ON THE DESIGN OF A QUERY PROCESSING STRATEGY IN A DISTRIBUTED DATABASE ENVIRONMENT

BY
C.T. Yu, C.C. Chang
Dept. of Electrical Engineering & Computer Science
UNIVERSITY OF ILLINOIS AT CHICAGO
CHICAGO, ILLINOIS 60680

Abstract:

An algorithm is given to process a given query in a fragmented distributed data base environment. Unlike previous algorithms, it has the following desired features.

(1) It makes use of redundant relations to reduce communication cost;
(2) a copy of each relation referenced by the query is selected so that the set of relations are contained in the minimum number of sites;
(3) an efficient algorithm to process fragments is provided;
(4) all relations that need not be sent to the assembly site to produce the answer are identified. Thus, unnecessary sending of these relations across sites and processing on these relations, which are common in earlier algorithms, are avoided;
(5) useless semi-joins are discarded and "worse" semi-joins are replaced by better ones;
(6) a process to estimate the cost and the benefit of a semi-join, based on dynamic execution of semi-joins is introduced. It is expected that the new process is more accurate than earlier estimation process.

The algorithm is easy to implement and is operational.

1 Requirement

We are implementing a front end distributed query processing strategy that will exist on top of existing data management systems. The underlying data management systems may be heterogeneous and may be of any structural data model, although in our initial implementation all of the databases are relational. The user views the data as a single unified database through a uniform global schema which is relational [Codd, Date].

The physical relations may have redundant copies at different sites, or a logical relation may be split horizontally, based on a predicate applied to some field, with each fragment of the relation stored at a different site.

In this paper we assume that the cost to send an amount of data from one site to another is independent of the sending or receiving site. This is consistent with early papers on distributed query processing [HeYa, SDD1, YLCC, Chan1, etc] and is consistent with the behavior of the network on which this query processing strategy is being implemented.

It has been shown that optimal processing for even some restricted types of queries is NP-hard [Hevn, GoSh, YLCC]. We now design a heuristic strategy which processes queries efficiently in the above environment and which is simple enough to be implemented easily.

None of the existing query processing stategies fits our requirements in the following senses.

Early strategies [SDD1, ChHo, HeYa, Chan1, YuLO] do not use the redundant copies of relations and assume that a copy of each relation referenced by the query has been pre-selected. It is obvious that the selection of a copy for each relation affects the performance of processing the query. In fact, optimal selection can be shown to be NP-hard [YLCC]. In this paper, we give a simple algorithm to solve the copy selection problem and to identify the primary copy and a number of secondary copies of each relation referenced by the query.

Most query processing algorithms [BeCh, HeYa, Chan1, SDD1V2] did not discuss the processing of fragments. While version one of the SDD-1 algorithm [SDD1V1] mentioned how fragments can be handled, the discussion was left out in the second version of the algorithm [SDD1V2]. Furthermore, it was pointed out by [Chan2] that such processing of fragments can be very inefficient. In this paper, we give a simple strategy to process fragments efficiently.

Other processing algorithms [ChHo, YLCC], while obtaining optimal sequences of semi-joins [BeCh] for certain type of queries, are not general enough to process queries in our environment and they are more complicated to implement. The R* system [WDHL] enumerates many strategies and chooses the one with least cost. This may be appropriate for embedded queries which are likely to be executed repeatedly. In our environment, the users are expected to submit ad-hoc queries. Furthermore, it was pointed out in [BlLu] that for a query referencing only a few relations, there are over a billion possible strategies to answer the query. Thus, our aim is to design a simple and yet reasonably efficient algorithm that yields good performance.

Most existing query processing algorithms

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1983 ACM -0-89791-104-0/83/005/0030 $00.75

[SDD1, Chan1, HeYa, YLCC] consist of two main processing phases. The first phase consists of using semi-joins [BeCh] to reduce the relations referenced by the query, while the second phase joins the relations together to produce the answer. It turns out that some of the relations involved in the semi-joins are unnecessary in producing the answer in the second phase. As a result, these relations need not be sent to a common site and communication cost is saved. Furthermore, even some of the semi-joins involved in the first phase are either redundant or replaceable by better semi-joins. In this paper, we give a methodology to identify relations not needed to produce the answer, to delete redundant semi-joins and to replace semi-joins by better semi-joins. While a partial solution to the above tasks have been given early by [SDD1, Luk2, Chan1, YLCC], the treatment given here is much more general and substantial.

The estimation of the size of an intermediate relation produced as a result of the execution of a semi-join has a significant impact on the performance of a distributed query processing strategy. Earlier estimation algorithms [HeYa, SDD1, Chan1, Chan2, YLCC, YuLO] are based on certain uniform distribution and independent distribution assumptions. The use of such assumptions may produce some errors. While the error produced on the first intermediate relation may be small and be tolerated, the error will tend to become larger as more intermediate relations are produced. In order to control the magnitude of the error, we propose that semi-joins be executed dynamically. That is, after executing a semi-join, the size of the reduced relation is obtained exactly and this information is passed back to our algorithm, which then estimates the size of the next reduced relation. Since the sizes of all current relations are known exactly, we expect the error in the estimation of the next reduced relation to be small. An estimation procedure in the environment of executing the semi-joins dynamically, is provided in this paper.

When a semi-join on attribute A from relation Ri to relation Rj is executed, data from the site containing Ri(A) is sent to the site containing Rj. We note that if Ri(A) contains more than half of all the possible distinct values, it is cheaper to send its complement. This idea is incorperated into our algorithm.

In subsequent sections, we will outline the solutions to the problems given above: i) the copy identification problem, ii) the elimination of unnecessary relations in the second phase, iii) the elimination of redundant semi-joins and the replacement of semi-joins by better semi-joins in the first phase, iv) the processing of fragments, v) the estimation of relation size using dynamic selection of semi-joins, and vi) the use of sending relations in complements. The solutions to the problems are integrated to form the core of our distributed query processing algorithm.

2 Notations and Definitions

A query has two components: the target component and the qualification component. The qualification component selects the tuples of the referenced relations that satisfy the qualification, while the target component specifies the attributes of the selected tuples to be outputted to the user. Specifically, queries discussed here are of the form similar to those in [SDD1, HeYa etc].

