Computer Communication Technology and its effects on Distributed Query Optimization Strategies

J. M. Morrissey and S. Bandyopadhyay*

School of Computer Science
University of Windsor
Windsor, Ontario, Canada N9B 3P4
Phone : (519) 253–4232 extn 2999; e-mail : subir@cs.uwindsor.ca

* Support received from NSERC grant numbers 8–9214 and 8–9145.

Abstract **The premise of this paper is that query optimization in distributed databases can be improved by taking into account certain parameters such as the network architecture and communication delays. Current heuristics make simplistic assumptions, disregarding the actual characteristics of the network. Here, we assume that the network and distributed database managers will communicate and propose a heuristic which utilizes available delay information to minimize the total response time. Initial experiments indicate that our heuristic performs well in comparison to other distributed query optimization algorithms.**

I. INTRODUCTION

Distributed database management systems (DDBMS) [9] offer many advantages including increased reliability, more efficient processing of local queries and increased data availability. However, query optimization in such systems is a very important task. The objective is to select a sequence of operations and data transfers so that some cost function in minimized. Different cost functions have been proposed including the total amount of data transferred over the network; the total CPU processing costs; and the total amount of time taken to process the query. The latter is our primary concern here, where we assume a relational DDBMS consisting of a number of independent nodes connected via a point-to-point network. We assume that each node has local processing capabilities and can access all data.

Distributed query processing research has concentrated on joins [1, 14], semi-joins [2, 3, 5, 10, 11, 13, 15, 16] or a combination of both [12]. There is also some interest in improving sub-optimal solutions produced by heuristics [6, 7]. Here, we are concerned with semi-join strategies, where semi-joins are executed so as to reduce the size of intermediate relations and thus minimize the time required to process the query. A semi-join from R_a to R_b, denoted $R_a \rtimes R_b$, is executed by first projecting R_a over the common join attribute; shipping the projection to the site of R_b; and then performing the join of R_b and the projection. The size of R_b is reduced since the semi-join eliminates tuples which can not be part of the result. However, it is important that the cost of the semi-join is less than its benefit. The cost is defined as the delay in transmitting the projected attribute (of R_a) to the site of R_b; the benefit is defined as the expected reduction in the delay to transmit R_b to the query site, due to tuples being eliminated by the semi-join. A central concept in our heuristic is that a semi-join will only be executed if the benefit outweighs the cost.

In common with other query processing algorithms we assume select-project-join queries and a three phase approach: selections and projections are first performed locally to reduce the relations as much as possible; semi-joins are then used to further reduce the relations; and finally, all relations are shipped to some query site for assembly. Clearly, fast and efficient communications is essential for minimizing the query processing time. Current optical communications technology [8] allows for very high speed data transfers but typically query optimization algorithms do not take into account such details. Here we propose that query optimization heuristics utilize available information regarding the actual communication setup and delays on the network. The cost and benefit of a semi-join are determined by two factors: the amount of data to be transferred and the delay on the communications path between the two nodes. Statistics are available to the distributed database manager so that the amount of data can easily be estimated. Currently a simplistic assumption is made by query optimization algorithms: the time to send data from a source to a destination is determined only by the amount of data and is independent of the actual communication path. For packet switched systems this is not true since different paths may require a different number of edges and the delay, in general, differs from edge to edge.

In this paper we use a more realistic cost model to estimate the time τ to communicate ρ packets from source s to destination d. Our delay model is given below :

$$\tau = c(s, d) * \rho + c_{set\text{-}up}$$

where $c(s, d)$ is the cost to send a packet from source s to destination d and $c_{set\text{-}up}$ is the set up time. It is important to note that, in our approach, both $c(s, d)$ and $c_{set\text{-}up}$ are variables whose values are determined by network parameters, at the time of formulating the semi-join optimization strategy. In this paper, we are ignoring $c_{set\text{-}up}$ and we assume that query optimizers may access a global delay table maintained by the routing manager which gives $c(s, d)$, the expected delay per packet from any source node to any destination node. These delay tables are updated, by the routing manager, at intervals to reflect changes in loads on different edges of the network. Our approach is to use current delay table values when determining our query processing strategy.

