DISTRIBUTED QUERY PROCESSING IN A RELATIONAL DATA BASE SYSTEM

Robert Epstein
Michael Stonebraker
Eugene Wong

Electronics Research Laboratory
College of Engineering
University of California, Berkeley 94720

ABSTRACT: In this paper we present a new algorithm for retrieving and updating data from a distributed relational data base. Within such a data base, any number of relations can be distributed over any number of sites. Moreover, a user supplied distribution criteria can optionally be used to specify what site a tuple belongs to.

The algorithm is an efficient way to process any query by "breaking" the qualification into separate "pieces" using a few simple heuristics. The cost criteria considered are minimum response time and minimum communications traffic. In addition, the algorithm can optimize separately for two models of a communication network representing respectively ARPANET and ETHERNET like networks. This algorithm is being implemented as part of the INGRES data base system.

KEYWORDS AND PHRASES: Distributed databases, relational model, distributed decomposition, communication networks, distribution criteria.

I Introduction

In this paper we are concerned with algorithms for processing data base commands that involve data from multiple machines in a distributed data base environment. These algorithms are being implemented as part of our work in extending INGRES [HELD75, STON76] to manage a distributed data base. As such, we are concerned with processing interactions in the data sublanguage, QUEL. The specific data model that we use is discussed in Section II. Some of our initial thoughts on these subjects have been presented elsewhere [STON77, WONG77].

We are not concerned here with control of concurrent updates or multiple copies [THOM75, LAMP76, ROTH77, CHU76]. Rather we assume that these are handled by a separate mechanism or can be integrated into our algorithms.

This paper is organized as follows: In section II we formalize the problem by indicating our view of a distributed data base and the interactions to be solved. Then, in section III we discuss our model for the computer network. In section IV a detailed algorithm is presented for handling the decomposition of queries in a distributed environment. There are a few complications concerning updates and aggregates in a distributed data base which are covered in sections V and VI. Lastly, in section VII we draw some conclusions.

Research sponsored by the U.S. Army Research Office Grant DAAG29-76-G-0245, and the Joint Services Electronics Program Contract F44620-76-C-0100.

II The Distributed Data Base Model.

We adopt the relational model of data [CODD70, CHAM76] and assume that users interact with data through the non procedural query language, QUEL [HELD75]. Algorithms for processing QUEL in a single machine environment (so called "decomposition") have been presented in [STON76, WONG76]. New algorithms or extensions are needed in a distributed environment. Some familiarity with the notion of decomposition will be helpful in understanding this paper.

The data base is assumed to consist of a collection of relations $R_1, R_2, \ldots, R_n$. Each relation, R_i, may be at a unique site or may be spread over several sites in a computer network.

We shall refer to a relation as being "local" if it is stored entirely at one site, and "distributed" if portions of it are stored at different sites. By default relations will be assumed to be local unless explicitly stated to be otherwise. They can be explicitly created (or extended) to be distributed on all or a subcollection of the sites in the computer network. Call the sites $S_1, S_2, \ldots, S_n$. At a given site S_j there may be a portion (or fragment [ROTH77]) of R_i. Call the portion R_i^j.

We shall assume that each fragment is in fact a subrelation, i.e., a subset of the tuples, of a given relation. Hence there is no notion of fragments being projections of a given relation [ROTH77]. Supporting those more general fragments is infeasible given the current structure of INGRES.

A distribution criterion, which determines how tuples are to be allocated to fragments, may be optionally associated with each relation. If no distribution criterion exists, then tuples will be placed at the site where they happen to be processed. Figure 1 indicates one collection of relations, the fragments that are actually stored and their distribution criteria. An example query for such a data base is to find the job numbers supplied by the supplier named XYZ. In QUEL this is expressed as:

```
range of s is supplier
range of y is supply
retrieve into w(y.jno) where
     y.sno = s.sno
          and
     s.sname = "XYZ"
```

Note that this query involves fragments at all three sites.