{(Ri.Ai1, ...,Rj.Ajl) |
(Rk.Akl=Rt.Atl)AND ...AND(Ru.Aul=Rv.Avl)}

where the qualification component is a conjunction of equality clauses of the form (Rk.Akl=Rt.Atl) AND....AND (Ru.Aul=Rv.Avl) and the target component is (Ri.Ai1, ..., Rj.Ajl). If a clause involves a relation with a constant or it involves 2 attributes of the same relation, it is eliminated by local processing [HeYa, Wong]. For the sake of simplicity, let us rename all attribute names so that two attributes have the same name iff they are related by either an equality clause or a number of clauses through transitivity of equality. For examples, (Ri.A=Rj.B) AND (Rj.B=Rk.C) will be renamed (Ri.A=Rj.A) AND (Rj.A=Rk.A); (Ri.A=Rj.A) AND (Rk.A=Re.A) will be renamed (Ri.A=Rj.A) AND (Rk.B=Re.B), because the A's in the two clauses are not related.

If a relation R has attribute A, then R(A) is the projection of R on attibute A. |R| is the size of relation R.

A semi-join from Ri to Rj on attribute A, denoted by Ri-A->Rj, is the join of Ri and Rj on attribute A, then projected on the attributes of Rj. It can be executed by sending the distinct values of Ri on attribute A to the site(s) containing Rj and then eliminating those tuples of Rj which do not contain a value in Ri(A). Here, A can be either a single attribute or a set of attributes.

A semi-join Ri-A->Rj is implied by the query if either (Ri.A=Rj.A) appears in the qualification component of the query or the clause can be deduced from the qualification. For example, the semi-join Ri-A->Rj is implied by (Ri.A=Rk.A) AND (Rk.A=Rj.A).

A database state is the assignment of contents to the relations.

3 The copy identification Problem

In our environment, a relation may have a number of horizontal disjoint fragments or may have a number of copies but not both. If it is fragmented, then since the fragments are disjoint, all fragments of the relation have to be chosen to process the query. If a relation has a number of copies, the problem is how to select one copy of each relation referenced by the query so that the cost to process the query is minimized. Following early papers [HeYa, SDD1, YLCC, Chan1, etc], the cost to send X amount of data from one site to another is C0 + C1 X, where C0 and C1 are constants, and the cost to process a query is the summation of costs for sending data among sites. It can be shown that the copy identification problem is NP-hard for processing simple queries, where a simple query [HeYa] is one referencing relations having a single joining attribute only.

Proposition 3.1: The copy selection problem, even for simple queries is NP-hard [YLCC].

When data transmission cost is dominant as compared to local processing cost, it is likely that the minimization of sites containing the referenced relations will reduce the traffic cost.

Example 3.1: Let a query be {R2.C|R1.C=R2.C}. Assume site S1 contains relations R1 and R2 and site S2 contains relation R2. Suppose one copy of each relation is to be selected. One can select (i) the R1 in S1 and the R2 in S2, or (ii) both the R1 and the R2 in S1 to perform semi-join R1-C->R2 (or R2-C->R1) to obtain the answer. Obviously, the first selection incurs data transmission from S1 to S2 (or S2 to S1) while the second one does not require any data transmission between different sites. []

Unfortunately, to find a minimum number of sites covering a set of all relations referenced by the query is also NP-hard, as the Minimum-subset-problem [AHU] is reducible to it.

We propose to find a primary copy and a number of secondary copies of each unfragmented relation which is referenced by the given query. The primary copy of a unfragmented relation, say Ri, has the characteristics that (1) if Ri is to be reduced (i.e., semi-join of the form Rk-->Ri is to be executed), the primary copy of Ri should be used. No secondary copy of the relation is used. (2) After Ri has been reduced, no secondary copy of Ri should be used to reduce other relations. A secondary copy of Ri may be used to reduce another fragmented or unfragmented relation, if Ri has not been reduced. (Condition (1) and (2) can be relaxed to allow some secondary copies of Ri to be reduced with same minor changes in section 6.)

Example 3.2: Let a query be {R1.D|R1.C=R2.C}. R1 has two fragments F11 and F12 with fragment F11 at site S1 and the other fragment F12 at site S2. R2 has two copies, one at site S1 and the other copy at site S2. Existing algorithms (e.g. [Chan2], [HeYa], [SDD1], etc.) do not address the issue of using redundant copies but select one copy of R2. A possible semi-join to be executed is of the form R1-C->R2 or R2-C->R1. As a result, F11(C) or F12(C) or R2(C) has to be transmitted across sites. Suppose both copies of R2 are chosen with one copy designated as the primary copy and the other as a secondary copy. Then the semi-join R2-C->R1 can be executed with no data transfer across sites. This saves communication cost. Furthermore, the reduction of the two fragments of R1 by the two copies of R2 can take place in parallel. []

We now give a simple algorithm to find a primary copy of each relation referenced by a query such that the set of primary copies of the relations are contained in the minimum number of sites. Additional copies of the relations in the chosen sites are designated secondary copies. The algorithm runs in exponential time, although the number of sites and the number of copies of the relations referenced by a query is usually not large enough to generate a large running time.

Clearly, all fragments referenced by the query have to be chosen. The relations in the sites containing the fragments are chosen and are eliminated before Algorithms Pre-Processing and Opt-Site which will be given later are executed. If the answer site contains a fragment and a number of relations, then the copies of the relations are designated as primary copies.

Let F be a subset of sites each containing a fragment of a relation referenced by the query. Assume {R1, ..,Rm} is the set of relations left after the elimination of all fragments and relations in F. Assume further {S1, ..,Sn} is the set of sites each containing at least a relation in {R1, ..,Rm}. The following algorithm finds the minimum number of sites from {S1, ..,Sn} to cover the set of relations {R1, ..,Rm}.

Algorithm Opt-Site

Step 1 For each relation Ri, let Si1, ..., Sij_i be the sites, each containing a copy of Ri. The fact that any one of these sites can be used is denoted by (Si1 OR Si2 OR .... OR Sij_i).

Step 2 Since all relations referenced by the query have to be covered, we compute $\text{AND}_{i=1}^{m}$ (Si1 OR Si2 OR .... OR Sij_i).

