Subquery Allocations in Distributed Databases Using Genetic Algorithms

Narasimhaiah Gorla
American University of Sharjah,
PO Box 26666, Sharjah, UAE
and
Suk-Kyu Song
Youngsan University
Pusan, Korea

ABSTRACT

Minimization of query execution time is an important performance objective in distributed databases design. While total time is to be minimized for On Line Transaction Processing (OLTP) type queries, response time has to be minimized in Decision Support type queries. Thus different allocations of subqueries to sites and their execution plans are optimal based on the query type. We formulate the subquery allocation problem and provide analytical cost models for these two objective functions. Since the problem is NP-hard, we solve the problem using genetic algorithm (GA). Our results indicate query execution plans with total minimization objective are inefficient for response time objective and vice versa. The GA procedure is tested with simulation experiments using complex queries of up to 20 joins. Comparison of results with exhaustive enumeration indicates that GA produced optimal solutions in all cases in much less time.

Keywords: Physical Database Design, Genetic algorithms, Distributed database design, Subquery allocation, Response time minimization

1. INTRODUCTION

Distributed database systems have become very important and common in today's geographically distributed organizations. The performance problems relating data distribution and query processing in distributed databases are known to be critical issues [10]. Distributed database design and query optimization have been active research areas [21][27][13][19][15][5][14]. The design of distributed query processing can be divided into two aspects: query execution order and query execution plan or operation allocation. While query execution order specifies the order in which subqueries are executed, operation allocation indicates the sites from which the operations (subqueries) are executed. Queries can be classified into OLTP (On-Line Transaction Processing) and Decision Support [3]. "To inquire about airline seat availability" is an example of OLTP type query in the travel industry, which is a highly repetitive transaction that requires high throughput. A transaction such as "to provide sales details of specific routes by region and travel agent" in the travel industry is an example of decision support, which requires low response time. Thus, the query execution plans for OLTP and decision support queries need to be designed with total time minimization and response time minimization objectives, respectively. Total time minimization aims at minimizing the total resource consumption (I/O, CPU, and Communication) and maximizing overall throughput of the system, while the response time minimization aims at minimizing the time between query origination and result receipt. While OLTP and decision support queries are most common in the business world, previous research paid little attention to design distributed databases to optimize both these types of queries together. The optimization of both these types of transactions will make the database operations efficient, thus making the working environment of business decision makers more efficient. The objective of the paper is to present a methodology for a distributed database query optimization strategy that selects between total-time and response time (elapsed-time) cost functions according to the target application type. The queries are decomposed into subqueries and these subqueries are allocated among the nodes of the network so that the two objective functions (minimization of total execution time and minimization of response time) are optimized in order to meet the processing requirements of OLTP and Decision Support transaction types.

While most previous works focused on query execution order, operation allocation has received little attention. In today's geographically distributed organizations, since more sophisticated data access is needed by managers in areas such as decision support and deductively augmented database systems, answering OLTP and decision support type queries often requires a large number of joins [20]. If a query references n relations, and each relation R_i has X_i copies, $1 \le i \le n$, then a straightforward enumeration algorithm for selecting one copy of each relation takes time $O(\prod_{i=1}^{n} X_i)$ [20]. The problem of finding the minimum cost allocation is NP-hard. In order to deal with this hard problem, we use Genetic Algorithms [11] to arrive at near optimal solution. Genetic algorithm is a heuristic solution that has been used to solve intractable

problems in database design [5] [8] [22]. Next, we present prior research in distributed database design and query processing. Section 2 has discussion of cost models. Section 3 has research results based on genetic algorithm. Section 4 has conclusions.

Previous Research

[15] provides a survey of techniques useful for query processing in distributed databases. In order to improve distributed database performance, two types of problems are important: data allocation (how to allocate data fragments to sites) and operation allocation (how to allocate subqueries to sites).