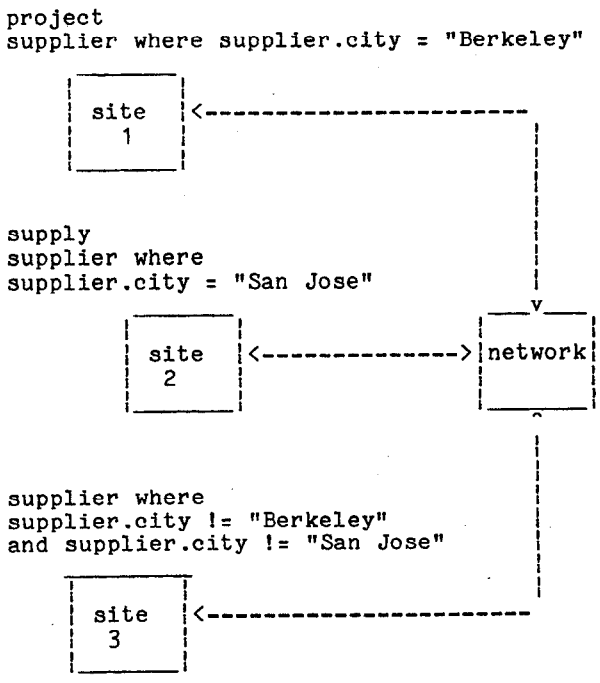
III Communications Network Model.

We are primarily concerned with two types of communication networks namely site-to-site and broadcast networks. In a site-to-site network we assume that there is a fixed cost to send one byte of data from any site to any other site. In a broadcast network we assume that the cost of sending data from one site to all sites is the same as that of sending the same data from one site to a single other site.

Figure 1
(A sample distributed data base)

The users view: supplier (sno, sname, city)
project (jno, jname, city)
supply (sno, jno, amount)

The distribution of Fragments:



Our site to site model of a network is motivated by the ARPANET [ROBE70] with one important simplification. In ARPANET the actual cost to communicate between two arbitrary sites will depend on what route the data must travel i.e. on the topology of the network.

Our broadcast model is similar to the ETHERNET [METC76] with two important simplifications. We assume that every site is always free to accept new messages. Thus a message will never have to be retransmitted because a site was too busy to accept the message. The ETHERNET restricts the recipient of a message to be either one site or every site. We assume a more general addressing scheme where anywhere from one to all sites can be addressed in a message.

Regardless of which network model is used, we shall assume that it is desirable to present large blocks of data to the network for transmission. In other words, bulk transmission is more efficient.

It will be shown that a broadcast (or ETHER) network is particularly well suited to a relational data base environment. The query processing algorithm can take explicit advantage of the broadcast capability of such a network. However, it is our intention to have INGRES support a data base on either type of network.

IV Query Decomposition.

INGRES supports four different data manipulation commands: retrieve, append, delete, and replace. As mentioned in [STON76], all update commands (append, delete, and replace) are actually processed by converting them to retrieve commands (to discover what to do) followed by low level file manipulation operations. Thus, the algorithm to decompose queries on a distributed system can be independent of the whether it is an update or retrieval until the very end. For this reason, we shall restrict our discussion here to "retrieves" and cover the relevant problems of updates in the next section.

Optimization Criteria.

Minimizing response time and minimizing network traffic will be taken to be the two main optimization criteria in a distributed data base environment. Minimizing response time involves minimizing the amount of processing needed to solve a query and using as much parallelism as possible from the various computer sites. Minimizing network traffic involves transmitting only the minimum data needed to solve the query. These two criteria are not unrelated. An increase in network traffic will improve response time if it results in greater parallel processing.

Network Decomposition Algorithm.

We first present the skeleton of the basis algorithm. We subsequently discuss the detailed optimization tactic involved.

The algorithm to decompose a query has the following inputs:

1) the conjunctive normal form of the query (i.e. the qualification is a collection of clauses separated by AND's; each clause containing only OR and NOT).
2) the location of each fragment and its cardinality.
3) the network type (site-to-site or broadcast).