0-7803-2766-7-9/95/$4.00 © 1995 IEEE

Determining an optimal semi-join strategy is NP-hard [6, 7] so our approach is to develop a heuristic which establishes a sequence of semi-joins to efficiently reduce the response time.

2. Heuristic for determining schedule

We now describe our approach briefly, beginning with some definitions. We have a query Q at site QS which has the following format :

$$Q = R_0 \bowtie R_1 \bowtie R_2 \bowtie \;.\;.\;.\; \bowtie R_n$$

where relation R_i is at site S_i. We assume that no two relations are at the same site and that QS is distinct.

Definition

If relations R_a and R_b have attributes d_{aj} and d_{bj} (defined on the same domain) then d_{aj} (d_{bj}) is a *potential reducer* for relation R_b (R_a).

Definition

If attributes d_{aj} and d_{bj} are defined on the same domain then d_{aj} (d_{bj}) is a *potential reducer* for attribute d_{bj} (d_{aj}) (provided $a \neq b$, that is they are not part of the same relation.)

Definition

A reducer is *beneficial* if the time to generate and ship it to R_a is less than the reduction in time required to send (the reduced) R_a to QS.

We compute the communication time to send each of the relations R_i, $0 \leq i \leq n$, to the query site. We choose R_x such that its communication time is a maximum. Since this communication time degrades the response time the most, we attempt to reduce this relation as much as possible using a cost and benefit analysis. Let d_{bj} be a potential reducer for R_x. We recursively attempt, using a greedy heuristic, to reduce d_{bj} using semi-joins. For example, we may decide that the best strategy for reducing R_x is to execute $d_{aj} \ltimes d_{bj} \ltimes R_x$. In this case, the cost is the sum of the time required to send d_{aj} to d_{bj} plus the time to send $d_{aj} \ltimes d_{bj}$ to R_x plus the time to send the reduced R_x to the query site. Clearly this cost must be less than the cost of sending the unreduced R_x to QS.

This process is repeated for the relation having the next largest communication time. The process stops when every relation

- has been reduced or
- has a communication cost less than the worst (reduced) communication time.

The algorithm is outlined below:

W = 0

L ← List of all relations R_i, $0 \leq i \leq n$

While L is not empty

  Pick the relation R_j such that time T_j to send R_j to QS is the largest of all relations.

If $T_j < W$, break out of the loop

T_{opt} ← find_best_schedule(QS, R_j, W)

If $T_{opt} > W$, W ← T_{opt}

Remove R_j from L.

The function find_best_schedule determines the best way to reduce R_j and returns the time for communicating the reduced R_j to QS. The function attempts to reduce the time needed to transfer R_j to QS to below W, where W is the worst communication time for any reduced relation. As soon as this happens the function does not attempt any further reduction since it is futile to do so (since it will not reduce the response time). So long as the objective is not reached, the function will attempt to find beneficial reducers. An example in the next section illustrates our approach.

3. An Example

Consider the query $Q = R_1 \bowtie R_2 \bowtie R_3$. The data are given in Table 1 below. This example is taken from [2] and we use the same notation and terminology used there.

R_i	S_i	d_{i1}		d_{i1}	
		b_{i1}	ρ_{i1}	b_{i1}	ρ_{i1}
R_1	1000	400	0.4	100	0.2
R_2	2000	400	0.4	450	0.9
R_3	3000	900	0.9	-	-

S_i gives the size of R_i, in some suitable unit such as bytes or packets. There are two join-attributes, d_{i1} and d_{i2}. The size of attribute d_{ij} is given by b_{ij}; the selectivity of d_{ij} by ρ_{ij}.

In addition we need the current delays between relation sites. In this example we use the following delay table :

		Destination Sites			
		R_1	R_2	R_3	QS
Source Sites	R_1	-	2	5	3
	R_2	1	-	7	2
	R_3	5	6	-	4

We now show how our heuristic generates a schedule for this data. Initially the worst time W is 0.