Regarding data allocation problem, [1] developed a methodology for identification and allocation of vertical and horizontal fragments based on the user queries /updates in distributed databases, with the objective of minimizing total transmission cost. [25] proposed an integrated methodology to the problems of data fragmentation, replication, and fragment allocation in distributed databases. [5] used genetic algorithms to solve data partitioning problem after modeling it as a traveling salesman problem. [2] propose a methodology for distribution design for Object DBMS, using both vertical and horizontal partitioning techniques. [18] determine vertical and horizontal fragments in distributed object-oriented and object-relational databases.

Regarding operation allocation problem, [20] conducted simulation experiments to compare four heuristic algorithms (branch-and-bound, greedy, local search, and simulated annealing) for assignment of sub-queries. The objective function is total query cost comprising of local processing and communication costs. [9] propose run-time operation allocation policies for hierarchically structured, hypercube-based multicomputer system. The site assignments are not determined a priori, instead, they are assigned during the execution of a query.

Regarding combined problem, [6] propose an integrated methodology to determine the optimal allocation of relations and query operations to sites to minimize the communication costs. [19] extended the work of [6] and used the objective function of total system cost comprising of storage, I/O, CPU, and communication. The authors did not consider response time objective function in their analysis. [14] extend the previous work of [19] by incorporating network latency time and parallel processing among the nodes.

[19] state "It is extremely important to recognize the ability of distributed systems to do parallel processing, because it is a key component in achieving fast processing" (p. 315). Furthermore, OLTP and decision support type of transactions need to be processed with different objective functions. We extend previous research by considering both the total time and response time minimizations in the objective function, so that OLTP and decision support queries can be optimized, respectively.

We include the response time cost model to the operation allocation problem by considering inter-operation parallelism and recursive cost function. We incorporate the allocation and parallel processing of subqueries for handling OLTP and Decision Support type transactions. Thus, our contribution in this research considering both types will be valuable to the industry.

2. DEVELOPMENT OF COST MODELS

Query Processing

The first step of query processing in a distributed context is to transform a high-level global query into an efficient execution strategy (the ordering of operations) on local databases. The set of execution order of subqueries and their precedence relationships can then be represented as a query tree. Each operation in the query tree is viewed as a separate subquery with one or two input relations and an output relation. An input relation is either a relation maintained by the system or the output relation of another query. The output of a subquery is an intermediate relation, which is stored at the site it is referenced and deleted after the query is answered. We consider the relational algebra operators: projection, selection and join. Other operations can be included without altering the operation allocation algorithm proposed in this research. Also note that we assume that the structure of the query, i.e., the query execution order, is fixed prior to operation allocation. Our assumption is consistent with those of previous researchers [28] in distributed databases. [28], in the design of Mermaid system, assumed some adhoc execution orders for designing optimization algorithms for distributed query processing.

There is a site set associated with each node in the query tree. The members of the site set for a leaf node are those sites that hold a copy of that relation. The site set for an operation node contains those sites that can perform the operation. In general, selection and projection operations requiring relations should be executed at only those sites that hold a copy of relations referenced so that there is no transmission of a relation required at the site of the operations, but join operations can be executed at any site. In the query tree, cost is associated with the operation nodes representing local processing times, including estimated CPU time and I/O time for its execution. The communication cost is associated with transmitting the output relation from the site of source node to the site of the receiving node.

Cost Models

The total time is the sum of all cost (time) components, while the response time is the elapsed time from the initiation to the completion of the query, including time to transmit the results back to

the site where the query has originated. We assume pre-compiled queries in our cost computations.

Let T, I, and K be the set of all sites, relations, and queries, respectively. A query transaction k can be decomposed into j subqueries (operations). Following is the list of variables.

(1) Y^k_{jt} (specifying the site at which each subquery is executed) is 1 if subquery j of query k is done at site t, otherwise it is 0. We also introduce $Y^k_{jp[m]t}$ where p[m] represents two previous operations for join operation j, and m is 1 for the left previous operation and 2 for the right previous operation in the query tree. So $Y^k_{jp[m]t}$ is 1 if the left (m = 1) or right (m = 2) previous operation for join operation j of query k is done at site t, otherwise it is 0.

