good strategies in query optimization algorithms [1, 2, 3, 7, 8, 14, 17]. The main thesis we advance in this paper is that any optimization techniques using conventional semi-joins can be improved by the use of augmented semi-join techniques. The savings in the cost of data transfers is at least half the cost of the conventional semi-join method.

3. Distributed Set Operations: Intersection, Union and Difference

Underlying the method of augmented semi-join evaluation are methods for representing and manipulating distributed sets. Let $\Gamma = \{S_1, S_2, \ldots, S_n\}$, be collection of subsets of a universe U. For notational convenience we shall assume that the subset S_i is stored at site i. We present briefly two strategies for computing the set operations: the Bitmap method and the Disjoint Subset Partioning method. The two techniques rely on a particular local representation of a set at a site. A detailed description and analysis of these methods can be found in [20]. One approach to performing set intersection, that reduces the data exchanged between sites, involves the transmission of a bit vector. This approach, which we designate as the *Filter Method*, can be viewed as a communication protocol for set operations and may be used to further enhance our proposal to be described subsequently. We first give an outline of the Filter method.

3.1 The Filter Method.

Suppose we desire the intersection at site i, of two sets S_i and S_j. The idea of the filter approach consists of choosing K randomizing hash functions $g_1, g_2, \ldots, g_K$, and a bit vector $B = \langle b_1, b_2, \ldots, b_N \rangle$, where $N \gg \max(|S_i|, |S_j|)$ but is small enough to be cost beneficial. The vector B is initialized to zeroes. For each element $x \in S_i$, K bit positions are computed as $p = g_j(x)$ for $j = 1, 2, \ldots, K$, and each bit b_p is set to 1. The bit vector B is then shipped to site j. Suppose the elements of the sets have an average size of w bits, then for the method to be cost beneficial, to ship B, we must have $w * |S_i| \leq N$.

At site j, using each element $y \in S_j$, bit positions are computed using the same hash functions and the corresponding bits are inspected. If any position addressed by an element is zero, then that element is definitely not in the intersection and can be eliminated. On the other hand if all K positions inspected are set to 1, then the element y is a candidate in the intersection. The set of candidate elements identified as being in the intersection include those that have been wrongly identified. Let $S_j^{\cap}$ denote the set of elements identified as the probable elements in the intersection, then $S_j^{\cap} \supseteq (S_i \cap S_j)$.

The filter method outlined above is essentially the idea of the use of the Bloom Filter [5], hence the name. Let $N_i = |S_i|, N_j = |S_j|$ and let $n = |S_i \cap S_j|$ so that $|S_j - (S_i \cap S_j)| = N_j - n$. The filtering error at site j is given by the product of the probability that an element y is not in the intersection and all K positions inspected by the $G_j(y)$ was set to one. The filtering error is given by

$$Prob(error) = (\frac{N_j - n}{N_j})[1 - (1 - \frac{1}{n})^{KN_i}]^K \approx (1 - \frac{n}{N_j})[1 - e^{-KN_i/n}]^K.$$

This error may be reduced by reversing the process from site j to site 1 with a new initialized vector. The values now used to hash into the new vector are the identified candidates $y \in S_j^{\cap}$. The process may be repeated for a number of times as long as there is a reduction in the size of elements identified and the cumulative cost of transmitting the bit vector is less than that of sending the candidate values. The method is similar in may ways to the probabilistic algorithm for the set intersection problem in [18]. In determining the reduced relation for a join computation one round of such a bit vector exchanges between any two sites in a schedule should suffice. The idea is easily extended to more than 2 sets on different sites.

3.2 The Bitmap Method

The first of our two augmentation techniques is as follows. Each element in the set S_i stored at location i is augmented with a bit map that indicates the presence or absence of the element in the sets at the other sites. The sets may be stored as a table of ordered tuples $\langle x_q, b_1, b_2, \ldots, b_{n-1} \rangle$. where the number of distributed sets is n. The first component x_q in each tuple specifies the element, and every other component b_j is set to 1 if $x_q \in S_j$ for $j = 1, 2, \ldots n, j \neq i$, and 0 otherwise. The technique is better illustrated through an example.

Consider the three sets $S_1 = \{a, b, e, f, g, m, n, q\}$, $S_2 = \{a, e, f, g, o, \ p, r, u, v\}$ and $S_3 = \{e, f, p, r, m, q, v\}$. These may be stored as the tables in Figure 3.1 at their respective sites. From each site the intersection and difference of that set with any combination of the rest can be computed. The overhead incurred is the storage cost for the bitmap and the preprocessing cost to inform each site of what elements every other site has. This cost is incurred only once and does not feature in all subsequent queries, except for subsequent updates. The extra storage required to retain the bitmap at site i is $(n - 1) * N_i$ bits, where N_i is the cardinality of the set S_i.

Figure 3.1. The representaion of sets S_1, S_2 and S_3 augmented with bitmaps.

S_1

Value	Site 2	Site 3
a	1	0
b	0	0
e	1	1
f	1	1
g	1	0
m	0	1
n	0	0
q	0	1

S_2

Value	Site 1	Site 3
a	1	0
e	1	1
f	1	1
g	1	0
o	0	0
p	0	1
r	0	1
u	0	0
v	0	1

S_3

Value	Site 1	Site 2
e	1	0
f	1	1
p	0	1
r	0	1
m	1	0
q	1	0
v	0	1

3.3 The Disjoint Subset Partitioning Method.

This technique requires a more detailed description than the former due to its unusual structure and direct access addressing mechanism.