Step 1) The expected delays in communicating R_1, R_2 and R_3 to the query site QS are as shown in Fig 1. This shows that R_3 has the largest delay and has to be minimized first.

Step 2) Possible potential reducers for R_3 are attributes d_{11} and d_{21}. Since the delay for communicating d_{21} from its site S_2 to site S_3 is greater than that for d_{11}, the schedule involving d_{21} will have parallel transmission of d_{11} to S_3 (Fig 2).

Step 3) We now check the possibility of reducing the reducer d_{11}. Attributes d_{11}, d_{21} and d_{31} are defined on the same domain and hence may be used to reduce one another. When we consider the possibility of using d_{21} to reduce d_{11}, the cost is the time to communicate d_{21} to site S_1. The size of d_{21} is 400 packets and the delay from S_2 to S_1 is 1 so that the cost is 400 units of time. The corresponding benefit is due to the following :

- the selectivity of d_{21} is 0.4 so that the reduction in size of d_{21} is 400 * (1 − 0 .4) = 240 packets. The delay in communicating $d_{21} \rtimes d_{11}$ to S_3 is reduced by 240 * 5 = 1200 units of time.
- the selectivity of $d_{21} \rtimes d_{11}$ is 0.4 * 0.4 = 0.16 instead of 0.4 for d_{11} alone so that the reduction in the number of tuples of R_3 that need be communicated to query site is 3000*0.4 *(1− 0.4) = 720. The delay in communicating the reduced R_3 to QS is 720 * 4 = 2880.

In this case the cost is less than the benefit and $d_{21} \rtimes d_{11}$ is a better reducer than d_{11}.

Step 4) We look at reducer d_{21} in a similar way. It turns out that $d_{21} \rtimes d_{11}$ is the best reducer for R_3 giving us the schedule shown in Fig 3. W is now 3600.

Step 5) In a similar way, the schedules for R_1 and R_2 are formed (Fig 4). We note that once the delay falls below W, we stop the process since we are trying to reduce the response time only. Other mixed strategies to minimize communication cost at the same time are clearly possible.

The schedule proceduced for this data by [2] is shown in Fig 5. In comparison, our method reduces the response time from 5200 to 3600 units.

4. Experimental results and Conclusions

We have tested our heuristic with some sample queries. Preliminary results suggest that our heuristic is substantially better than standard heuristics, producing schedules of lower total response time. Currently we are performing a large scale testing of the heuristic, using a large database of queries generated using methods similar to those outlined in [4]. Experimental results will be presented at conference time.

In this paper we have presented a heuristic which efficiently generates semi-join schedules for minimizing the total query response time. It is unique in that it takes into account certain network parameters such as communications delays, coupling network characteristics and query optimization heuristics. Our hypothesis is that such an approach is applicable to wide range of communication protocols and will lead to better optimization of distributed database operations.

5. Acknowledgments

We wish to thank colleagues for helpful discussions.

References

[1] A. K. Ahn and S.C. Moon. Optimizing joins between two fragmented relations on a broadcast local network. *Info. Syst.*, 16(2), 1991.

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

[3] P.A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. Rothnie. Query processing in a system for distributed databases (sdd-1). *ACM Trans. on Database Systems*, 6(4), 1981.

[4] D. Bitton, D.J. DeWitt, and C. Turbyfill. Benchmarking database systems in a systematic approach. In *Proceedings of the 9th International Conference on Very Large Databases*, 1983.

[5] P.A. Black and W.S. Luk. A new heuristic for generating semi-join programs for distributed query processing. *IEEE COMPSAC*, 581–588, 1982.

[6] P. Boderick, J. Pyra, and J. S. Riordan. Correcting execution of distributed queries. In *Proc. of 2nd Int. Symp. on Databases in Parallel and Distributed Systems*, 1990.

[7] P. Boderick, J.S. Riordan, and J. Pyra. Deciding to correct distributed query processing. *IEEE Trans. on Knowledge and Data Engineering*, 4(3), 1992.

[8] C. A. Brackett. Dense wavelength division multiplexing networks : Principles and applications. *IEEE Jour. Selected Areas in Communications*, 8(6), 1990.