(2) X_{it} (representing the data allocation) is 1 if relation i is stored at the site t. otherwise it is 0. Z^k_{ij} is 1 if input (or intermediate) relation(s) i is referenced by subquery j of query k. We also introduce $Z^k_{ijp[m]}$ where p[m] represents two previous operations for join operation j; $Z^k_{ijp[m]}$ is 1 if input (or intermediate) relation i is referenced by the left (m = 1) or right (m = 2) previous operation for join operation j of query k, otherwise it is 0.

The operation allocation problem can be expressed as follows:

Find: Y^k_{jt} (operation allocation; site t for subquery j of query k)

Given: X_{it} (data allocation; relation i stored at site t) and Z^k_{ij} (relation (or intermediate result) i referenced by subquery j of query k)

Total Time Model

The total time for each query is the sum of local processing times and communication times for all subqueries. Total Time = $\sum_j (LP^k_j + COM^k_j)$, where LP^k_j represents the local processing time of the subquery j (a node in the query tree) of a query k. COM^k_j represents the communication time of transmitting the input relation(s) to the site at which the subquery j of a query k is being executed.

Local processing time (LP^k_j)

The local processing time of a subquery depends on an operation type, the size of the input relation(s), the CPU speed and the I/O speed of the site selected. We assume that CPU processing is proportional to the amount of data accessed and that I/O time is proportional to the number of blocks read or written.

(A) For a selection or projection operation on a relation, the local processing time for the subquery j of the query k is defined as:

$$LP^k_j = \sum_t Y^k_{jt} (IO_t \sum_i Z^k_{ij} B^k_{ij} + CPU_t \sum_i Z^k_{ij} B^k_{ij}) \quad (1)$$

Where

B^k_{ij} is the number of blocks of relation i accessed by subquery j of query k,

IO_t is the I/O time of site t in msec for transferring 4k byte page into main memory,

CPU_t is the CPU time of site t in msec per 4k byte page for selection and/or projection..

(B) We also assume that the intermediate result of each unary or join operation is transmitted directly to the next join site and stored at the next join site before the execution of the next join operation. As such, the local processing time for the join j of the query k is defined as:

$$LP^k_j = \sum_t Y^k_{jt} IO_t \sum_m \sum_i \rho_m Z^k_{ijp[m]} B^k_{ijp[m]} + \quad (2a)$$

$$\sum_t Y^k_{jt} (IO_t \prod_i Z^k_{ij} B^k_{ij} + CPU_t \prod_i Z^k_{ij} B^k_{ij}) \quad (2b)$$

where ρ_m represents the selectivity of the two previous operations (m = 1 or 2), where the selectivity is the ratio of output relation size and input relation size, and $B^k_{ijp[m]}$ is the size of an input (intermediate) relation where p[m] represents two previous operations of the join operation j (m is 1 for the left and 2 for the right operation).

Note that ρ_m can represent selection, projection or join selectivity. (2a) represents the I/O time to store the intermediate results of the previous operations to the site of the current join operation. (2b) represents the I/O and CPU processing times for the current join operation. Note that we convert $B^k_{ijp[m]}$ (the size of intermediate results being stored at the join site) to B^k_{ij} (the size of same intermediate results being retrieved for the current join operation) for notational convenience so that B^k_{ij} will be used for the next join operation with the join selectivity of the current join operation.

Communication time (COM^k_j)

When either of the relation(s) to be joined is not produced at the site at which the join operation is performed, communication for join operations is needed, and is expressed as follows:

$$COM^k_j = \sum_m \sum_t \sum_p Y^k_{jp[m]t} Y^k_{jp} C_{tp} (\sum_i Z^k_{ijp[m]} B^k_{ijp[m]})$$

where C_{tp} is the communication cost between site p and site t in msec per 4k byte page.