3.3.1 Partitioning Procedure.

The idea is to store each set S_i as disjoint subsets $\{Z_1^i, Z_2^i, \ldots, Z_m^i\}$, such that set operations of intersection, difference and union over Γ, can be computed simply as a union of disjoint subsets with the least amount of data transfers. For the purpose of semi-join evaluation only the set intersection operations are relevant.

Each set S_i, is represented at the site i as disjoint subsets partitioned as follows. Let $Z_{0,1}^i$ denote S_i at site i, and let the rest of the collection $\Gamma - S_i$, be relabeled as $\{S_1^i, S_2^i, \ldots, S_{n-1}^i\}$. Then S_i may be represented as a collection of 2^n disjoint (possible empty) subsets $Z_{n-1,j}^i$, defined recursively using only set intersection and difference operators as follows

$$Z_{1,1}^i = S_i,$$
$$Z_{l,2j-1}^i = Z_{l-1,j}^i \cap S_l^i,$$
$$Z_{l,2j}^i = Z_{l-1,j}^i - S_l^i,$$
$$\text{for } 1 \leq l \leq n,\ 1 \leq j \leq 2^{l-1}.$$

The recursive set partitioning of the set S_i corresponds to a binary tree-like partitioning of the elements of this set. The root corresponds to the elements of S_i. Each level of the tree corresponds to the elements of the set S_l^i. If p_l is a node at level l with left and right siblings at level l-1, then the left child of p_l corresponds to the elements derived from the intersection of the elements corresponding to p_l and the set S_{l+1}^i. The right child of p_l represents the elements corresponding to the difference of elements of p_l and S_{l+1}^i.

As an example, consider the three sets $S_1 = \{a, b, e, f, g, m, n, q\}$, $S_2 = \{a, e, f, g, o, p, r, u, v\}$ and $S_3 = \{e, f, p, r, m, q, v\}$. The corresponding disjoint subsets generated for each are shown in the Figures 3.2a 3.2b and 3.2c.

Let T_i be a binary tree corresponding to the set-partitioning tree of the set S_i. We refer to T_i as the SP-tree of the set S_i. The characteristics of such a tree representation are summarized below.

Lemma 3.1. *The set of elements in S_i is given by the union of the subset of elements corresponding to the leaf nodes of T_i.*

$$S_i = \bigcup_{1 \leq j \leq 2^{n-1}} Z_{n-1,j}^i.$$

Call the nodes at level l of the SP-tree odd-labeled if they correspond to the subsets $Z_{l,2j-1}^i$ and even-labeled if they correspond to the subsets $Z_{l,2j}^i$, for $1 \leq j \leq 2^{l-1}$

Lemma 3.2. *The intersection S_i and S_l^i, for $1 \leq l \leq n-1$, is given by the union of subsets corresponding to the odd-labeled nodes at level l.*

$$S_i \cap S_l^i = \bigcup_{1 \leq j \leq 2^{l-1}} Z_{l,2j-1}^i.$$

Lemma 3.3. *The difference between S_i and S_l^i for $1 \leq l \leq n-1$ is given by the union of subsets of the elemented corresponding to the even-labeled nodes.*

$$S_i - S_l^i = \bigcup_{1 \leq j \leq 2^{l-1}} Z_{l,2j}^i.$$

Lemma 3.4. *The subsets of elements corresponding to the internal node p is derived as the union of the subsets of the elements corresponding to the leaf nodes of the subtree rooted at p.*

$$Z_{l,j}^i = \bigcup_{1 \leq k \leq 2^{n-l-1}} Z_{n-1,2^{n-l-1}(j-1)+k}^i.$$

Given a representation of each of the set S_i as a disjoint subset of elements derived using the set partitioning method, the union and the intersection of S_i and any other set S_k for $1 \leq k \leq n$, $k \neq i$, is computable entirely from a union of some subsets of S_i.