Step 3 The above expression is re-written in disjunctive normal form with each component being the conjunction of a number of S-literals.

Step 4 Choose a component having the minimum number of S-literals. If the answer site is contained in a component having the minimum number of literals, then choose the component.

It is easy to see that a S-literal corresponds to a site and a component corresponds to a set of sites containing a copy of each of the referenced relations. Thus, the minimum number of sites covering all the referenced relations can be obtained by choosing a component having the minimum number of S-literals.

Since the above algorithm runs in exponential time, we now reduce the number of sites and the number of relations by the following method.

Definitions: Relation Ri dominates relation Rj if (the sites containing Ri) $\subseteq$ (the sites containing Rj).

Site Sk dominates site Sl if (the relations in site Sl) $\subseteq$ (the relations in site Sk). If (the relations in Sk) = (the relations in Sl), then both Sk dominates Sl and Sl dominates Sk. If Sl is the answer site, then we choose to let Sl dominate Sk but not conversely (i.e., we eliminate Sk).

Suppose Ri dominates Rj. By definition, each site containing a copy of Ri contains a copy of Rj. Since a copy of Ri must be chosen, the site containing that copy of Ri must contain a copy of Rj and therefore the covering of Rj need not be

considered. This allows us to eliminate a dominated relation. The primary copy of the dominated relation is at the same site of that of the relation that dominates, which may not have been determined.

Suppose site Sk dominates site S1. If a solution consists of site S1 and some other sites, we can replace site S1 in the solution by site Sk without losing the covering of any relation and having no more sites. Thus, a dominated site can also be eliminated. A dominated site does not contain a primary nor a secondary copy of a relation for the processing of the query.

Finally, we note that if a relation has a single copy only, then the site that contains the copy must be chosen. Such a relation is called an essential relation. The primary copy of an essential relation is unique.

The following algorithm eliminates the dominated sites and the dominated relations and selects the essential relation. It should be executed before the algorithm Opt-Site.

Algorithm Pre-Processing

```
while (there is a relation or a site dominance)
      {
       eliminate the dominated site
              or the dominated relation;
      }
while (there is an essential relation)
      {
       select the site containing an essential
       relation;
      }
```

The following proposition states that the optimal solution is obtained by using Pre-Processing before Opt-site.

Proposition 3.2: Algorithm Pre-Processing together with Algorithm Opt-Site gives the smallest number of sites.

Sketch of proof: It is easy to see that Opt-Site obtains the smallest number of sites from {S1, ..,Sn} containing the set of relations {R1, ..,Rm}. Consider the three operations defined in Pre-Processing: site domination, relation domination and selecting essential relation. It can be proved by induction on the number of applications of the operations that the optimal solution is preserved after an application. []

If local processing cost is significant, it will be beneficial to perform local processing in parallel. In this case, the maximum number of sites may be desirable. Algorithm Pre-Processing and Opt-Site can be modified slightly to give the maximum number of sites.

After the execution of Pre-Processing, the primary copies of essential relations are identified, the location of the primary copy of each dominated relation is restricted to that of the primary copy of the relation that dominates. After the execution of Opt-Site, the component having the minimum number of S-literals determines the primary copies of the remaining relations. All other copies of the relations in the chosen sites are secondary copies.

4 Elimination of Unnecessary Relations

A query Q, which references a set of relations R1, R2, ..., Rn, is given. After a sequence of semi-joins act on the relations to produce R1´, R2´, ..., Rn´, the problem is to identify those relations that are not needed to answer Q.

It has been pointed out in [YLCC, Chan1] that certain relations that are involved in the execution of some semi-joins in the first phase need not be used in the second phase to produce the answer. However, their results are only applicable to certain sequences of semi-joins. The result presented here applies to any sequence of semi-joins. It is known [YuLO, YLCC] that optimal processing of a certain type of queries would involve the execution of semi-joins on one attribute and then a switch to another attribute before completion of semi-joins on the first attribute. The strategy in [Chan1] does not allow such switching. Thus, if an optimal or close-to optimal sequence of semi-joins are to be desired, the elimination of useless relations by the results in [Chan1] is not applicable. Furthermore, [Chan1] does not allow a relation having a target attribute to be eliminated, while in this proposed algorithm, such a relation can be eliminated. The observation that some such relations can be eliminated was made when examining our test queries. The theory developed here is applicable to both multiple attribute semi-joins and single attribute semi-joins while [Chan1] uses single attribute semi-joins only.

After our algorithm executes one semi-join, the number of tuples of the reduced relation is returned so that the estimation is accurate. In order that this component of our algorithm integrates smoothly with the estimation component, the following solution is proposed.

The query Q consists of two components, the qualification QUAL and the target TL. After a semi-join is executed, both the qualification and the target may change. Let the new query be Q´ = {TL´|QUAL´}. Clearly, the answer produced by Q on the relations {R1,R2, ...,Rn} must be the same as that produced by Q´ on the relations {R1´,R2´, ...,Rn´} where one of the R´s is modified due to the semi-join and the other (n-1) relations are unchanged. If both QUAL´ and TL´ do not reference Ri´, then Ri´ can be eliminated from Q´. Our method consists of finding QUAL´ and TL´ and eliminating any relation which occurred in Q before the semi-join but disappears from Q´ after the semi-join is executed.

Definitions: The joining attributes of a relation R, denoted by J(R), are the set of attributes of R appearing in QUAL; the target attributes of R, T(R), are the set of attributes of R appearing in TL.

Let S: Ri-A->Rj be the semi-join executed (both Ri and Rj have not been eliminated).

Algorithm Elim(S)

```
If COND = (A=J(Ri)) AND (T(Ri) C A) is true
then     /* Ri can be eliminated from the query */
     {   modifyQL(S);
```

if T(Ri)$\neq\emptyset$ then modifyTL(S); }

Procedure modifyQL(S) /* modifying QUAL to QUAL´ */
for each attribute X in A, eliminate the clause (Ri.X=Rj.X) if exists in QUAL and replace each clause of the form (Ri.X =Rk.X), if exists, by (Rj.X=Rk.X), k$\neq$i,j, if the latter does not already exist;

Procedure modifyTL(S) /* modifying TL to TL´ */
for each attribute X in A, replace Ri.X in TL by Rj.X;

Example 4.1: Q = {R2.D|(R1.C=R2.C)AND(R2.D=R3.D)}.
QUAL = (R1.C=R2.C) AND (R2.D=R3.D)
TL = R2.D

Let the sequence of semi-joins executed be
S1: R1-C->R2
S2: R2-D->R3
S3: R3-D->R2
S4: R2-C->R1.

