DOI: 10.3745/KIPSTD.2009.16-D.3.295

분산 데이타베이스에서의 질의실행시간 최소화를 위한 유전자알고리즘: 총 시간 대 반응시간

송 석 규†

요 약

질의실행시간최소화는 분산 데이타베이스 설계에 있어 가장 중요한 목적중의 하나이다. 총시간최소화는 온라인거래처리시스템의 목적인 반면, 반응시간최소화는 의사결정지원 질의시스템의 목적이다. 본 논문에서는 질의실행시간최소화를 달성하기 위해 질의를 세분화하여 최적의 데이터베이스 사이트에 할당하는 분석모델을 개발하였으며, 문제해결방법으로 유전자알고리즘을 채택하였다. 총시간최소화 관점에서의 질의실행계획은 반응시간최소화 관점의 질의실행계획에는 적합하지 않다는 것을 증명하였으며, 그 반대의 경우도 증명하였다. 최대 20개의 조인이 포함되는 질의를 설계하여 시뮬레이션 실험을 통해 테스트를 수행하였고, 유전자알고리즘과 완전한 전수조사와의 결과를 비교함으로써 모든 경우에 유전자알고리즘을 채택한 해결책이 최적의 결과를 도출하였음을 증명하였다.

키워드 : 분산 데이타베이스, 질의최적화, 하부질의 할당, 질의실행계획, 유전자알고리즘

A Genetic Algorithm for Minimizing Query Processing Time in Distributed Database Design: Total Time Versus Response Time

Song, Sukkyu†

ABSTRACT

Query execution time minimization is an important objective in distributed database design. While total time minimization is an objective for On Line Transaction Processing (OLTP), response time minimization is for Decision Support queries. We formulate the sub-query allocation problem using analytical models and solve with genetic algorithm (GA). We show that query execution plans with total time minimization objective are inefficient from response time perspective and vice versa. The procedure is tested with simulation experiments for queries of up to 20 joins. Comparison with exhaustive enumeration indicates that GA produced optimal solutions in all cases in much less time.

Keywords : Distributed Databases, Query Optimization, Sub-Query Allocation, Query Execution Plans, Genetic Algorithms

1. Introduction

Distributed database systems have become very important and common in today's geographically distributed organizations. Two important aspects of distributed database design are data allocation and query optimization. It is vital to the system performance that the query processing be designed to produce the most efficient execution plans of users' queries. Distributed database design and query optimization are still an active research area since several researchers have been contributing to the area [Kossmann, 2000; Cheng et al, 2002; Johansson et al, 2003; Arcangeli et al, 2004; Syam, 2005; Gu et al, 2006; Seshadri and Cooper, 2007]. The design of distributed query processing can be divided into two aspects: query execution order and query execution plan or operation allocation [Kossmann, 2000]. Query execution order indicates the order in which the subqueries should be executed. Operation allocation indicates the site at which an operation (subquery) should be executed.

Queries can be classified into OLTP (On-Line Transaction Processing) and decision-support types of queries [Bergsten et al., 1993; Ziane et al., 1993]. "To inquire about air line seat availability" is an example of OLTP

† 정 회 원 : 영산대학교 호텔경영학과 교수
논문접수 : 2008년 11월 19일
수 정 일 : 1차 2009년 3월 3일, 2차 2009년 4월 28일
심사완료 : 2009년 5월 12일

type query in the travel industry, which is a highly repetitive transaction that requires high throughput. Typically, the decision support queries are more complex, less frequent, and access a large volume of data in an ad hoc manner. An inquiry 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, OLTP and decision support have different goals and thus query execution plans for them need to be designed with total time minimization and response time minimization objective functions, respectively. The distributed database design with total time minimization that is optimal for OLTP transactions is inefficient for decision support transactions and vice versa. 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. We are the first to undertake the design optimization in distributed databases to minimize the execution time of these two transaction types. The objective of the paper is to present a methodology for distributed database design in which 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. The outcomes of our approach are selection of execution sites for operations of a given distributed query plan and comparison of allocations under the two objective functions.