Figure 3.2a

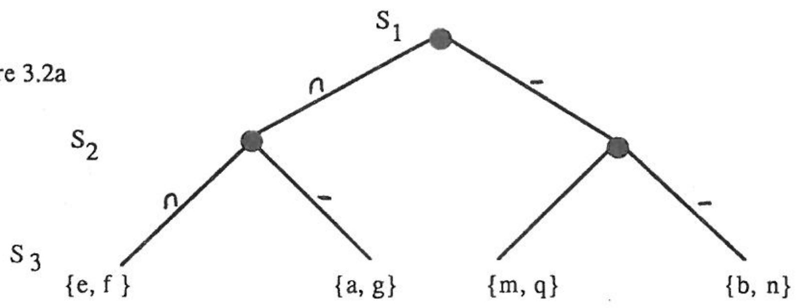


Figure 3.2b

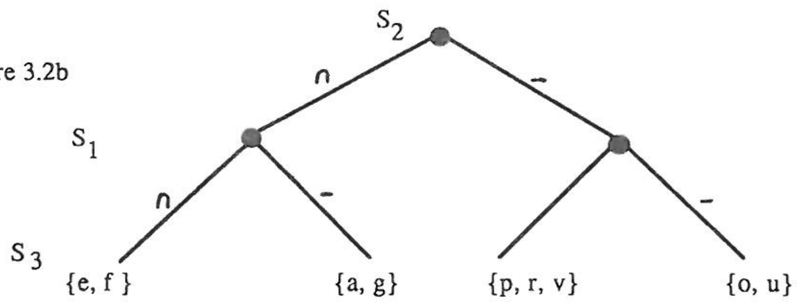
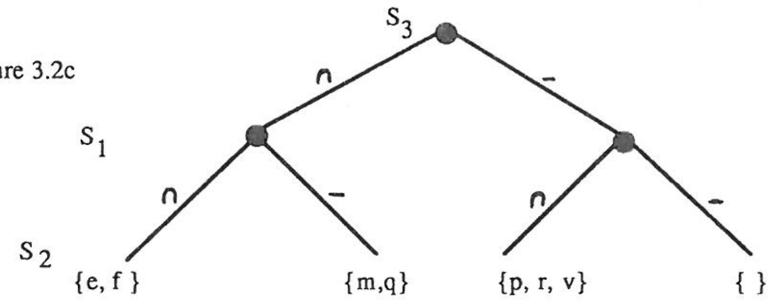


Figure 3.2c



Theorem 3.1. *Let S_l^i be the relabeling of the set S_k with respect to the site i. Then the intersection of S_i and S_k is computable by a union of disjoint subsets as*

$$S_i \cap S_k = S_i \cap S_l^i = \bigcup_{\substack{1 \le j \le 2^{l-1} \\ 1 \le r \le 2^{n-l-1}}} Z^i_{n-1,2^{n-l}(j-1)+r}.$$

The difference between S_i and S_k is computable by a union of disjoint subsets as

$$S_i - S_k = S_i - S_l^i = \bigcup_{\substack{1 \le j \le 2^{l-1} \\ 1 \le r \le 2^{n-l-1}}} Z^i_{n-1,2^{n-l-1}(2j-1)+r}.$$

The proof of Theorem 3.1 follows from the Lemmas 3.1 to 3.4.

The significance of Theorem 3.1 is that, once the cost of preprocessing is incurred in determining the tree at each site, subsequent set operations of union, difference and intersection of any arbitrary number of sets can be computed optimally (i.e. with minimum data transfer). Note that at a site i only the elements of the set S_i are stored.

3.2.2 Accessing the Disjoint Subsets.

Each of the disjoint subsets may be stored at the terminal nodes of an explicit tree structure and and accessed by traversing the corresponding binary tree. Alternatively, the subsets may be stored and and accessed by a direct access method. The direct addressing technique is elaborated here as follows Let T_i be the set-partitioning tree at site i. The globally distributed sets $S_1, S_2, \ldots, S_n$ are considered to be ordered according to the indices $1, 2, \ldots, n$. Then each node of the SP-tree T_i, can be defined by a formula $\xi^i_{l,j}$. The formula $\xi^i_{l,j}$ is composed of operands S_k for $1 \le k \le n$ and operators of set intersection and difference only. For instance in the example of Figure 3.2, the leaf node of the SP-tree $Z^1_{3,2}$ is defined as

$$Z^1_{3,2} = \{a, g\} = (S_1 \cap S_2) - S_3.$$

Similarly

$$Z^2_{3,3} = Z^3_{3,3} = \{p, r, v\} = (S_2 - S_1) \cap S_3 = (S_3 - S_1) \cap S_2.$$

Hence we have $\xi^1_{3,2} = (S_1 \cap S_2) - S_3$, $\xi^2_{3,3} = (S_2 - S_1) \cap S_3$ and $\xi^3_{3,3} = (S_3 - S_1) \cap S_2$.

We define a vector $V^i_{l,j}$ corresponding to the formula $\xi^i_{l,j}$ as follows.

$V^i_{l,j} = \langle \beta_1, \beta_2, \ldots \beta_n \rangle$; where $\beta_i = 1, \beta_k \in \{0, 1, X\}$ for $k = 1, 2, \ldots n, k \ne i$;

$$\beta_k = \begin{cases} 0 \text{ if } S_k \text{ is introduced in } \xi^i_{l,j} \text{ with an intersection ;} \\ 1 \text{ if } S_k \text{ is intoduced in } \xi^i_{l,j} \text{ with a difference ;} \\ X \text{ if } S_k \text{ is absent .} \end{cases}$$

If $V^i_{l,j} = \langle \beta_1, \ldots, \beta_r, \ldots, \beta_n \rangle$ is a vector corresponding to the formula $\xi^i_{l,j}$ with $\beta_r = X$, then $V^i_{l,j}$ is considered as being composed of two vectors, $V^i_{l+1,(2j-1)} = \langle \beta_1, \ldots, 0, \ldots, \beta_n \rangle$ and $V^i_{l+1,(2j)} = \langle \beta_1, \ldots, 1, \ldots, \beta_n \rangle$. This implies that $\xi^i_{l,j} = \xi^i_{l+1,2j-1} \cup \xi^i_{l+1,2j}$.

Lemma 3.5. *Given a collection of distributed sets $S_1, S_2, \ldots, S_n$, such that T_i is the SP-tree at the site i with leaf nodes corresponding to the formula $\xi^i_{n,j}$ for j = 1, 2, ..., 2^{n-1}: Then each formula $\xi^i_{n,j}$ is uniquely defined by a binary vector $V^i_{n,j} = \langle \beta_1, \beta_2, \ldots, \beta_n \rangle$ where $\beta_r \in \{0,1\}$ only.*

The Lemma 3.5 provides a means of directly storing and addressing disjoint subsets of S_i at the site i. Hence there is a a one-to-one mapping of the vectors $V^i_{n,j}$ onto the formulas $\xi^i_{n,j}$ that define the subsets of the elements at the leaf nodes of the SP-tree. The subset defined by the formula $\xi^i_{n,j}$ is stored at a location with the logical address j-1. We define the mapping function

$$\Phi_i : V^i_{n,j} \rightarrow \{0, 1, 2, \ldots, 2^{n-1} - 1\}$$

by the Theorem 3.3 below.