Note that if a previous operation and the join operation are executed at the same site (t=p), then C_{tp} =0. Communication for sending the final result is also needed if the final operation is not performed at

the query originating site. Since there is only one previous operation for the final operation, we assume that $Z^k_{ijp[2]}$ for all i is 0 (also $B^k_{ijp[2]} = 0$). It should be noted that we consider communication cost to include data transmission cost. However, in real world, communication cost may also include time to synchronize the two CPUs -- we ignore this synchronization time, since this is usually a fixed overhead cost and it is not variable like data transfer cost.

Response Time Model

In a distributed database system, it is possible to decompose a query into subqueries, which can be processed in parallel and also their intermediate relations can be transmitted to the required site in parallel. Two types of parallel execution are possible: (1) intra-operation parallelism, and (2) inter-operation parallelism. A typical example of intra-operation parallelism is pipelining of a single join operation, by which two sites work in parallel; that is, the site that requests remote data will begin its join processing as soon as the first tuple or packet of data has arrived, whereas in sequential processing, the site receiving data will not begin its join processing until all of the required data has arrived. With inter-operation parallelism, several subqueries in a single query can be executed in parallel. In calculating response time, however, we limit the possible parallelism to the only immediate child nodes of join operation and not among the child nodes of different join operations.

Response time is calculated by taking into consideration the possibility of performing local processing and data transmission in parallel under the condition that the operations are performed at different sites as mentioned in the previous section. The response time of query k is: $RT^k_j = COM^k_j(p[1]) + LP^k_j(p[1]) + RT^k_j(p[1])$

where $RT^k_j(p[1])$ is the recursive function for the response time.

The first term $COM^k_j(p[1])$ is the communication time sending the results to the query originating site ($Z^k_{ijp[2]}$ for all i is 0 and $B^k_{ijp[2]} = 0$) and the $LP^k_j(p[1])$ refers to the local processing time of the final operation. For the recursive function $RT^k_j(p[1])$ (but we will use RT^k_j for convenience), we calculate the cost as follows. Four scenarios exist depending upon sites at which the join operation j and the two preceding operations p[1] and p[2] are executed. Figure 1 shows the four scenarios with three sites for operation allocation; in each scenario, the bottom two sites are used for preceding operations and the top site is used for join operation.

Scenario – 1:

The join operation j and the two preceding operators p[1] and p[2] are executed at the same site; that is, $Y^k_{jp[1]t} Y^k_{jp[2]t} C_{tp} = 0$, $Y^k_{jp[1]t} Y^k_{jt} C_{tp} = 0$ and $Y^k_{jt} Y^k_{jp[2]t} C_{tp} = 0$ then RT^k_j can be calculated by using the equation. $LP^k_j + \sum_m LP^k_j(p[m] + RT^k_j(p[m]))$. Here, LP^k_j is the local processing time for sub query j, $LP^k_j(p[m])$ is the local processing time for the preceding left (m=1) or right (m=2) operation (i.e. subsub query). These local processing times are calculated using the equations introduced in the previous section. $RT^k_j(p[m])$ is the (response) time when a preceding operator is available for local processing.

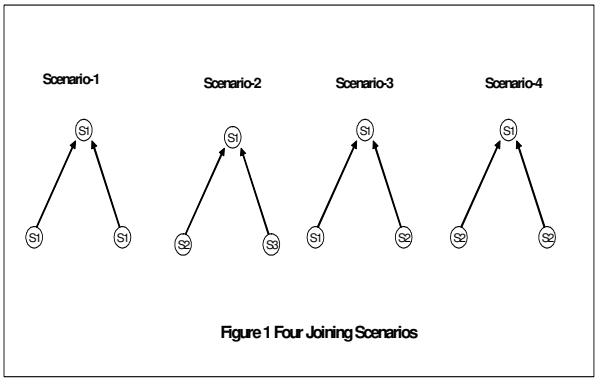

Figure 1 Four Joining Scenarios

Scenario –2:

The join operation j and the two preceding operators p[1] and p[2] are performed at three different sites. In this case the three operators can be run in parallel. Then the response time of the entire group is computed as the maximum of resource consumption of individual operators and the usage of all the shared resources (such as communication times) (Kossman, 2000). Then RT^k_j is given by

Max { LP^k_j, (3a)