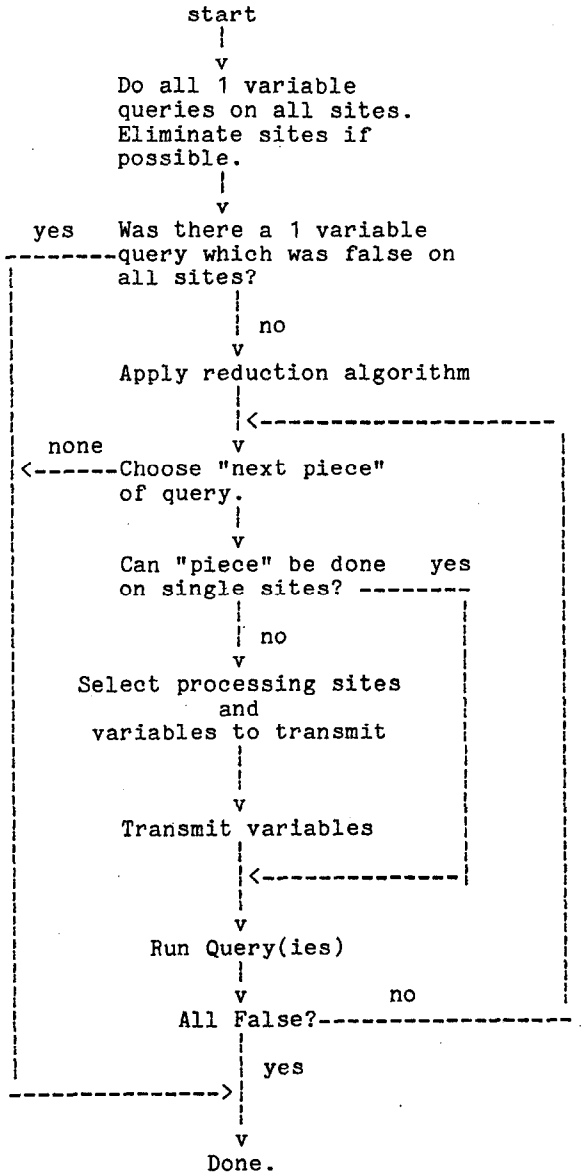
The algorithm is presented within the flowchart in Figure 2. The particular site where the query originates is called the "master" INGRES site. The master INGRES communicates with one "slave" INGRES at each site that is involved in processing the query. These "slaves" can be created by the "master" when appropriate. There are two types of commands that a master INGRES can give to a slave INGRES.

1) run the (local) query Q.
2) move the (local) fragment R_i of relation R to a subset of the sites in the network, S_1, S_2, ..., S_m.

The algorithm proceeds as follows:

(1) Do all one variable sub-queries. This has been shown in [YOUS78] to be almost always a good idea for non-distributed data bases. It should be equally true for distributed data bases.

Figure 2



Note that in the example query from section II, we have a one-variable subquery:

```
range of s is supplier
retrieve into temp(s.sno)
        where s.name = "XYZ"
```

In step (1) the master INGRES at the site where the query originated instructs a slave at each of the three sites to run the above subquery. The result is a fragment of temp at each site. The original query now becomes:

```
range of t is temp
range of y is supply
retrieve into w(y.jno)
        where y.sno = t.sno
```

Before actually running such a subquery, if the relation has a distribution criterion, check to see if a clause in the distribution criterion at site i contradicts a clause in the subquery. In general this requires a propositional calculus theorem prover. However, there are some simple cases for which contradiction is easily determined. For example, if the user's query includes a clause such as

... and supplier.city = "San Francisco"

examining the distribution criterion for site i might show

supplier where supplier.city = "Berkeley"

Thus without actually running the sub-query on site i, the portion of supplier on site i can be eliminated from the query.

(2) If there is a sub-query that was not satisfied on any site in (1), the entire query is false and we are done. This would happen in our example if temp had no tuples at all sites.

(3) Apply the reduction algorithm [WONG76] to the query. This will recast the original query into a sequence of component queries, each of which is processed in order, independently.

Consider a query $Q(x_1,x_2,\cdots,x_n)$ in QUEL where each variable x_i references a relation R_i. Q is said to be <u>reducible</u> if it can be replaced by a sequence of two queries (Q', Q'') that overlap on only one variable, i.e.,

$$Q(x_1, x_2, \ldots, x_n)$$
becomes:
$$Q'(x_m, x_{m+1}, \ldots, x_n)$$
followed by:
$$Q''(x_1, x_2, \cdots x'_m)$$

where the range of x'_m is the result relation of Q'. For example the query:

```
range of s is supplier
range of y is supply
range of j is project
```

```
retrieve (s.sname)
        where s.sno = y.sno
        and y.jno = j.jno
```

can be reduced to two components namely:

```
retrieve into temp (y.sno)
        where y.jno = j.jno
```

followed by

```
range of t is temp
retrieve (s.sname)
        where s.sno = t.sno
```

It is shown in [WONG76] that the irreducible components of a query are unique, and an intuitive argument is presented which indicates that these components form an advantageous sequence of subqueries. To test this hypothesis, experimentation with actual data was recently undertaken by [YOUS78] and the results were convincingly affirmative. It should be equally advantageous in a distributed system.