Theorem 3.2. *Let the subset defined by the formula $\xi^i_{n,j}$ be stored at the location j-1, and let the vector corresponding to the formula $\xi^i_{n,j}$ be $V^i_{n,j} = \langle \beta_1, \beta_2, \ldots, \beta_n \rangle$. Then the mapping Φ_i, of the vectors $V^i_{n,j}$ onto the locations $\{0, 1, 2, \ldots, 2^{n-1} - 1\}$ is defined by*

$$\Phi_i(\langle \beta_1, \ldots \beta_n \rangle) = \sum_{1 \le j < i} \beta_j 2^{n-j-1} + \sum_{i < j \le n} \beta_j 2^{n-j}.$$

The proof of Theorem 3.2 employs the Lemma 3.5. Any expression involving intersection and difference operators only can be answered from a single. As an illustration suppose we desire at site 1 the set of elements given by expression

(1.) $$E_1 = ((S_1 \cup S_2) \cap S_3) - (S_4 \cup S_5);$$

The expression E_1 is transformed to E_1' where

$$\begin{aligned} E_1' &= ((S_1 \cap S_3) \cup (S_2 \cap S_3) - (S_4 \cup S_5), \\ &= (S_1 \cap S_3 - S_4 - S_5) \cup (S_2 \cap S_3 - S_4 - S_5), \\ &= (S_3 \cap S_1 - S_4 - S_5) \cup (S_3 \cap S_2 - S_4 - S_5). \end{aligned}$$

The elements of each of the formulas $(S_3 \cap S_1 - S_4 - S_6)$ and $S_3 \cap S_2 - S_4 - S_5)$ can be computed at the site of S_3 and the result transferred to site 1. On the other hand the first term

Table 1 : Setup and Communication Costs of the Bitmap and Subset Partitioning Method.

	Total Setup Cost		Additional Storage (in Bits)	
	Communication (in bits)	Local Processing	Per Site	Total
Bitmap Method	$O(nwN)$	$O(N_i \log N_i)$	nN_i	nN
Subset Partitioning Method	$O(nwN)$	$O(N_i \log N_i)$	χ_i	$\sum_{i=1}^{n} \chi_i$

$N = \Sigma N_i$
w = average number of bits per value
$\chi_i = \min \{ 2^n , N_i \log N_i)$

Table 2: Data Transfers in Bits.

		Operations to be Computed at Site of S_1			
	Method	$S_1 \cap S_2$	$S_1 \cup S_2$	$S_1 - S_2$	$S_2 - S_1$
Conventional Method	Normal	$\|S_2\|w$	$\|S_2\|w$	$\|S_2\|w$	$\|S_2\|w$
	With Bloom Filter *	$2N + S^n w$	$2N + S^n w$	$2N + S^n w$	$2N + S^n w$
Augmented Methods	Using Bitmap	0	0	0	$\|S_1 - S_2\|w$
	Using Disjoint Subsets	0	0	0	$\|S_1 - S_2\|w$

$N > \max \{ | S_1 |, | S_2 |\}$

* 2N bits suffices if only an approximate solution is sought.

can be computed at sit 1 and second term computed at site 2 and the result of the latter shipped to site 1. This second strategy of computing E' transfers less volume of data unless the the result of $(S_1 \cap S_3 - S_4 - S_5) = \phi$. A consistent set of rules for transforming any arbitrary expression to one involving a union of disjoint terms is necessary. We do not address this problem here. The interested reader may consult [10, 21] for similar concepts used in the optimization of query expressions. Proofs of all theorems and lemmas can be found in [20]. A summary of the three methods of handling distributed set operations is given in the Table 1.

Evaluating Queries in Distributed Databases

4.1. Representation of Join Domains.

Consider the relations $R_1, R_2, \ldots, R_n$ that are assumed to be located at different sites. For simplicity we assume that only a single copy of each relation exists and corresponding pairs of join attributes have common names. More than one attribute of the same relation may participate in the join expression required to answer the query.

Let A be an attribute name common to a set of relations. Call the set of relations having a common attribute of A the *relevant set* of A, denoted as $\rho(A)$. For example if P(A, B), R(C, D) and S(A, C, E) are a set of distributed relations, then $\rho(A) = \{P, S\}$, and $\rho(C) = \{R, S\}$. An attribute participates in a join computation only if the cardinality of the relevant set is greater than one, i.e $|\rho(A)| > 1$. If semi-joins are to be computed by the filtering method, no special information is required. We concentrate on the two other methods of augmenting the join domains to facilitate the semi-join computations.

The representation of the join attributes values is carried out as follows. In the case of the bitmap augmentation, the join attribute values for each relation may be either directly augmented with the bitmap vector or the distinct values separately stored with the bitmap information in the for of tables as described in the preceding section. To represent the join domain values using the disjoin partitioning method we do the following. For each attribute A having a relevant set $\{R_{i_1}, R_{i_2}, \ldots R_{i_k}\}$, the projections of each relation on A are derived. Let these be denoted as $\pi_A(R_{i_1})$, $\pi_A(R_{i_2}) \ldots, \pi_A(R_{i_k})$. Each $\pi_A(R_{i_j})$ may now be stored as disjoint subsets using the set intersection and difference partitioning technique of the preceding section. The evaluation of the semi-join expression can now be perceived as the derivation of the set of tuples that satisfy the values in the intersection of the join attributes of the relevant relations.