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 [Martin et al., 1990; Caudrado, 1995; Florin and Alin, 2008]. If a query references n relations, and each relation R_i has X_i copies, $1 \leq i \leq n$, then a straightforward enumeration algorithm for selecting one copy of each relation takes time $O(\prod_{i=1}^{n} X_i)$ [Martin et al., 1990]. The problem of finding the allocation that yields the minimum cost is NP-hard [Kossmann, 2000]. In order to deal with this hard problem, we use Genetic Algorithms [Goldberg, 1989] to arrive at near optimal solution. Genetic algorithm is a heuristic solution that has been used to solve intractable problems in database design [Song and Gorla, 2000; Cheng et al, 2002; Du et al, 2006].

The organization of the paper is as follows. Next, we present prior research in distributed database design and query processing, highlighting the research gaps. Section 2 has discussion of cost models, including query processing model and analytical cost models for total time and response time. Section 3 has research methodology using genetic algorithm; section 4 has illustration of our procedure for both objective functions: total time and response time minimizations along with time performance analysis. Section 5 has conclusions.

1.1 Previous Research

In order to improve distributed database performance, previous researchers conducted studies in both data allocation (how to allocate data fragments to sites) and operation allocation (how to allocate subqueries to sites), though operation allocation did not receive as much attention. Apers [1988] 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. They compared both optimal and heuristic solutions using static and dynamic processing schedules. The focus of their study was fragment allocation, but not operation allocation.

Martin et.al. [1990] 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. They did not address subquery parallelism or response time objective functions. Frieder and Baru [1994] 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. Tamhankar and Ram [1998] proposed an integrated methodology to the problems of data fragmentation, replication, and fragment allocation in distributed databases. Cheng et al [2002] used genetic algorithms to solve data partitioning problem after modeling it as a traveling salesman problem. Baiao et al [2004] propose a methodology for distribution design for

Object DBMS, using both vertical and horizontal partitioning techniques.

Kossmann [2000] provides a survey of techniques useful for query processing in distributed databases. Cornell and Yu [1989] propose an integrated methodology to determine the optimal allocation of relations and query operations to sites to minimize the communication costs. They propose two-step approach. First, they decompose queries into individual relational algebra operations and use them to determine the optimal allocation of relations and operations to sites. It was formulated as a linear integer-programming problem with the objective function of minimizing total communication costs. They provide response time analysis based on queuing network model. However, their objective function includes communication costs only [March and Rho 1995] and their response time calculation does not consider the parallelism among the query executions of different sites.

March and Rho [1995] extended the work of Cornell and Yu [1989] and used the objective function of total system cost comprising of storage, I/O, CPU, and communication. After decomposing relations and transactions, they apply iterative genetic algorithm to determine the 'optimal' allocation of fragments and operations. The authors did not consider response time objective function in their analysis and they suggest consideration of inter-operation parallelism as a potential research direction. Kulkarni and Jain [1993] study the interaction effect of concurrent transactions in distributed databases using simulation. Johnsson, March, and Naumann [2003] extend the previous work of March and Rho [1995] by incorporating network latency time and parallel processing among the nodes.

There have been fewer studies that contributed to the design of query execution plans in distributed databases. March and Rho [1995] 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 require 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. Thus, our research makes a contribution in the design of query execution plans in the distributed databases.

We include the response time cost model to the operation allocation problem by considering inter-operation parallelism and recursive cost function. We consider explicitly query tree and query execution order in the models. Our simulation experiments using genetic algorithm are run in replicated distributed database environment with complex queries. Since any database environment consists of both OLTP and Decision Support type of transactions, our contribution in this research considering both types will be valuable to the industry. We incorporate the allocation and parallel processing of subqueries for handling OLTP and Decision Support type transactions.

2. Development of cost models

2.1 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 [Ozsu and Valduriez, 1991]. 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 researches in distributed databases as such some ad hoc execution orders for designing optimization algorithms for distributed query processing are assumed [Kossmann, 2000].

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 processing 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.

2.2 Cost Models

As stated earlier, we investigate the subquery allocation problem using two types of objective functions: (1) total time minimization and (2) response time minimization. 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. Furthermore, 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} specifies the site at which each subquery is executed. Y^k_{jt} 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:

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)

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

2.3 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 represent 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.

2.3.1 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 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.