This step will perform the reduction algorithm and arrange the irreducible components of the query in their unique sequence. We can then consider the subqueries independently.

The example query from section II is not reducible so this step has no effect.

(4) Choose the "next piece" of the query to process. A query consists of a target list and a qualification (which is in conjunctive normal form). We define a "piece" as one or more clauses and their associated target lists.

Based on the query structure and the size and location of the fragments, the next "piece" of the query to be processed is selected. The algorithm to do this will be explained shortly. In our example there is only one remaining clause, which therefore must be the next piece.

(5) If the piece to be run can be done on individual sites without moving portions of the relations, then we proceed to step (9). In our example temp is at three sites and supply is at one. Hence, data must be moved in order to proceed.

(6) Select the site(s) that will process the next piece of the query. Depending on the number of sites, and whether it is a site-to-site or broadcast network, anywhere from one to all possible sites may be chosen. Suppose for our example that all three sites are chosen.

(7) The sub-query must be two variable or more in order to reach this step. In order to process an n variable subquery, fragments from n - 1 relations must be moved, and the remaining relation will remain fragmented. Each site that does processing must have a complete copy of the n - 1 relations. If processing is done on a single site, it must have copies of all n relations.

For our example we can broadcast supply to all sites. Each site will then have all of supply and a fragment of temp and will process the query producing a local fragment w. The answer to the query is the distributed relation w.

Alternatively, we can choose to broadcast temp to all sites involved in the query. Here we can view supply as distributed but with zero tuples on two of the three sites. Hence, fragments of temp will be sent to site 2. Site 2 then processes the same local query as above to produce a w; while sites 1 and 3 have no work to do.

Lastly, we can choose to equalize the distribution of the tuples in the relation that remains fragmented so as to guarantee that all processing sites have the same amount of work to do. This requires sending each site a complete copy of temp, and moving one-third of the supplier tuples to each of the other two sites before proceeding as above.

(8) Move the selected relation fragments to the selected sites. Each site will be directed, in turn, to send a copy of its selected fragments to the sites selected in step (6).

An optimization here is to have each site send only the domains needed in the query. Thus a fragment can be projected and duplicates removed.

This step can take full advantage of a broadcast network since we are often broadcasting from one site to many sites.

(9) The master INGRES now broadcasts the query to the selected sites and waits for all sites to finish. Once the query has been transmitted, this step involves only local processing.

If no site finds the qualification to be true, then we know the query is false and no further processing need be done. Otherwise the clauses just run are removed from the query and the new range of the remaining variables is changed if necessary.

(10) Go to step (4)

The Optimization Problem.

As outlined in the flowchart, the processing algorithm that we have chosen involves several decisions: (1) What is the "next piece" of the query that should be processed? (2) Which sites should take part in the processing of that piece? (3) If the number of participating site is greater than one, which of the relations should be left fragmented? (4) Should the fragmented relation be equalized in size among the processing sites?

Choosing the "Next Piece".

We have chosen to solve a query by "divide and conquer". That is, a query will be broken into one or more parts and each part will be processed in turn. The algorithm we pursue can justifiably be called a "greedy" algorithm. We will try to optimize each step, individually, without explicitly looking ahead to examine the global consequences. A possible refinement is to consider the structure of the residual query when deciding what should be done. The trade off between cost and benefit remains to be studied.

Results from decomposition on a single site have shown that reducing a query into its irreducible components is a good heuristic. Thus, after all one variable subqueries have been removed, the reduction algorithm will be used to transform the original query into its irreducible components which overlap on none, or one variables.

$$Q \rightarrow K_1, K_2, \ldots, K_i$$

We could stop at this point and simply execute each component in order, which is what we do on a single site. However, since any one component may span multiple sites, it may be desirable to subdivide further. Every component contains one or more clauses:

$$K_i = C_1, C_2, \ldots, C_j$$

The question is whether we should process the entire component at once or subdivide it? The answer is to subdivide only if the size of the result relation from subdividing is smaller than the communication cost of transmitting the source relations needed to process it. Here is an example to intuitively illustrate what can be achieved.