Initially, J(R1)={C}, J(R2)={C,D} and J(R3)={D}.

After executing S1, since J(R1)={C} and T(R1)=$\emptyset$, R1 is eliminated.
new QUAL = (R2.D=R3.D)
TL remains unchanged.

It should be noted that since attribute C does not appear in the new QUAL, J(R2) becomes {D} only.

After executing S2, since j(R2)={D} and T(R2) ={D} $\subseteq$ {D}, R2 is also eliminated.
Since T(R2)={D}$\neq\emptyset$, TL is modified.
new QUAL = $\emptyset$
new TL = R3.D.

At that point, since QUAL becomes null, semi-joins S3 and S4 will not be executed. []

Definition: Two queries Q and Q´, which reference relations {R1, ...,Rn} and {R1´, ...,Rn´} respectively are equivalent if their answers are identical for every database state for {R1, ...,Rn}, where {R1´, ...,Rn´} are obtained from {R1, ...,Rn} through a sequence of semi-joins implied by Q.

Proposition 4.1: After the semi-join S: Ri-A->Rj is executed, if condition COND is satisfied and the semi-join S is implied by Q, then Q which references {R1, ...,Rn} is equivalent to Q´ which references {R1, .., Ri-1,Ri+1, ..., Rj-1,Rj´,Rj+1, .., Rn} where Rj´ is obtained from Rj by S and Q´ is obtained from Q as given in Algorithm Elim.
Sketch of proof: Let Q be {TL|QUAL}. Since semi-join Ri-A->Rj is implied by Q, we can transform Q to an equivalent query Q1={TL|QUAL1} by the following procedure
(1) apply modifyQL(Ri-A->Rj) to QUAL to obtain QUAL1;
(2) for every attribute X in A, add a clause (Ri.X=Rj.X) to QUAL1.

When J(Ri)=A, the above transformation guarantees that Ri in QUAL1 appears only in the clauses of the form (Ri.X=Rj.X), where X in A. Thus, when the query-graph of QUAL1 is constructed, Ri is adjacent to Rj only. By the leaf-to-root reduction introduced in [BeCh], if we consider the subtree with Rj as the root and Ri as its leaf, Rj is fully reduced (see [BeCh]) by Ri with respect to the sub-qualification Ri.A=Rj.A after the execution of Ri-A->Rj (i.e. Ri can be eliminated from the entire qualification). When T(Ri) $\subseteq$ A, Q1 can be transformed further to another equivalent query Q2={TL1|QUAL1} by modifyTL(Ri-A->Rj). Thus, Ri can be eliminated from both the qualification and the target. []

Proposition 4.2: After the semi-join S is executed, if COND is not satisfied, for Q which references {R1, ...,Rn} and every Q´ which references {R1, .., Ri-1,Ri+1, ..., Rj´, .., Rn}, there exists a database state for {R1, ...,Rn} such that Q´ and Q are not equivalent, where Rj´ is obtained from Rj by S.
Sketch of proof: If COND is not satisfied, then at least one of the following cases arises.

case 1: T(Ri) is not a subset of A, i.e. there exists an attribute B which is in T(Ri) but not in A.

If Ri is eliminated, obviously applying any Q´ on {R1, .., Ri-1,Ri+1, ..,Rj´, ..,Rn} is unable to answer anything under Ri.B. Thus, the answer obtained will not be identical to that obtained by applying Q on {R1, ..,Rn}.

case 2: A$\neq$J(Ri), i.e., J(Ri) contains an attribute C which is not in A.

Since attribute C has not been eliminated, there is at least one other relation, say Rk, having C. A database for {R1, ..,Rn} can be assigned such that the values of Ri.C are disjoint from those of Rk.C and the set of tuples satisfying the qualification on {R1, .., Ri-1,Ri+1, ..,Rj´, ..,Rn} is non-empty. Thus, Q´ which references the n-1 relations will yield a non-empty answer, is not equivalent to Q which yields null, since (Ri.C=Rk.C) cannot be satisfied. []

Proposition 4.1 states that after executing semi-join S, if the condition COND is satisfied, then the answer on the original query Q, which references Ri is the same as that of the new query which does not reference Ri. Thus, Ri can be eliminated. Proposition 4.2 states that if COND is not satisfied, then there is no equivalent query which does not reference Ri.

In example 4.1, the SDD-1 algorithm [SDD1V1, SDD1V2] may execute the sequence of 4 semi-joins and then send the reduced relation R1, R2 and R3 to an assembly site to produce the answer, while our algorithm, after executing the first two semi-joins, would have the answer as the reduced R3. Clearly, if the answer site is not the same as the site containing R3, then only the reduced R3 needs to be sent, otherwise no relation needs to be shipped. Thus, a substantial saving is achieved. Chang´s algorithm [Chan1] would send R2 and R3 to the assembly site, if semi-joins S1 and S2 are

performed. If S3 is also performed, then R3 needs not be sent.

In example 4.1, there is only one relation in the target component and the query is a tree query [BeCh, YuOz]. As a result, algorithms in [ChHo, YuLO] using dynamic programming can be very effective in obtaining the answer. When the number of relations having target attributes is more than 1 or the query is a cyclic query [BeCh, YuOz], the above algorithms [ChHo, YuLO] are not applicable.

The algorithm to eliminate a relation is a minor variation of the algorithm to determine whether a given query is a tree query [YuOz, Grah]. That algorithm has been used in constructing acyclic database schemes in logical database design [BFMM, FaMU].

5 Replacement of Semi-joins by Better Ones and Identification of Useless Semi-joins

Example 4.1 illustrates that certain semi-joins can be discarded. We now identify those semi-joins that can be discarded and those semi-joins which can be replaced by better ones.