$LP^k_j(p[1]) + RT^k_j(p[1])$, (3b)

$LP^k_j(p[2]) + RT^k_j(p[2])$, (3c)

$COM^k_j(p[1]) + COM^k_j(p[2])$ } (3d)

where

$COM^k_j(p[1]) = Y^k_{jp[1]t} Y^k_{jp} C_{tp} (\sum_i Z^k_{ijp[1]} B^k_{ijp[1]})$

$COM^k_j(p[2]) = Y^k_{jp[2]t} Y^k_{jp} C_{tp} (\sum_i Z^k_{ijp[2]} B^k_{ijp[2]})$

In the above, (3d) represents shared resource consumption, which is the communication time. (3a) is the local processing time for subquery j and (3b) and (3c) are the processing times for the two preceding operations of subquery j. The communication costs will be additive, since those

are the overheads on the receiving node, as represented by (3d).

Scenario –3:

The sites at which two preceding operations of subquery j are performed are different and the join subquery j uses one of these sites. There is no communication cost between one of the preceding operators, say p[1], and the operator j. That is, $Y^k_{jp[1]t} Y^k_{jt} C_{tt} = 0$, $Y^k_{jp[2]p} Y^k_{jt} C_{tp} \neq 0$ and $Y^k_{jp[1]t} Y^k_{jp[2]p} C_{tp} \neq 0$, then RT^k_j is given by:

$$\text{Max } \{ LP^k_j + LP^k_j(p[1]) + RT^k_j(p[1]), \quad (4a)$$

$$LP^k_j(p[2]) + RT^k_j(p[2]), \quad (4b)$$

$$COM^k_j(p[2]) \ \} \quad (4c)$$

where

$$COM^k_j(p[2]) = Y^k_{jp[2]p} Y^k_{jt} C_{tp} \left(\sum_i Z^k_{ijp[2]} B^k_{ijp[2]}\right)$$

Since sub query j and the left previous operation p[1] are executed at the same site, the local processing times of the two sites need to be added (4a). Since right previous operation p[2] is executed at a different site, its local processing time (included in (4b)) can be executed in parallel. In addition, the communication time (4c) can be implemented in parallel as well.

Scenario – 4:

In secenario-4, the two preceding operations of subquery j, p[1] and p[2], are executed at the same site, while the subquery j is executed at a different site. There is communication time involved in shipping data from both the preceding operations p[1] and p[2] to the site of subquery j. That is, $Y^k_{jp[1]p} Y^k_{jt} C_{tp} \neq 0$, $Y^k_{jp[2]p} Y^k_{jt} C_{tp} \neq 0$ and $Y^k_{jp[1]p} Y^k_{jp[2]p} C_{pp} = 0$. Also, there will be no parallelism between the operations p[1] and p[2]. Then RT^k_j is given by

$$\text{Max } \{ LP^k_j, \quad (5a)$$

$$LP^k_j(p[1]) + LP^k_j(p[2]) + RT^k_j(p[1]) + RT^k_j(p[2]), \quad (5b)$$

$$COM^k_j(p[2]) + COM^k_j(p[2]) \} \quad (5c)$$

where

$$COM^k_j(p[1]) = Y^k_{jp[1]p} Y^k_{jt} C_{tp} \left(\sum_i Z^k_{ijp[1]} B^k_{ijp[1]}\right)$$

$$COM^k_j(p[2]) = Y^k_{jp[2]p} Y^k_{jt} C_{tp} \left(\sum_i Z^k_{ijp[2]} B^k_{ijp[2]}\right)$$

In the above, since subquery j is executed at a different site than the preceding operators, its local processing of subquery j (5a) can be done in parallel to the communication time (5c) and the processing times of p[1] and p[2] . Since the preceding operators are executed at the same site, their local processing times are additive (5b). Also, the communication costs will be additive, since those are the overheads on the receiving node. Above equations hold whether previous operations are joins, selections, or projections, or other relational algebra operators.