4.2. Distributed Query Evaluation.

The procedure of evaluating a semi-join when the join domains are augmented with the bitmap is the same as when they are augmented with the partitioned subsets of their values. We will elaborate on the semi-join evaluation method on the assumption that we have the join attribute values partitioned into disjoint subsets. Consider a tree query Q defined by a conjunction of equi-join clauses of the form $R_i.A = R_j.A$. We assume that the effect of all local processing reduces the attributes of relations referred to in Q to those that are either in the target list of the result or are join attributes. Let the query graph of Q be $G_Q = (\mathcal{V}_Q, \mathcal{E}_Q)$, where $\mathcal{V}_Q = \{R_{i_1}, R_{i_2}, \ldots R_{i_k}$, i.e., the set of relations referenced in Q. A semi-join expression $EXP(Q)$ for the query Q is composed of the union of the semi-join expressions of the nodes $EXP_i(Q)$ computed at each node. The evaluation cost of the expression EXP(Q) is the sum of the evaluation cost of each expressions $EXP_i(Q)$. The result of the evaluation is the collection of reduced relations that must be shipped to the result site to complete the query.

In general an optimum schedule that gives the minimum data will be needed to define the evaluation sequence of the semi-join expression. Heuristic for determining a good schedule under the conventional semi-join evaluation process are discussed in [1,2,6,9,15,24]. Using the augmented information each site can determine a possible effective reduced size the required relation from the query, without join attribute transfers. To generate a full reduced relation, some information still needs to be exchanged, and a good schedule achieves fully reduced relations with minimum volume of data transfers.

We use the following heuristic for generating an evaluation sequence. Given a query graph G_Q, the site of the node $\mathcal{V}$ can identify itself as either a leaf node or non-leaf node. It is a leaf node it is linked by a single edge, otherwise it is a non-leaf node. Any non-leaf node with maximum number of edges is taken as the root node of the query graph.

The evaluation of each of the expressions $EXP_i(Q)$ is carried out in two phases. The first phase consists of the reduction of a relation R_i if possible using the join attributes of all relations R_j, where R_j is a descendent of R_i in the query graph. The second phase of computing $EXP_i(Q)$, is a further reduction of R_i where the join attributes used are those of the ancestors R_j, of R_i. The two phase evaluation of the semi-join expression is analogous to the UP and DOWN execution of semi-join programs in [2].

The process is illustrated with the query graph shown in Figure 4.1. The labels of the edges denote the common join attributes. Let $EXP'_i(Q)$ denote the intermediate expression evaluated in the first phase at each site of the node R_i. Then the evaluation process is a s follows.

Phase 1.

$$EXP'_2(Q) = (R_2 \ltimes R_5) \ltimes R_6,$$

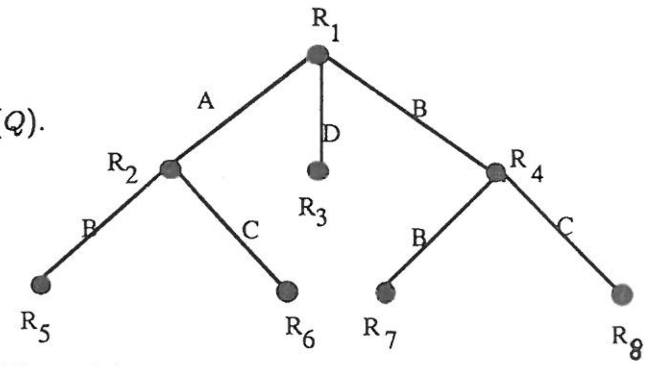
$$EXP'_4(Q) = (R_4 \ltimes R_7) \ltimes R_8;$$

$$EXP'_1(Q) = ((R_1 \ltimes EXP'_2(Q)) \ltimes R_3) \ltimes EXP'_4(Q).$$

Phase 2.

$$EXP_2(Q) = EXP'_2(Q) \ltimes EXP_1(Q);$$

$$EXP_3(Q) = EXP'_3(Q) \ltimes EXP_1(Q);$$



Figue 4.1: An example query graph.

$EXP_4(Q) = EXP_4'(Q) \ltimes EXP_1(Q);$

$EXP_5(Q) = R_5 \ltimes EXP_2(Q);$

$EXP_6(Q) = R_6 \ltimes EXP_2(Q);$

$EXP_7(Q) = R_7 \ltimes EXP_4(Q);$

$EXP_8(Q) = R_8 \ltimes EXP_4(Q).$

To appreciate the effect of the stored information on the join attributes we focus our attention on the data transfers required to reduce R_4 in the above example. At the site of R_4, the following subsets are known: $b_4 = \pi_B(R_4) \cap \pi_B(R_7) \cap \pi_B(R_7)$ and $c_4 = \pi_C(R_4) \cap \pi(R_8)$. Similarly, at the site of R_7 we have $b_7 = b_4$, at the site of R_8 we have $c_8 = c_4$ and at the site of R_1, we have $b_1 = b_4 = b_7$. The information exchanged between sites during the evaluation process consists of either simply a message, with negligible cost, or a message plus the join attribute values. The cost of transferring the join attribute values is the significant cost of interest.