Given any sequence of semi-joins, our intention is (1) to find a better sequence of semi-joins, (2) to identify the useless semi-joins in the better sequence and discard them. It turns out that if a semi-join involves an eliminated relation (which is identified by the result of section 4), then we can construct a better semi-join, without knowing subsequent semi-joins to be executed.

To facilitate the discussion, a special type of graphs are defined. A Q-graph G=(V,E) is a connected graph containing both directed and undirected edges; V is the set of relations of the query and E represents the qualification; the graph contains one connected component of undirected edges and zero or more rooted trees of directed edges. The connected component represents the relations which have not been eliminated and the clauses in the qualification referring to those relations. Each rooted tree consists of the relation at the root which has not been eliminated (the root also appears in the connected component) and other nodes representing eliminated relations. Initially, the Q-graph is the same as the query-graph of the original query [BeGo, YuOz], where each edge (Ri,Rj) with label A denotes the joining condition Ri.A=Rj.A. After executing a semi-join, if the semi-join does not cause a relation to be eliminated, then the Q-graph remains unchanged. If the semi-join, say Ri-A->Rj, causes Ri to be eliminated, then Ri is marked eliminated and the undirected edge between Ri and Rj in the Q-graph, if exists, is replaced by a directed edge with label A from Ri to Rj and every edge between a relation R, which is not Rj and has not been eliminated, and Ri with label B (B is a subset of A) is replaced by a undirected edge with the same label between R and Rj. If two or more edges with different labels exist between a pair of relations, then they are represented by a single edge with a label which is the union of the labels in the edges. Figure 5.1 shows the changes of the Q-graphs of a query after a sequence of semi-joins is executed.

Definition: Root(R) is R, if R has not been eliminated and is the root of the tree containing R if R has been eliminated. For example, in Figure 5.1(b), Root(R1) is R2, and in Figure 5.1(e), Root(R1) is R3.

Q={R2.D|(R1.C=R2.C)AND(R4.C=R1.C)AND(R2.D=R3.D)}
semi-joins: R1-C->R2, R2-C->R4, R4-C->R2
and R2-D->R3

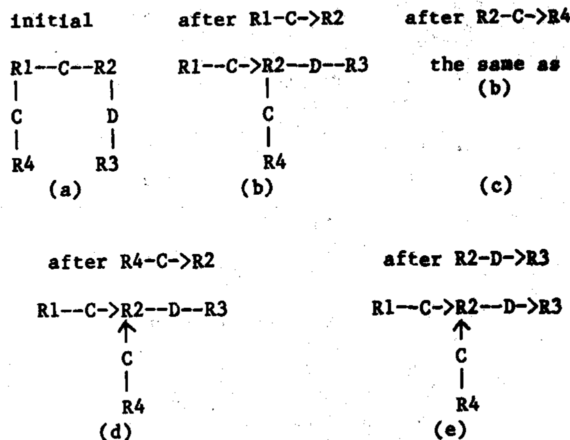

Figure 5.1 Transformation of Q-graphs

Since an eliminated relation will no longer be used in producing the answer, intuitively, it is not wise to execute any semi-join reducing an eliminated relation or using an eliminated relation to reduce another relation. What we will establish is to replace an eliminated relation by its root whenever it is involved in a sequence of semi-joins, so that a better sequence of semi-joins is obtained.

In SDD-1, the cost of a semi-join S: Ri-A->Rj, cost(S), is defined to be $|Ri(A)|$ if Ri and Rj are at different sites, otherwise the cost is zero. The benefit of the semi-join is defined to be $|old\ Rj| - |new\ Rj|$, where the new Rj is the Rj after being reduced by S.

Let M be the set of relations referenced by the query. Let SQ1 be a sequence of semi-joins executed on the relations in M and SQ2 be another such sequence. The cost of a sequence of semi-joins is defined to be the sum of the cost of each semi-join in the sequence.

Definition: SQ2 is a better sequence of semi-joins than SQ1 if (i) cost(SQ1) $\geq$ cost(SQ2) (ii) E1 $\subseteq$ E2, where Ei is the set of eliminated relations after the execution of SQi, $1 \leq i \leq 2$. and (iii) for each relation R in M-E2, R after execution of SQ2 $\subseteq$ R after execution of SQ1.

Condition (ii) and (iii) are to ensure that the cost to send the relations, which have not been eliminated by the semi-joins, to assembly site is smaller if SQ2 is used instead of SQ1.

The following algorithm maps a given sequence of semi-joins to a better one.

Algorithm Better(S) /* assume S is Ri-A->Rj */

replace S by S1: Root(Ri)-A->Root(Rj),
where if Root(Ri)=Root(Rj), then the semi-join
S1 is not executed.

The following result can be proved.

Proposition 5.1: Let SQ1 be a sequence of semi-joins, which does not eliminate any relation. If SQ2 is the corresponding sequence obtained by Algorithm Better, then SQ2 is better than SQ1.

Sketch of proof: Let n be the number of semi-joins in SQ1. Algorithm Better transforms SQ1 into a corresponding sequence of n semi-joins, SQ2. Let M be the set of all relations referenced by the query. Let E2i be the set of relations that have been eliminated after executing the first i semi-joins in the sequence SQ2; E1 and E2 be the set of relations that have been eliminated after executing SQ1 and SQ2 respectively. Since SQ1 doesn't eliminate any relation, it is clear that E1 is null. Thus, E1 $\subseteq$ E2n=E2. It can be proved by induction that

(i) for every relation u which have not been eliminated by SQ2 i.e. u is in M-E2i, then u in SQ2 $\subseteq$ u in SQ1, and

(ii) for every relation e in E2i, if Y, the common joining attributes in the original query between relation e and relation Root(e) is not null, then Root(e) projected on Y in SQ2 $\subseteq$ e projected on Y in SQ1.