The stopping condition of the recursive function RT is as follows. We define: if p[m] in $Z^k_{ijp[m]}$ is equal to zero in the response time recursive function, where zero for p[m] means that the previous operation for this operation j (subquery) is original relation. In scenarios 2 and 3, parallelism between the preceding operations p[1] and p[2] is implied. It is assumed there is no clash in data access between the two preceding operations, i.e. $Z^k_{ij}(p[1]) * Z^k_{ij}(p[2]) = 0 \ \forall_i$, otherwise local processing times can be additive in the worst case.

3. OPTIMIZATION WITH GA

We use the heuristic procedure based on GA to solve due to intractability of the allocation problem. GA has been used by several researchers [5] [8] [12] [14] [24] to solve computationally complex optimization problems in database design. When compared to other heuristic algorithms [16], GA provides global 'optima' with less time.

The Genetic Algorithm Procedure

The GA starts with an initial population which is usually chosen at random and contains a wide variety of members. Each member in a population represents a possible solution to the problem at hand and is commonly called a chromosome. In a typical GA [11], each solution (chromosome) is evaluated according to an evaluation (fitness) function. The population evolves from one generation to the next through the application of genetic operators: selection, crossover, and mutation. During selection operation, members of the population (parents) are selected in pairs to produce new possible solutions. The fitter a member of the population, the more likely it is to produce offspring. Crossover operator is then used to result in offspring inheriting properties from both parents. The offspring is evaluated and placed in the next population, possibly replacing weaker members of the last generation. Crossover operator is applied with a certain probability (crossover rate). Mutation operator is used to allow further variation of offspring. Mutation operator is also applied with a certain probability (mutation rate).

We use integers as the genetic representation of a solution of operation allocation. The length of chromosome is equal to the number of operations in the query tree. Each integer at the particular position in chromosome represents the site selected for a particular operation. The initial population is

generated by using a random number between one and the number of sites from the uniform distribution. The fitness of each individual member in the population is the query execution cost calculated using the equations presented in section 2. For the selection process, we adopt a technique termed "stochastic remainder without replacement" [11]. We also incorporate "elitism", in which the GA keeps track of the best fitness chromosome in the population. If the best fitness chromosome is not in the new population, it is put into the new population by removing the worst fitness chromosome. The effect of elitism is that the GA always finds a better solution than the one in the previous generation unless all solutions in the new generation are worse the best one from the previous generation.

[23] identified values for population size, crossover rate (0.95) and mutation rate (0.005) that produce good GA performance. However, we found that when setting crossover rate at 0.7 and mutation rate 0.2, the genetic algorithm performed better than using the rates suggested by [11]. The population size is set at 50, and the stopping condition is when the number of iterations reached 50 or there is no more improvement in the best solution.

Performance of GA

In order to compare the results from GA with optimal, we ran two types of experiments: one keeping the cost coefficients constant and the other varying cost coefficients. In case 1, I/O, CPU, and communication cost coefficients are fixed. We assumed network to consist of 5 sites. Using a three-join query, we solved two problems, one with objective function of total time and the other with response time. We assume that each relation is allocated two sites. The solution obtained by GA matched the optimal solution obtained by exhaustive enumeration. The exhaustive enumeration has a solution space of about 2000 and it took about 2 minutes to evaluate. The run time for GA is less than half of that required for exhaustive enumeration. We solved two additional problems, using the four-join query. The size of solution space by exhaustive enumeration was about 5,000 and it took 20 minutes to solve, while GA took about 1 minute. Furthermore, the GA found the optimal solutions for both the problems.

In case 2, we varied the cost coefficients for I/O, CPU, and communication and solved four more problems, with 3-join and 4-join queries and with both the objective functions. The GA found the optimal solutions for all the problems.