The data transfers required in the first phase in the reduction of R_4 consist of the following.

1. Messages from sites of R_7 and R_8 to the site of R_4. { No join attribute values are transferred }
2. At the site of R_4, the semi-join $R_4' = (b_4 \bowtie R_4 \bowtie c_4)$, is computed and the projection $b_4' = \pi_B(R_4')$ is deduced. If the cardinality $|b_4'|$ is less than the cardinality $|b_4|$, either b_4' or the difference $(b_4 - b_4')$, whichever is less, is transferred to the site of R_1. Note that if $(b_4 - b_4')$ is shipped, the site of R_1 can derive the elements in b_4'. If $|b_4'| = |b_4|$, only a message needs to be sent to the site of R_1. No join attributes are required.

Assuming that at the site of R_1, a semi-join is computed and the relation R_1 is eventually reduced to R_1'. Let $b_1' = \pi_B(R_1')$. The transfers of either b_1' or $(b_1 - b_1')$ to the site of R_4 occurs only if $|b_1'| < |b_1|$. Otherwise only a message is sent.

The second phase in the evaluation of $EXP_4(Q)$ proceeds as follows.

3. Let the new join attribute values for B received at the site of R_4 be b_4'. If b_4' is received then R_4' may be further reduced to R_4'' by computing $R_4' = (b_4' \bowtie R_4')$.
4. Since R_4 has two descendents, namely R_7 and R_8, two new join attribute values, $b_4'' = \pi_B(R_4'')$ and $c_4'' = \pi_C(R_4'')$ are computed. If $|b_4''| < |b_4|$ the values of either b_4'' or the set difference $(b_7 - b_4'')$ are shipped to the site of R_7. A similar procedure is carried out with respect to the site of R_8 by comparing the sizes of $|c_4''|$ and $|c_4|$.

Using the conventional semi-join evaluation procedure, the full reduction of R_4 to R_4'' would have been achieved as follows.

1. A transfer of the join attributes $b_7 = \pi_B(R_7)$ from the site of R_7 to the site of R_4.
2. A transfers of $c_8 = \pi_C(R_8)$ from the site of R_8 to the site of R_4.
3. The derivation of $R_4' = (b_7 \bowtie R_4 \bowtie c_8)$, followed by the transfer of $b_4 = \pi_B(R_4')$ to the site of R_1.

Let R_1' be the eventual reduced relation of R_1 using the join attributes from R_2, R_3 and R_4. The second phase of the conventional semi-join computation involves the following.

4. The transfer of $b_1' = \pi_B(R_1')$ from the site of R_1 to that of R_4.
5. The derivation of $R_4'' = (b_1' \bowtie R_4')$ and then the transfer of $b_4'' = \pi_B(R_4'')$ and $c_4'' = \pi_C(R_4'')$ to the sites of R_7 and R_8 respectively.

Note that the filter technique may still be used for the semi-join evaluation even under join attributes are augmented with either the bitmaps or the disjoint subsets of the join attribute values. Clearly the technique of the augmented semi-join in reducing the relations at the respective sites is superior to the conventional semi-join evaluation strategy. Let each of the relations R_i, in a collection of distributed relations $R_1, R_2 \ldots R_n$, be augmented with disjoint subsets of the join domain values. Then given a tree query Q with semi-join expression $EXP(Q)$, the semi-join expression may be evaluated using the augmented semi-join evaluation algorithm below.

Algorithm Eval

Input :

A query Q, a query graph G_Q, and the relations $R_1, R_2, \ldots, R_k$, referenced in Q.

Output :

Fully reduced relations $R_1', R_2', \ldots, R_k'$, necessary and sufficient to answer Q at the result site.

Comment :

The relation R_i is assumed resident at the site i. In the query graph, let R_j be the ancestor of R_i and let $R_{i_1}, R_{i_2}, \ldots R_{i_m}$ be the descendents of R_i. The edge (i,j) is assumed to be labeled by the attribute A_{i_j}, while the edge (i_r, i) is labeled by A_{i_r}, for $r = 1, 2, \ldots, m$. The quantities $a_{i_r}' = \pi_{A_{i_r}}(R_{i_r}')$, and $\tilde{a}_{i_r} = (a_{i_r} - a_{i_r}')$ denote respectively, the join attribute values and the complement of the join attribute values received at site i from site i_r. Note that the elements in $a_{i_r} = \pi_{A_{i_r}}(R_{i_r}) \cap \pi_{A_{i_r}}(R_i)$ are available at the two sites i_r and i.

Method :

Step 1 [Initialization] : From the request site transmit G_Q to all sites i, having relations $R_i \in Q$.

Step 2 [Start of phase 1] : For each site i do the following. If R_i is a leaf node in the query graph then send a message to the parent node to proceed. If R_i is not a leaf node then wait for a'_{i_r} from all descendents R_{i_r}. If a_{i_r} is not received but $\tilde{a}_{i_r}$ is received then set $a'_{i_r} \leftarrow (a_{i_r} - \tilde{a}_{i_r})$ otherwise set $a'_{i_r} \leftarrow a_{i_r}$.

Step 3 [Data transfer to parents] : Compute $R'_i \leftarrow R_i \bowtie a'_{i_1} \bowtie \ldots \bowtie a'_{i_m}$ and set $a'_{i_j} \leftarrow \pi_{A_{i_j}}(R'_i)$. If i is a root node then proceed to Step 4 and initiate the execution of phase 2. If $|a'_{i_j}| = |a_{i_j}|$ then send a message to parent at site j. Otherwise send either a'_{i_j} or $\tilde{a}_{i_j}$, whichever is less, to the parent site j.