Consider a semi-join S: Ri-A->Rj in SQ1 mapped to the semi-join S1: Root(Ri)-A->Root(Rj) in SQ2. If Ri has not been eliminated, then Ri=Root(Ri). By (i), cost(S1) $\le$ cost(S). If Ri has been eliminated and Rj is in the same subtree as Ri, then S1 is not executed and clearly cost(S1) < cost(S). If Ri has been eliminated but Rj is outside the subtree, then Root(Ri) can be proved to contain attribute A. Let Y be the joining attributes in common between Ri and Root(Ri). Y is not null, because A $\subseteq$ Y. By (ii), Root(Ri) projected on Y in SQ2 $\subseteq$ Ri projected on Y in SQ1. Since A $\subseteq$ Y, Root(Ri) projected on A in SQ2 $\subseteq$ Ri projected on A in SQ1. Thus, cost(S1) $\le$ cost(S) in all cases. Since the above statement is true for any semi-join in SQ1 and the corresponding semi-join in SQ2, cost(SQ2) $\le$ cost(SQ1). Condition (i) guarantees that the cost to transfer data to the assembly site after the execution of SQ2 is no larger than that to transfer data to the assembly site after the execution of SQ1. Thus, SQ2 is better than SQ1. []

Example 5.1 Q = {I1.D|(I1.C=I2.C)AND(I1.D=I2.D)
AND(I1.C=C1.C)AND(C1.C=C2.C)}

Let the sequence of semi-joins executed be SQ1:

S1: C1-C->I1
S2: I1-D->I2
S3: I2-C->C2
S4: C2-C->C1
S5: C1-C->I2

By Algorithm Better, the above sequence is transformed to SQ2:

S1: C1-C->I1 (C1 is eliminated; Root(C1)=I1)
S2: I1-D->I2
S3: I2-C->C2
SS4: C2-C->I1 (C2 is eliminated; Root(C2)=I1)
SS5: I1-C->I2

Since SQ1 and SQ2 differ only at the last 2 semi-joins, cost(S1,S2,S3 in SQ1)=cost(S1,S2,S3 in SQ2).

However, cost(S4)=cost(SS4) and
cost(S5)>cost(SS5) (since I1(C) $\subseteq$ C1(C)).
Thus, cost(SQ1) > cost(SQ2).

Moreover, SQ1 does not eliminate any relation but SQ2 eliminates {C1, C2}.

It is clear that after semi-joins S4 and SS4,
I1 in SQ2 $\subseteq$ I1 in SQ1,
and the I2 in both sequences are the same.

But, after S5 and SS5, the I1 in both sequences remain unchanged and I2 in SQ2 $\subseteq$ I2 in SQ1.

Thus, SQ2 is better than SQ1. []

SDD-1 generates a sequence of semi-joins, which will be executed on the relations referenced by the query. At termination, all relations are sent to an assembly site to produce the answer. If the same sequence of semi-joins are fed into our algorithm, then the useless semi-joins are eliminated, some semi-joins are replaced by better semi-joins and only a subset of the relations which are really needed to produce the answer are sent to the assembly site. As a result, fewer semi-joins are executed, more costly semi-joins are replaced by less costly semi-joins and there are fewer relations, each of which is no larger than the corresponding one reduced by the SDD-1 algorithm, to be sent to the assembly site. Thus, our algorithm is less costly. The cost of an algorithm is defined to be the sum of the costs of sending data (This is referred to as <u>total time</u> [HeYa]).

Proposition 5.2: cost(our algorithm) $\le$ cost(SDD-1) for any query and any database state. The inequality will be strict for some queries and some database states.

The concept of obtaining better sequences of semi-joins was given in [Luk2, SDD1V1, SDD1V2]. However, they refer to relations which are not eliminated and the entire sequence of semi-joins is generated before improvement to the sequence is made. In our approach, each "worse" semi-join involving eliminated relation is either discarded or replaced by better one and we execute semi-joins dynamically.

6 Processing of Fragments

There are three ways to perform semi-joins: send one fragment to reduce another fragment, send one relation to reduce another fragment or send one relation to reduce another relation. As pointed out by [Chan2], there is no benefit of

reducing a fragment by another one. Unlike the method given by [Chan2] which reduces a fragment by a relation, we propose the reduction of one relation by another relation. This allows us to eliminate unnecessary relations and useless semi-joins without any modification to the theory developed in early sections. Furthermore, there is more parallelism in the sending of data in implementing a relation to relation reduction than a relation to fragment reduction.

We now find the least cost strategy in performing a semi-join Ri-A->Rj, where Ri and Rj may be fragmented.

In our environment, a fragmented relation may have a placement dependency with another fragmented relation. If such a dependency between Ri and Rj exists, then a fragment of Ri has a non-empty join with a fragment of Rj if they are located at the same site. As an example, if Ri and Rj are an employee relation and a relation containing the dependents of the employees respectively, then it is common that information about an employee and his/her dependents are located at the same site.

There are four cases to be considered in performing the semi-join: Ri-A->Rj.

Case 1: Ri and Rj are fragmented relations. If there is a placement dependency between them, then the cost is 0, otherwise, we proceed as follows.

Let the fragments of Ri and Rj be {Fik, $1 \le k \le n$} and {Fje, $1 \le e \le m$} respectively. A strategy to perform the semi-join from Ri to Rj is to send each fragment of Ri projected on A to each of a subset of s sites containing the fragments of Rj and sending the remaining (m-s) fragments of Rj to one of the s sites, where the value of s and the subset of s sites are to be determined so that the cost of sending is minimized.

Assuming that the value of s has been determined, the choice of the s sites is given as follows.

For each site containing a fragment of Rj, say Fje and possibly a fragment of Ri, say Fik, define the weight of the site to be

$$\begin{cases} |Fje|+|Fik(A)| & \text{if the site contains Fje and Fik} \\ |Fje| & \text{if Fik is absent from the site} \end{cases}$$

Arrange the sites containing the fragments of Rj in ascending order of weight and rename them i.e. weight(Sj1) $\le$ weight(Sj2) $\le$ ... $\le$ weight(Sjm). Then the subset of s sites containing the fragments of Rj are $\{Sj_{m-s+1}, \ldots, Sjm\}$. It is easy to see that the cost of sending for this subset of s sites is $H(s) = s * \sum_{k=1}^{n} |Fik(A)| + |Rj| - \sum_{k=m-s+1}^{m} weight(Sjk)$ and this cost is less than or equal to the cost of sending for another subset of s sites.

To determine the proper value for s, we invoke the following algorithm:

Algorithm ChooseS

s = 1;
while ($\sum_{k=1}^{n}$|Fik(A)| < weight(Sj_{m-s})) s=s+1;
return (s);

Proposition 6.1: The minimum cost (among all strategies given above) of executing a semi-join from Ri to Rj is obtained by choosing s as given by Algorithm ChooseS and those s sites are the ones having highest weights.