Consider the database from section II with relations:

```
supplier (sno, sname, city)
project (jno, jname, city)
supply (sno, jno, amount)
```

and the query: "find the names of supplier-project pairs such that the supplier supplies the project and the two are in the same city." In QUEL this can be stated as:

```
range of s is supplier
range of j is project
range of y is supply

retrieve (s.sname, j.jname)
        where s.city = j.city
        and   s.sno = y.sno
        and   j.jno = y.jno
```

This query is irreducible and involves three variables. There is only one irreducible component (namely the entire query) and it involves three clauses. To keep the example simple we shall ignore site 3 from Figure 1 and pretend there are only two sites, which have:

site 1	site 2
project (200 tuples)	supply (400 tuples)
supplier (50 tuples)	supplier (50 tuples)

Assume that the tuples of the three relations have the same width. Now consider the costs and benefits of subdividing the component.

We need to examine the four possible choices:

```
(Q1) retrieve into temp(s.sname,
     s.city, y.jno)
       where s.sno = y.sno
```

```
(Q2) retrieve into temp(s.sname,
       s.sno, j.jname, j.jno)
         where s.city = j.city

(Q3) retrieve into temp(j.jname,
       j.city, y.sno)
         where j.jno = y.jno

(Q4) retrieve (s.sname, j.jname)
         where s.city = j.city
         and s.sno = y.sno
         and j.jno = y.jno
```

These are the only possible alternatives. Choosing any two clauses would be tantamount to Q4 since any two clauses involve all three relations.

Q4 would require moving 200 project tuples and 50 supplier tuples at which point we could complete the query. Therefore, any subdivision will have to do better than that.

Suppose we choose to do Q1. This would require moving 50 supplier tuples. The result relation would have to be moved in order to process the remaining query:

```
range of t is temp
range of j is project
retrieve (t.sname, j.jname)
        where t.city = j.city
        and   j.jno = t.jno
```

If s.sno were a unique identifier for supplier and each supplier supplied only one job, then the result from Q1 would be at most 50 tuples. Thus the total network cost would be moving 50 supplier tuples followed by at most 50 tuples of temp.

But also consider the worst case. Suppose that sno is not unique and the entire cross-product of supplier and supply if formed. It would have 50 * 400 = 20,000 tuples! The decision whether to subdivide a component must be based on the expected size of the result relation. Some method for estimating the result size is needed.

But suppose the worst happens and the result size explodes geometrically. We can throw the result away and simply revert to running the whole component. In that case we would lose the time spent processing the abandoned query.

The accuracy of estimating the result size depends on a careful examination of the semantic content of the query (e.g., whether a clause involves an equality), on information concerning the distribution of values of the domains, (e.g., sno may be a primary key for the supplier relation with no more than one tuple for each possible value), and on the number of variables in the target list and their sizes. While compiling and keeping extensive statistics is impractical, a value indicating whether a domain is nearly a primary key or not may be both useful and easy to keep.

Determining How to Process the "Next Piece".

The second part of the optimization problem is how to process a given subquery. That is, which sites should be used as processing sites and which relation fragments should be moved. The more sites involved, the greater the parallelism. However, using more sites may involve greater communication costs and delays.

Presumably we will want to make decisions which minimize some application dependent cost function which might look something like:

```
c1 * network_traffic + c2 * processing_time
```

To do this we will need to know how to calculate the amount of network traffic involved, the relative processing time, and the cost/benefits of equalizing the fragmented relation. We will now proceed to explain how to determine each of these. Notationally, we know there are:

N total sites in the network
n relations referenced in the "next piece"

We need to choose:

K sites to be processing sites
R_p as the relation to be left fragmented

Let's number the processing sites so that 1 <= j <= K. Furthermore, let's define M_i as the number of sites where R_i has data stored. At each processing site j we have a query involving

$$Q_j = Q(R_1, R_2, \ldots, R_p^j, \ldots, R_n)$$

Determining Network Cost Given R_p and K.

Assuming that one relation R_p is left fragmented, and K sites (out of N) participate in processing, then the fragments of R_i have to be moved as follows:

> for a processing site j, R_i^j is moved to K - 1 other sites.
>
> for a non-processing site j, R_i^j is moved to K other sites and any fragments of R_p^j are moved to any one processing site.

The total communication cost is given by

$$comm = \sum_{j=1}^{K} C_{K-1}\left[\sum_{i\neq p}|R_i^j|\right] +$$

$$\sum_{j=K+1}^{N}\left[C_K\left[\sum_{i\neq p}|R_i^j|\right] + C_1\left[|R_p^j|\right]\right]$$

where $C_K(x)$ denotes the cost of sending x bytes of data to K sites, a cost that depends on the network that is used.