Step 4 [Start of phase 2] : If no join attribute values a'_{i_j} is received from the parent node at site j, then do nothing. If $\tilde{a}_{i_j}$ is received compute $a'_{i_j} \leftarrow a'_{i_j} - \tilde{a}_{i_j}$. Compute $R'_i \leftarrow R'_i \bowtie a'_{i_j}$. If R'_i has no children its process terminates.

Step 5 [Data transfer to descendents] : For each child node R_{i_r} at the site i_r do the following. Compute $a''_{i_r} \leftarrow \pi_{A_{i_r}}(R'_i)$. If $|a''_{i_r}| = |a'_{i_r}|$ send only a message to i_r. Otherwise set $\tilde{a}_{i_r} \leftarrow a'_{i_r} - a''_{i_r}$, and $a'_{i_r} \leftarrow a''_{i_r}$. If $|a'_{i_r}| < |\tilde{a}_{i_r}|$ send a'_{i_r} else send $\tilde{a}_{i_r}$ from i to the descendent at i_r.

In the algorithm we have assumed that the join attribute values are transferred. The use of join filters may be used if one is prepared to tolerate some errors. The reduction of each relation at the respective sites proceeds in parallel. Except for delays incurred when sites wait for messages and possibly join attribute values that they have to receive, the computation at each site proceeds asynchronously.

4.3 An Example.

Consider the following set of distributed relations : $R_1(A, B)$, $R_2(C, D)$, and $R_3(A, C, E)$, shown in Figure 4.2.

The relevant sets are $\rho(A) = \{R_1, R_3\}$ and $\rho(C) = \{R_2, R_3\}$. Suppose we desire the result of a query Q at the site of R_1 where this is specified as

$$Q = (R_1.A = R_3.A) \wedge (R_2.C = R_3.C).$$

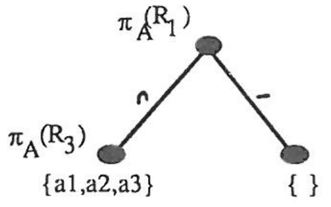
The corresponding query graph is shown in Figure 4.3.

Figure 4.2 : The relations R_1, R_2 and R_3, the partitioned subsets of the join domain values and the reduced relations R'_1 , $R'_2$and R'_3 generated.

R_1

A	B
a1	b1
a2	b2
a3	b3

At site 1



SP-tree of attribute A at site 1

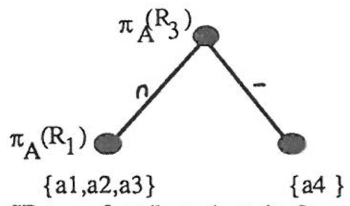
R'_1

A	B
a1	b1
a3	b3

R_3

A	C	E
a1	c1	e1
a1	c4	e2
a4	c3	e3
a4	c2	e3
a1	c5	e4
a2	c5	e5
a3	c3	e6

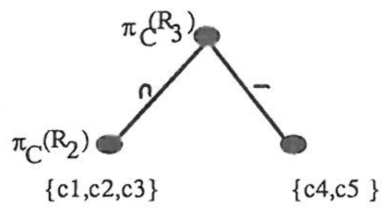
At site 3



SP-tree of attribute A at site 2

R'_3

A	C	E
a1	c1	e1
a3	c3	e6

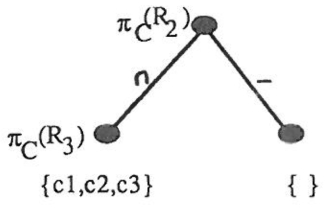


SP-tree of attribute C at site 3

R_2

C	D
c1	d1
c2	d2
c3	d3

At site 2



SP-tree of attribute C at site 2

R'_2

C	D
c1	d1
c3	d3

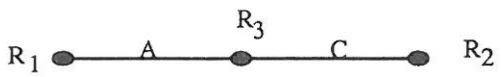


Figure 4.3 The query graph G_Q of example 4.3.

Let $EXP(Q)$ be the semi-join expression to be computed. The augmented semi-join evaluation procedure as follows.

Phase 1.

The only non-leaf node is R_3, hence site 3 is recognized as the root.

i. The relation R'_3 is computed at site 3 where

$$R'_3 = (\pi_A(R_1) \cap \pi_A(R_3)) \bowtie R_3 \bowtie (\pi_C(R_2) \cap \pi_C(R_3)).$$

The result is the reduced relation R'_3, shown in Figure 4.2.

ii. The projection of R'_3 on attribute A gives $a'_{31} = \{a1, a3\}$. Since a'_{31} is less than the original set of of the elements $a_{31} = \{a1, a2, a3\}$, the values given by $\tilde{a}_{31} = a_{31} - a'_{31} = \{a2\}$, is shipped to site 1. Similarly the projection of R'_3 on attribute B is $b'_{32} = \{c1, c3\}$. Hence { c2 } is sent to site 2.

iii. At site 1, the result of $R'_1 = a'_{31} \bowtie R_1$, is computed. This is shown in Figure 4.4. Since R_1 is at the root of the query graph, the first phase of the evaluation is completed.

Phase 2.

iv. At site 1,a'_{13} is derived from $\{a1, a2, a3\} - \{a2\} = \{a1, a3\}$. The reduced relation $R'_1 = R_1 \bowtie a'_{13}$ is derived.

v. Similarly at the site 2, the original relation R_2 is now reduced to $R'_2 = a'_{23} \bowtie R_2$. The reduced relation R'_2 is indicated in Figure 4.2.