Sketch of proof: It can be proved that
(i) if $H(s+1)-H(s) \ge 0$, then $H(t+1)-H(t) \ge 0$, for all t>s, and
(ii) if $H(s)-H(s-1) < 0$, then $H(u)-H(u-1) < 0$, for all u<s.

Thus, we should find an s such that $H(s+1)-H(s) \ge 0$ and $H(s)-H(s-1) < 0$. Algorithm ChooseS finds such an s. []

case 2: Ri is a relation and Rj is a fragmented relation.

(a) Ri has not been reduced (i.e., no semi-join of the form Rk-->Ri has been executed). The secondary copies of Ri, if exist, will be used to save transmission time. If data transmission is necessary to perform the semi-join, a strategy similar to that given in case 1, with minor modification of parameters, is applicable. The details are not given here.

(b) If Ri has been reduced, then we can use the primary copy of Ri only (i.e., all other copies of Ri, which may have different contents will not be used).

case 3: Ri is a fragmented relation and Rj is a relation without fragments. Send all fragments of Ri to the site containing the primary copy of Rj.

case 4: Both Ri and Rj are unfragmented relations. If Ri has been reduced, then the primary copy of Ri should be used to reduce the primary copy of Rj, otherwise either the primary copy of Ri or a secondary copy of Ri is used to reduce the primary copy of Rj, depending on which is less in cost.

7 Estimation of Size

As mentioned early, our algorithm, after executing a semi-join will return the number of tuples of the reduced relation. Thus, before each semi-join is executed, the exact size of each relation and therefore the exact cost of the semi-join, are known. It remains to estimate the size of a relation Rj, to be reduced by a semi-join Ri-A->Rj.

Let ANj and BNj be the number of tuples of Rj after and before the semi-join is performed, ARj(A) and BRj(A) be the estimated number of distinct values of Rj on A after and before the semi-join is performed.

Then ANj is estimated to be

$$BNj * ARj(A) / BRj(A)$$
$$= BNj * ASj(A) / BSj(A),$$

where ASj(A) and BSj(A) are the selectivities of relation Rj on A after and before the semi-join is executed and the selectivity of a relation on A is the number of distinct values of the relation on A divided by the total number of possible distinct values of A [HeYa, SDD1, etc.]. The estimation of the selectivity of a relation on an attribute has been given early [SDD1, Luk1]. However, unlike their methods, we make use of the exact numbers of tuples of Rj and other relations (involved in previous semi-joins) to obtain more accurate

estimates for ASj(A) and BSj(A). Furthermore, BNj is obtained exactly. Thus, we expect our estimation to be more accurate. The size of the reduced Rj is given by the estimated number of tuples of the reduced Rj times the size of each tuple of Rj. If attribute A is eliminated after the semi-join is executed (for example, in example 4.1, after S1 has been executed, attribute A is eliminated), the size of a tuple of Rj needs to be modified and the number of tuples of Rj on the remaining attributes can be obtained using the result of [Yao].

Example 7.1: Let R1, R2 and R3 be three single-attribute relations. Let C be their common attribute having the values {0,1, ..,9}. Assume R1={0,5,9}, R2={0,2, 3,5,7,8,9} and R3={0,1,2,3,4, 5,6,8,9}. After the execution of R1-C->R2 and R2-C->R3, R3 has 3 tuples {0,5,9}. However, by [SDD1, HeYa], R3 is estimated to have 10*0.3*0.7*0.9 = 1.89 tuples. By our proposed estimation method, before the execution of R1-C->R2, BN2=7, BS2(C)=0.7 and AS2(C)=0.3*0.7 = 0.21. So, R2 reduced by R1 is estimated to have AN2 = BN2*AS2(C)/BS2(C) = 7*0.21/0.7 = 2.1 tuples. After the execution of R1-C->R2, the returned R2 has 3 tuples {0,5,9}. The selectivity of the reduced R2 on attribute A is 0.3. Before the execution of R2-C->R3, BN3=9, BS3(C)=0.9 and AS3(C) = 0.3*0.9 = 0.27. Thus, R3 reduced by R1 and R2 is estimated to have AN3 = BN3*AS3(C)/BS3(C) = 9*0.27/0.9 = 2.7 tuples. It is clear that our estimation method is more accurate. []

8 Sending in Complement

If a relation has more than half of all the possible attribute values in a given attribute, it is cheaper to send its complement, which has fewer attribute values, than the actual relation values on the attribute. In fact, it was demonstrated in [YuCC] that the improvement in data transmission cost of sending relations in complement or not in complement over sending the relations not in complement is gigantic in processing simple queries. The average improvement can range from twenty percent to a few hundred percent for simple queries referencing two to three relations, depending on the selectivities of the relations. As this type of queries occur rather often in practice, the technique of sending a relation in complement is worthwhile incorporated into a distributed query processing algorithm.

The cost of a semi-join Ri-A->Rj can now be computed as follows.

When Ri is partitioned into a number of sites, for each fragment of Ri under attribute A, say Fij(A), if the number of distinct values of Fij(A) is more than half the distinct values of attribute A, then Fij(A) is sent in complement; otherwise it is sent not in complement. When Ri is not fragmented, Ri projected on attribute A is sent in complement if there is a saving in data transmission. This process should be invoked only if the saving in data transmission is more than the local processing cost in converting a relation on certain attributes to its complement.

9 Integration

We now integrate the various techniques given in the previous sub-sections into one algorithm.

The first step is to choose a primary copy and a number of secondary copies of each relation referenced by the query using the algorithm developed in section 3. The cost of each semi-join from one fragmented or unfragmented relation to another is computed by the results of section 8 and 6. The benefit of each possible semi-join is computed using the estimation technique of section 7. Then a semi-join to be executed next is generated. For example, SDD-1 version 1 uses the least cost semi-join with benefit > cost, while SDD-1 version 2 chooses the semi-join with highest (benefit - cost). Other algorithms [HeYa, ChHo, YLCC, etc] use different criteria to select the next semi-join to be executed. Before the execution of the semi-join, the results of section 5 are used to check whether the semi-join is useless or replaceable by a better semi-join. If it is useless, then it is discarded and the next semi-join will be selected. If there is a strictly better semi-join, then it is replaced by the better semi-join. If the semi-join is not useless and there is no strictly better semi-join, then the semi-join is cheched whether it can be executed with less cost by sending in complement. After the semi-join, say Ri-A->Rj, is executed, Ri may be eliminated by the result of section 4. The cost and the benefit of the affected semi-joins are updated and the next semi-join is generated. This cycle of generating and executing a semi-join is repeated until no profitable semi-join can be obtained [SDD1] or some other criterion is used. Then all relations which have not been eliminated (section 4) are sent to a site to produce the answer.