Estimating Relative Processing Time.

Define proc(Q) to be the time required to process the query Q if it were done on a single site. If K is greater than one then the processing time of a query, Q, can be improved from proc(Q) to $\max_j$ proc(Q_j). In other words, the processing time for the whole query is equal to the processing time of the site j with the most processing to be done. It is reasonable to assume that the processing time at each site is given by

$$proc[Q_j] = \frac{|R_p^j|}{|R_p|}\ proc\ [Q]$$

so that the overall processing time is

$$\max_j\ proc[Q_j] = \max_j \frac{|R_p^j|}{|R_p|}\ proc\ [Q]$$

assuming that all sites are of approximately equal processing speed.

Cost for Equalization.

If each fragment of R_p was the same size, then the overall processing time would be given by 1/K proc(Q), since each fragment would have (1/K) of the data. (This assumes, again, that all computer sites are of approximately equal speed.) This is the motivation for equalization.

To determine the cost of equalizing, let us define the function pos(x) to be:

> if x > 0 then pos(x) = x
> if x <= 0 then pos(x) = 0

The amount of data to be moved in equalizing the fragments of R_p is given by

$$\sum_{j=1}^{N} pos\left[|R_p^j| - \frac{1}{N}|R_p|\right]$$

This added network cost would result in a processing improvement from $\frac{|R_p^j|}{|R_p|}$ to $\frac{1}{K}$. Thus it may be desirable to trade some network cost for an improvement in overall processing time.

We now know how to compute network communication cost, relative processing time, and the cost and benefit of equalizing R_p. We will now use this knowledge to minimize some example cost functions.

Minimizing Communication Costs.

For the moment let's assume that the overall optimization criteria is to minimize communication costs. It is important to treat site-to-site and broadcast networks separately. We will first consider using a broadcast network and solving for K and R_p.

For a broadcast network $C_K(x) = C_1(x)$ for all K >= 1. By examination of the communication cost function given on page 8 it can be seen that the communication cost function is always minimized by K = 1 or $K \geq M_p$. To see this observe that the cost function has three terms. The first term will be zero if K = 1. The third term will be zero if every site which has part of R_p is a processing site. Thus K must be $\geq M_p$, where M_p is the number of sites where R_p is present.

If we assume that $C_1(x)$ is linear in x and rearrange some terms then

$$[COMM]_{broadcast} = C_1\left[\sum_i |R_i| - \sum_i |R_i^1|\right] \text{ for } K=1$$

$$\left[\text{COMM}\right]_{\text{broadcast}} = C_1\left[\sum_i |R_i| - |R_p|\right] \quad \text{for } K=N$$

Hence, the decision rule for minimizing communication in the broadcast network case is given by

> If $\max_j \sum_i |R_i^j| > \max_i |R_i|$, choose K = 1 and choose the processing site to be the one containing the most data. In this case there is no R_p. In other words, if one site has more data than the largest relation, then K = 1 and choose that site.
>
> If $\max_j \sum_i |R_i^j| \leq \max_i |R_i|$, choose R_p to be the relation containing the most data and choose $K=M_p$.

The situation for site-to-site networks is quite different. For that case we shall assume $C_k(x) = k\, C_1(x)$ and that $C_1(x)$ is again linear in x. We note that, independent of K the choice of R_p that minimizes communications is the relation with the most data, i.e., $\max_i |R_i|$. Once R_p is chosen, the value of K that minimizes communications is determined as follows:

> Let the sites be arranged in decreasing order of $\sum_i |R_i^j|$. Then, choose K to be 1 if
>
> $$\sum_{i \neq p} \left[|R_i| - |R_i^1| \right] > |R_p^1|$$
>
> otherwise choose K to be the largest j such that
>
> $$\sum_{i \neq p} \left[|R_i| - |R_i^j| \right] \leq |R_p^j|$$

The interpretation of this decision rule is as follows. A site should be chosen as a processing site if and only if the data that it is to receive as a processing site is less that the additional data that it would have to send as a nonprocessing site.

If minimization of communication cost is the sole criterion of optimization, then the best choices for both K and R_p have been completely determined for both broadcast and site-to-site networks. We expect the following exceedingly simple rule to be reasonable for choosing the number of processing sites:

> Choose R_p to be the largest relation.
>
> if N = 2 and n = 2 then K = 2
>
> if $N \neq 2$ then $K = M_p$ for broadcast network and K = 1 for site-to-site network

Minimizing Processing Time.