In order to investigate the run-time efficiency of the operation allocation, we conducted two experiments, one by varying the number of joins from 3 to 20 using 5 database sites and the other by varying the number of sites from 3 to 12 using ten-join query. Figure 2 shows run time performance of GA varying number of joins. Exhaustive enumeration could be performed for only two cases (3-&4-joins). For 3-join case, exhaustive enumeration took 110 seconds, while GA took 10 seconds. For 4-join case, they were 1200 and 19 seconds, for exhaustive enumeration and genetic algorithm, respectively. Figure 3 shows the run time efficiency of GA with a 10-join query, varying the number of sites. With two copies each for a relation, exhaustive enumeration results in a large solution space, so we assumed one copy per relation. This results in a solution space of 59,049 for 3-site problem and 1,048,576 for 4-site problem. The run time of GA for 3-site case is 30 seconds and for exhaustive enumeration it is 2.5 hours; for a 4-site case, GA took 40 seconds and exhaustive enumeration took 43 hours. The run time of GA varied linearly with number of sites, while it was exponential for exhaustive enumeration.

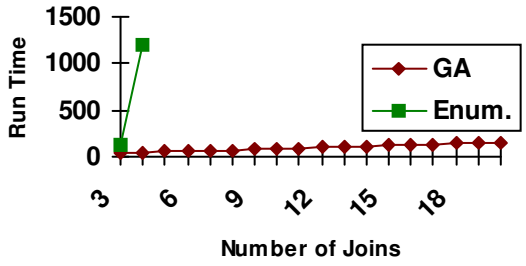

Figure 2: Execution Time (in seconds) of GA vs. Number of Joins

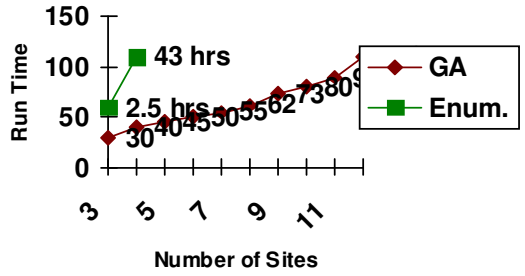

Figure 3: Execution Time (in seconds) of GA vs. Number of Sites

4. CONCLUSIONS

The purpose of this research is to solve the problem of allocating operations (subqueries) of a query to individual sites of a network, with two objective functions: total time minimization and response time minimization. Comprehensive cost models, including local processing and communication costs, considering parallelism of subqueries were developed for both objective functions based on the query trees that represent a set of operations with their precedence relationship. Our results show that the optimal allocations are quite different with the two objective functions. Response time minimization could be achieved through a large variety of parallel

execution and parallel transmission, for which subqueries were allocated to as many sites as possible. Total time minimization could be achieved when queries are executed by using a minimum number of sites. In extreme case, all subqueries could be executed at the same site if all necessary fragments reside at one site. Minimization of total system operating cost usually attempts to minimize resource consumption (CPUs, I/Os, and communication channels) -- more transactions can be processed for a given time period i.e., the system throughput is increased. On the other hand, a decrease in response time may be obtained by having a large number of parallel executions to different sites, requiring a higher resource consumption, which means that the system throughput is reduced. Furthermore, our results showed that the query execution plans with total time minimization results in higher response time compared to plans with response time minimization. Our results have shown the GA produced optimal solutions, as compared with the exhaustive enumeration for the problems that could be tested. We have also shown the efficiency of the genetic algorithm in solving complex queries, up to 20-join query tree. We believe our research provides a better understanding of the underlying query execution plans under the objectives of total time minimization and response time minimization.