[9] S. Ceri and G. Pelagatti. *Distributed Databases: Principles and Systems*. McGraw-Hill, 1984.

[10] L. Chen and V. Li. Improvement algorithms for semi-join query processing programs in distributed database systems. *IEEE Trans. on Computers*, 33(11), 1984.

[11] L. Chen and V. Li. Domain-specific semi-join: a new operation for distributed query processing. *Info. Sci.*, 52, 1990.

[12] M. Chen and P. S. Yu. Combining join and semi-join operations for distributed query processing. *IEEE Transactions on Knowledge and Data Engineering*, 5(3), 1993.

[13] H. Kang and N. Roussopoulos. Using 2–way semi-joins in distributed query processing. In *Proc. 3rd Int. Conf. on Data Engineering*, 1987.

[14] P. Legato, G. Paletta, and L. Palopoli. Optimization of join strategies in distributed databases. *Info. Syst.*, 16(4), 1991.

[15] N. Roussopoulos and H. Kang. A pipeline n-way join algorithm based on the 2–way semi-join program. *IEEE Trans. on Knowledge and Data Engineering*, 3(4), 1991.

[16] C. Wang, V. Li, and A. Chen. Distributed query optimization by one-shot fixed precision semi-join execution. In *Proc. 7th Int. Conf. on Data Engineering*, 1991.

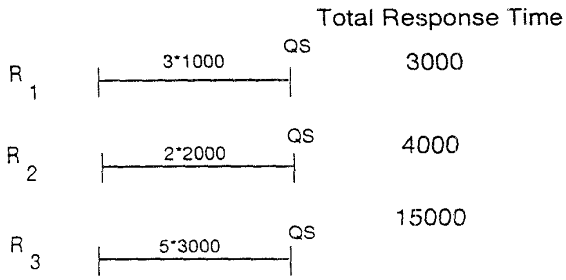

Fig 1 Communication time to query site

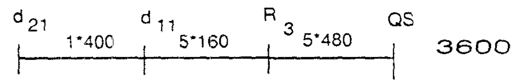

Fig 3 Final semi-join schedule for R_3

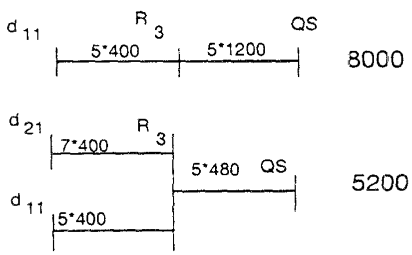

Fig 2 Some semi-join schedules for R_3

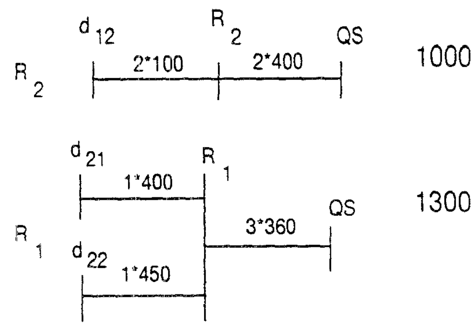

Fig 4 Final semi-join schedule for R_1 and R_2

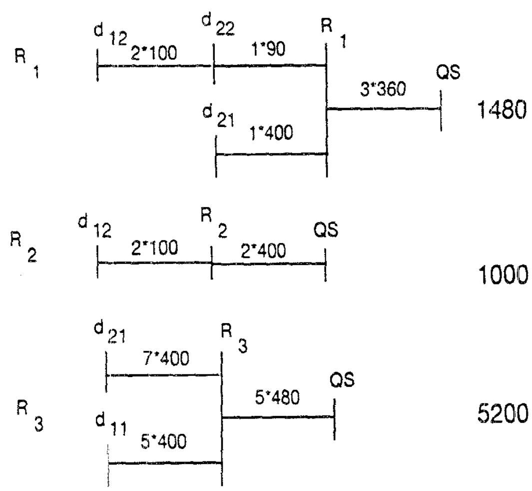

Fig 5 AHY Algorithm schedules taking delays