10 Conclusion

We have sketched an algorithm to process queries in a fragmented database environment.

The algorithm has the properties:

(1) It is extremely simple. There is no difficulty in implementing the algorithm.
(2) It makes use of redundant copies of relations to reduce communication cost and selects a primary copy and some secondary copies of each relation to process a given query. This feature is not discussed in early papers.
(3) An efficient algorithm is given for fragment processing.
(4) All relations that participate in the sequence of semi-joins executed in the first phase and need not be sent to the assembly site in the second phase are identified.
(5) Semi-joins involving eliminated relations are either useless or replaced by better semi-joins. As a result of (4) and (5), the algorithm is more efficient than SDD-1 in minimizing total cost.
(6) An estimation procedure, based on dynamic execution of semi-joins is provided. This allows more accurate estimation than those given in early papers.

A version of the algorithm described in this paper is operational at System Development Corporation. Numerous variations of this algorithm are tested in performance which is defined in terms of total time, response time and local processing cost.

11 References

[AHU] Aho, A. V. Hopcroft, J. E. and Ullman, J. D. "The Design and Analysis of Computer Algorithms", Addison-Wesley, Reading, Mass 1974.

[BFMM] Beeri, C., Fagin, R., Maier, D., Mendelzon, A. and Yannakakis, M., "Properties of acyclic database schemes", STOC, 1981, pp. 355-362.

[BeCh] Bernstein, P.A. and Chiu, D-M.W., "Using Semi-join to Solve relational Queries", JACM, 1981 pp. 25-40.

[BeGo] Bernstein P.A., and Goodman N. "The theory of semi-join", Technical Report, CCA, Nov. 1979.

[BlLu] Black, P.A. and Luk W.S. "A New Heuristic for Generating Semi-join Programs for Distributed Query Processing", IEEE COMPSAC, 1982.

[Chan1] Chang J.M.. "A Heuristic Approach to Distributed Query Processing", Proc. VLDB, 1982.

[Chan2] Chang J.M. "Query Processing in a fragmented database environment" Bell Lab. Technical Report, 1982.

[Chiu] Chiu, D-M.W. "Optimal Query Interpretation for Distributed Databases", Ph.D. thesis, Harvard Uni. 1979.

[ChHo] Chiu, D-M.W. and Ho, Y.C., "A Method for Interpreting Tree Queries into Optimal Semi-join expressions", Proc. ACM SIGMOD Int. Conf. on Man. of Data, 1980, pp. 169-178.

[Codd] Codd E.F. "A Relational Model of Data for Large Shared Banks", CACM. 1970.

[Date] Date, C.J. "A introduction to Database System", Addison-Wesley, Reading, MA, 1977.

[FaMU] Fagin, R., Mendelzon, A. and Ullman, J., "A simplified universal relation assumption and its properties", IBM technical report, 1980.

[GoSh] Goodman, N. and Shmueli, O. "The Tree Properties is Fundamental for Query Processing", ACM SIGACT-SIGMOD Conference on Principles of Databases, 1982.

[Grah] Graham, M.H., "On the universal relation", TR, Univ. of Toronto, sept. 1979.

[HeYa] Hevner, A. and Yao, S.B. "Query Processing in Distributed Database Systems", IEEE trans. on Software Engineering, Vol. SE-5, No. 3, (May 1979), pp. 177-187.

[Hevn] Hevner, A. "Query Optimization in distributed database system" Ph.D. thesis, Purdue University, 1979.

[KeYa] Kerschberg, L. and Yao, S.B. "Optimal Distributed Query Processing", Bell Telephone, Holmdel, 1980.

[Luk1] Luk, W.S. and Black, P.A. "On Cost Estimation in Processing a Query in a Distributed Database System", IEEE COMPSAC, 1981.

[Luk2] Luk, W.S. and Luk, L. "Optimizing Query Processing strategies in a distributed Database System", Simon Fraser Univ, Burnaby B.C.,Canada.

[SDD1V1] Goodman, N., Bernstein, P.A., Wong, E., Reeve, C. and Rothnie, J.B., "Query Processing in a system for Distributed Databases", CCA Technical Report, 1979.

[SDD1V2] Bernstein, P.A., Goodman,N., Wong, E., Reeve, C., and Rothnie, J.B., "Query processing in a system for distributed databases (SDD-1)", ACM TODS, Vol. 6, No. 4, Dec. 1981, pp. 602-625.

[WDHL] Williams, R., el.ta., "R*: An Overview of the Architecture", IBM Research, San Jose, California, USA.

[Wong] Wong, E., "Retrieving Dispersed Data from SDD-1: A system for distributed Databases", Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, 1977.

[Yao] Yao, S.B., "Appoximating Block Accesses in Database Orgnization", CACM, Vol. 20, no. 4, 1977.

[YuOz] Yu, C.T., Ozsoyoglu, M.Z. "An algorithm for tree-query membership of a distributed query". IEEE COMPSAC 1979, pp. 306-312.

[YuLO] Yu, C.T., Lam, K. and Ozsoyoglu, M.Z. "Distributed Query Optimization for tree Queries", Dept. of Information Engineering, U. of Illinois at Chicago Circle, Il, 1979.

[YLCC] Yu, C.T., Lam, K., Chang, C.C., Chang, S.k. "A Promising Approach to Distributed Query Processing", Berkeley Conference on Distributed Data Base, Feb., 1982, pp. 363-390.

[YuCC] Yu, C.T., Chang, C.C. and Chang, Y.C. "Two Surprising Results in Processing Simple Queries in distributed Databases", IEEE COMPSAC, Nov., 1982.