6. REFERENCES

1. Apers, P.M.G. 1988. Data allocation in distributed database systems. *ACM Trans. on Database Systems,* 13 (3), 263-304.
2. Baiao, F., Mattoso, M. & Zaverucha, G. 2004. A Distribution Design Methodology for Object DBMS. *Journal of Distributed and Parallel Databases.* 16 (1), 45-90.
3. Bergsten, B., Couprie, M. & Valduriez, P. 1993. Overview of Parallel Architectures for Database. *The Computer Journal*, 36, 734-740.
4. Carey, M. J. & Livny, M. 1991. Conflict Detection Tradeoffs for Replicated Data. *ACM Transactions on Database Systems.* 16, 703-746.
5. Cheng, C-H, Lee, W-K, & Wong, K-F. 2002. A Genetic Algorithm-Based Clustering Approach for Database Partitioning. IEEE *Transactions on Systems, Man, and Cybernetics,* 32(3), 215-230.
6. Cornell, D.W. & Yu, P.S. 1989. On optimal site assignment for relations in the distributed database environment. *IEEE Transactions on Software Engineering*, 15 (8), 1004-1009.
7. Davis, L. 1991. *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, N.Y.
8. Du, J, Alhajj, R, & Barker, K. 2006. Genetic algorithms based approach to database vertical partitioning. *Journal of Intelligent Information Systems*, 26 (2), 167-183
9. Frieder, O. & Baru, C. 1994. Site and Query Scheduling Policies in Multicomputer Database Systems. *IEEE Transactions on Knowledge and Data Engineering*, 6(4), 609-619.
10. Gog, A. & Grebla, H-A. 2005. Evolutionary Tuning for Distributed Database Performance. *The 4th International Symposium on Parallel and Distributed Computing*, 275-281.
11. Goldberg, D.E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing.
12. Gorla, N. 2001. An Object-Oriented Database Design for Improved Performance. *Data and Knowledge Engineering,* 37, 117-138.
13. Graefe, G. 1993. Query Evaluation Techniques for Large Databases. *ACM Comp. Surveys,* 25, 73-90.
14. Johansson, J M, March, S T, & Naumann, J D. 2003. Modeling Network Latency and Parallel Processing in Distributed Database Design. *Decision Sciences*, 34 (4), 677-706.
15. Kossmann, D. 2000. The State of the Art in Distributed Query Processing. *ACM Computing Surveys.* 32(4), 422-469.
16. Li, B. & Jiang, W. 2000. A novel stochastic optimization algorithm. *IEEE Trans. on Systems, Man, and Cybernetics,* Part B, 30(1).
17. Lim, S-J & Ng, Y-K. 1997. Vertical Fragmentation and Allocation in Distributed Deductive Database Systems. *Information Systems*, 22(1), 1-24.
18. Ma, H. & Kirchberg, M. 2008, Cost based fragmentation for distributed complex value databases. *Lecture Notes in Comp. Sci.*, 4801, 72-86
19. March, S.T. & Rho, S. 1995. Allocating Data and Operations to Nodes in Distributed Database Design. *IEEE Trans. on Knowledge and Data Engg*, 7(2).
20. Martin, T., Lam K. & Russel, J. 1990. An Evaluation of Site Selection Algorithms for Distributed Query Processing. *The Computer Journal*, 33(1), 61-70.
21. Sacco, G. & Yao, S.B. 1982. Query Optimization in Distributed Data Base Systems. *Advances in Computer*, 21, 225-273.
22. Song, S-K & Gorla, N. 2000. A Genetic Algorithm for Vertical Fragmentation and Access Path Selection. *The Computer Journal*, 43(1), 81-93.
23. Schaffer, J. D., Caruana, R. A., Eshlman, L. J., & Das, R. 1989. A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization, In J. D. Schaffer, (ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, 51-60
24. Tam, K.Y. 1992. Genetic algorithms, function optimization, and facility layout design. *European Journal of Operations Research*, 63(2).
25. Tamhankar, A.M. & Ram, S. 1998. Database Fragmentation and Allocation: An Integrated Methodology and Case Study. *IEEE Trans. on Systems, Man, and Cybernetics*, 28(3), 288-305.
26. Kulkarni, U R, & Jain, H K. 1993. Interaction Between Concurrent Transactions in the Design of Distributed Databases. *Decision Sciences*, 24(2), 253-277
27. Yu, C.T. & Chang, C.C. 1994. Distributed Query Processing. *ACM Computing Surveys*, 16, 399-433.
28. Yu, C. T., Chang, C., Templeton, M., Brin, D. & Lund, E. 1985. Query Processing in a Fragmented Relational Distributed System: Mermaid. *IEEE Transactions on Software Engineering*, 11, 795-809.

Received: June 2008. Accepted: Nov. 2009