Suppose for the moment that our goal was only to minimize processing time and we were willing to ignore any network costs or delays. In this case we would want K = N so that we could distribute the work among all sites. We would still need to choose one relation to remain fragmented. The actual processing time will be independent of whatever relation we choose. To see this notice that when we equalize the fragmented relation the processing time goes from $\max_j \frac{|R_p^j|}{|R_p|}\text{proc}(Q)$ to $\frac{1}{K}\,\text{proc}(Q)$, which is independent of R_p.

Since we can choose any R_p our choice for R_p should be the relation which minimizes network traffic. Thus we are looking for R_p which minimizes:

$$\sum_{j=1}^{N} C_{N-1}\left[\sum_{i \neq p} |R_i^j|\right] + \sum_{j=1}^{N} \text{pos}\left[|R_p^j| - \frac{1}{N}|R_p| \right]$$

Summary.

The true optimal solution cannot be determined without knowing the precise relationships between processing time and communication costs, the distribution of data among the N sites, and the true processing time for each site. Any optimal solution would have to consider the possibility of using any value of K from 1 to N.

Exactly how to choose R_p and K is an open ended question until the cost criteria have been specified. We have shown solutions at two extremes (minimum network traffic and minimum processing time).

Note the similarity between the "next piece" decision and the next tuple variable

to be chosen for substitution in single site decomposition. The same inability to achieve a true optimum is present in both cases.

Just as refinement of the decision making process was presented in [YOUS78] for decomposition, we plan to do experimentation concerning cases where each possible algorithm is best.

V Updates.

INGRES will be expanded to allow distributed relations with optional distribution criteria. When inserts occur they can cause tuples to be placed on specific sites. In the case of a replace command, they can cause tuples to be moved from one site to another.

We will assume that a distribution criterion maps a tuple to a unique site or to no site at all. It is possible that a criterion disallows certain values. Since the distribution criterion is related to an integrity constraint [STON75], tuples which cannot belong to any site could be treated as a violation of an integrity constraint.

With a distribution criterion we will guarantee that the collection of all the fragments of a relation contains no duplicates. Without a distribution criterion, any one fragment will contain no duplicates, but the collection of all sites can contain duplicates. To support any other rule seems too costly.

The processing of an update will proceed as follows:

(1) At the end of the query decomposition, one or more sites will have a portion of a temporary relation that holds the changes to be made to the result relation.

(2) Each site performs the updates specifically designated for it.

(3) The remaining updates (if any) are sent directly to their correct sites.

(4) Each site completes any new updates received.

The decomposition algorithm can help the update processing by never moving the result relation. For example, in the query:

```
range of p is project
range of s is supplier
delete p where p.city = s.city
```

decomposition has the choice of moving either p or s. By always choosing NOT to move p (the result relation variable), we optimize the update processing. This is because step (3) above will then have no tuples for other sites. However, choosing not to move the result relation, may be a poor tactic during decomposition. The trade-off cannot readily be determined without advance knowledge of how many tuples of the result relation satisfy the qualification.

We will now identify what must be done for each update command, with or without a distribution criterion.

For an append command, each tuple to be appended is put into a temporary relation and then split according to steps (1) to (4) above. If there is no distribution criterion, the tuples can be appended directly instead of going through a temporary relation.

For a delete command, the temporary relation consists of a tuple identifier (TID) and machine identifier (MID). The algorithm is the same regardless of the distribution criterion. Note that if we guarantee that the result relation is not moved during decomposition, we do not need the MID. The set of TIDs is only for the machine in which they reside. In fact, if the result relation is not moved during decomposition, the temporary relation is not needed. The deletes can be done directly.

For a replace command, the TID, MID, and the new tuple must be saved. If the relation does not have a distribution criterion the replace command is processed in the same manner as a delete. With a distribution criterion, it is possible that a tuple will have to be moved from its current site to another. If this happens, it must be deleted at the current site and marked to be appended at its new site. If decomposition moves the result relation and a tuple has to change sites because of a

replace, steps (3) and (4) must be repeated twice.

In order to provide for recovery from a system crash during an update, updates are not done directly, instead we use a "deferred update" file [STON76]. The deferred update file will contain a full image of all changes that must occur before any actual updates take place. The deferred update mechanism is hidden below the mechanism we have been discussing.