The evaluation of the corresponding semi-join expression results in the relations R'_1, R'_2 and R'_3. The relations R'_2 and R'_3 excluding any extraneous attributes may be transferred to the result site for the desired response.

5 Reducing Effect of Augmented Semi-join.

The augmented semi-join evaluation method limits the amount of join attribute values transferred during distributed query processing to at most half the data movement needed in conventional semi-join evaluation strategies. Under some circumstances, as in the case of simple queries, the augmented semi-join evaluation method requires no data transfers of join attribute values, to achieve full reduction of relations.

The reducing effect of the augmented semi-join procedure is formalized by the two theorems below.

Theorem 5.1. *Let σ_Q, be a conventional semi-join evaluation strategy, for a query Q and let $|\sigma_Q|$ be the volume of join attributes needed to be transferred between the sites to achieve complete reduction of each relation $R_i \in Q$. Then an equivalent augmented semi-join evaluation strategy $\tilde{\sigma}_Q$ achieves the same reduction of each of the $R_i \in Q$ by transferring at most $|\tilde{\sigma}_Q| \leq 0.5|\sigma_Q|$.*

Proof

Let σ_Q be a conventional semi-join evaluation strategy for reducing each relation $R_i \in Q$. Consider any two sites j, and k with relations R_j and R_k respectively such that $a_{jk} = a_{kj} = \pi_A(R_j) \cap \pi_A(R_k)$. The amount of data transferred between the sites of R_j and R_k in the conventional semi-join strategy is a_{jk}. Suppose in an equivalent augmented semi-join strategy we must transfer a'_{jk}. Then the amount of data actually transferred $\tilde{\sigma}_Q$ equals $\min(a'_{jk}, a_{jk} - a'_{jk})$. This quantity is maximum when $a'_{jk} = a_{jk} - a'_{jk}$ or $a'_{jk} = 0.5a_{jk}$. Since this is always true between any pairs of communicating sites involved in the semi-join expression, it follows that $|\tilde{\sigma}_Q| \leq 1/2|\sigma_Q|$. ∎

The significance of Theorem 5.1 is that in any distributed query optimization algorithm that uses augmented semi-joins, the volume of data transfers required is at most 50% of that required in using conventional semi-join evaluation method.

Theorem 5.2. *Let R_i, $i = 1, 2, \ldots n$ be a set of distributed relations, each having an attribute A. Let R_k be a designated relation Then any simple query of the form*

$$Q = \bigcap_{i-1}^{n} (R_i.A = R_k.A),$$

can be processed with no join attribute transfers.

Proof.

The evaluation of a semi-join expression is essentially a reduction of each of the relations referenced in a query to the set of tuples t such that

$$t.A \in (\bigcap_{i=1}^{n} R_i.A) \bigcap R_k.A).$$

But by the join attribute partitioning procedure the set of elements in $a_{ij} = (\cap_{i=1}^{n} R.A) \cap R_k.A$ is available at all sites of any $R_i \in Q$. Hence each site can compute $R'_i = R_i \bowtie a_{ij}$ without the need to transfer the join attribute values. ∎

6. Conclusion

In this paper we have shown that, by storing appropriate information with little extra storage, at the sites of a distributed databases, it is possible to reduce the volume of data transmitted when answering distributed queries using semi-join evaluations. We have presented two methods of augmenting stored relations with disjoint subsets of the join attribute values so that subsequent semi-join evaluation required for distributed query processing, is performed with half the volume of data needed in a conventional semi-join evaluation method. Consequently, any optimization algorithm that incorporates conventional semi-join techniques to decrease the amount of data transfers is further improved by the augmented semi-join technique presented in this paper.

The technique of using filters to determine tuples of relations having common values of join domains may be applied to the augmented semi-join methods as well. These results apply to acyclic queries and have been described when these queries are expressed in terms of equi-join clauses. The proposed method is, however, equally effective when distributed queries involve the μ-joins (e.g., union-join, intersection-join, symmetric difference join etc.) defined by Merrett [19]. These extensions as well as further discussions on cyclic queries are presented in a follow-up paper.

Acknowledgement

This research is supported in part from grants from the Natural Sciences and Research Council of Canada.

References

[1] Apers, P. M. G., Hevner, A. R. and Yao, S. B. Optimization algorithms for distributed queries. *IEEE Trans. on Software Engineering, 9, 1 (Jan. 1983), 57–68.*

[2] Bernstein, P. A. and Chiu, D. M. Using semi-joins to solve relational queries. *J. ACM, 28, 1 (Jan. 1981), 25–40*

[3] Bernstein, P. A., Goodman, N., Wong, E., Reeves, C. L. and Rothnie, J. B., Jr. Query processing in a system for distributed databases (SDD-1). *ACM Trans. on Database Systems, 6, 4 (Dec. 1981), 602–625.*

[4] Bernstein, P. A. and Goodman, N. Power of the natural semi-join. *SIAM J. of Comput. 4, 10 (Nov. 1981), 751–777.*

[5] Bloom, B. H. Space/Time trade-off in hash coding with allowable errors. *Comm. ACM, 13, 7 (Jul. 1970), 422–426*

[6] Ceri, S. and Pelagatti, G. Distributed query processing: principles and systems. *McGraw Hill Publ., New York (1984).*