All update processing is controlled by the master INGRES. The individual sites can all perform the updates in parallel. If the result relation is never moved during decomposition, additional network costs can only occur in append, and replace commands where the distribution criterion forces the tuples to be moved to a new site.

VI Processing of Aggregates.

The current implementation of INGRES processes aggregates (min, max, avg, sum, and count) first and then decomposes the remaining aggregate-free query. Although this is not always optimal, it typically is and it is a simple query processing strategy.

For distributed INGRES we also plan to process all aggregates first. Some optimization specific to aggregates can be done. For example, aggregates that range over only one relation, and are done without removing duplicates are processed on individual sites and the aggregated results are transmitted back to the master site where they are combined to produce the final result.

Aggregates which involve more than one relation or which require duplicates to be removed, are processed by first retrieving the values to be aggregated into a distributed temporary relation, and then aggregating on that temporary relation. If the aggregate requires duplicates to be removed, then the temporary relation will have to be collected onto a single site in order to remove duplicates.

Other optimization techniques such as processing as many aggregates as possible in the same pass through the relation, and optimizing the by-domains references in aggregate functions, will continue to be performed.

VII Conclusions.

The model we propose for a distributed data base is very flexible. Portions of a relation can exist at any number of sites and an optional distribution criteria can be used to assign tuples to specific sites.

The algorithm for decomposing a query involves examining the structure of a query's qualification. The query is processed by choosing for processing one or more clauses of the qualification at a time. It is inevitable that some data will have to be moved from site to site in order to process a query. The algorithm tries to move only the smallest amount of data, and tries to get the maximum amount of parallel processing possible. In addition, by trying to avoid moving the result relation, we help to optimize the update processing. During query processing it is frequently desirable to broadcast data from one site to several other sites, which makes a broadcast network extremely desirable.

REFERENCES

[CHAM76] Chamberlin, D.D.; "Relational Data Base Management: A Survey," Computing Survey, Vol. 8, no. 1, March 1976.

[CHU76] Chu, Wesley W.; "Performance of File Directory Systems for Data Bases in Star and Distributed Networks," AFIPS Conference Proceedings, vol. 45, 1976.

[CODD70] Codd, E.F.; "A Relational Model of Data for Large Shared Data Banks," CACM vol. 13, no. 6, June 1970.

[HELD75] Held, G.D., M.R. Stonebraker, and E. Wong; "INGRES - A Relational Data Base System," Proc. NCC vol. 44, 1975.

[LAMP76] Lamport, L.; "Time, Clocks and Ordering of Events in a Distributed System," Mass. Computer Associates Report CA-7603-2911, March 1976.

[METC76] Metcalf, R. M. and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," CACM, vol. 19, no. 7, July 1976.

[ROBE70] Roberts, L. and Wessler, B., "Computer Network Development to Achieve Resource Sharing," Proc. SJCC, 1970, AFIPS Press.

[ROTH77] Rothnie, J.B. and N. Goodman; "An Overview of the Preliminary Design of SDD-1: A System for Distributed Databases," 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, May 1977.

[STON75] Stonebraker, M.R.; "Implementation of Integrity Constraints and Views by Query Modification", University of California, Electronics Research Laboratory, Memorandum ERL-M514, March 1975.

[STON76] Stonebraker, M.R., E. Wong, P. Kreps and G.D. Held; "Design and Implementation of INGRES," ACM Trans. Database Systems, vol. 1, no. 3, Sept. 1976.

[STON77] Stonebraker, M.R. and E. Neuhold; "A Distributed Database Version of INGRES," 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, May 1977.

[THOM75] Thomas, R.H.; "A Solution to the Update Problem for Multiple Copy Databases Which Use Distributed Control," BBN Report 3340, Bolt Beranek and Newman Inc., Cambridge, Mass., July 1975.

[WONG76] Wong, E. and K. Youssefi; "Decomposition - A Strategy for Query Processing," ACM Trans. Database Systems, vol. 1, no. 3, Sept. 1976.

[WONG77] Wong, E.; "Retrieving Dispersed Data from SDD-1: A System for Distributed Databases," 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, May 1977.

[YOUS78] Youssefi, K.; "Query Processing for a Relational Database System," Ph.D Dissertation, University of California, Berkeley, 1978, Electronics Research Laboratory, Memorandum UCB/ERL M78/3, January 6, 1978.