2.3.2 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:

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

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.

2.4 Response Time Model

In a distributed database system, it is possible to decompose a query into subqueries that can be processed in parallel and also their intermediate relations can be transmitted in parallel to the required site. Two types of parallel execution are possible: (1) intra-operation parallelism, and (2) inter-operation parallelism [Srivastava and Elsesser, 1993]. 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 request 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:

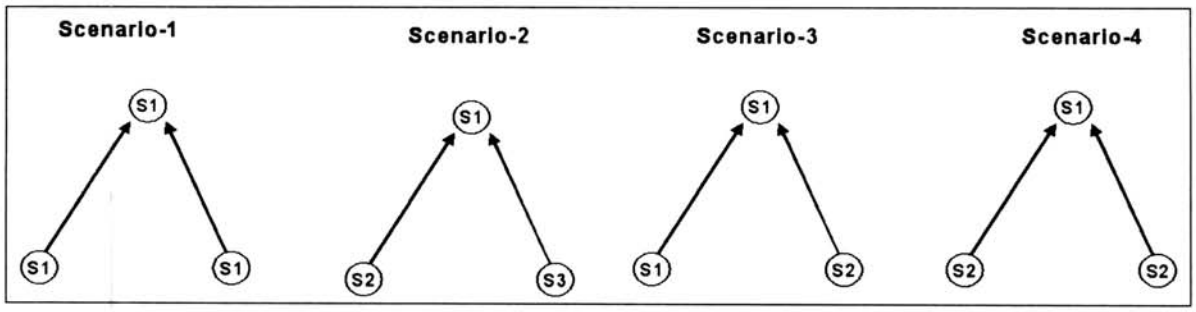
Response time RT^k_j = $COM^k_j(p[1])$ + $LP^k_j(p[1])$ + $RT^k_j(p[1])$
where $RT^k_j(p[1])$ is the recursive function for the response time.

The first term $COM^k_j(p[1])$ is to calculate 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 denote are used for preceding operations and the top site is used for join operation.

2.4.1 Scenario - 1:

The join operation j and the sites 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.

$$LP^k_j + \sum_m LP^k_j(p[m] + RT^k_j(p[m]))$$



(Fig. 1) Four Joining Scenarios

Here, LP_j^k is the local processing time for sub query j, $LP_j^k(p[m])$ 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. $RT_j^k(p[m])$ is the (response) time when a preceding operator is available for local processing.

2.4.2 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_j^k is given by

$$\begin{aligned}
\text{Max } \{ & \ LP_j^k, && \text{(3a)}\\
& LP_j^k(p[1]) + RT_j^k(p[1]), && \text{(3b)}\\
& LP_j^k(p[2]) + RT_j^k(p[2]), && \text{(3c)}\\
& COM_j^k(p[1]) + COM_j^k(p[2])\} && \text{(3d)}\\
\text{where } & COM_j^k(p[1]) = Y^k_{jp[1]i} Y^k_{jp} C_{tp} (\textstyle\sum_i Z^k_{ijp[1]} B^k_{ijp[1]})\\
& COM_j^k(p[2]) = Y^k_{jp[2]i} Y^k_{jp} C_{tp} (\textstyle\sum_i Z^k_{ijp[2]} B^k_{ijp[2]})
\end{aligned}$$

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).

2.4.3 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, $Y^k_{jp[1]i} Y^k_{ji} C_{ii} = 0$, $Y^k_{jp[2]p} Y^k_{ji} C_{tp} \neq 0$ and $Y^k_{jp[1]i} Y^k_{jp[2]p} C_{tp} \neq 0$, then RT_j^k is given by:

$$\begin{aligned}
\text{Max } \{ & \ LP_j^k + LP_j^k(p[1]) + RT_j^k(p[1]), && \text{(4a)}\\
& LP_j^k(p[2]) + RT_j^k(p[2]), && \text{(4b)}\\
& COM_j^k(p[2]) \ \} && \text{(4c)}\\
\text{where } & COM_j^k(p[2]) = Y^k_{jp[2]p} Y^k_{ji} C_{tp} (\textstyle\sum_i Z^k_{ijp[2]} B^k_{ijp[2]})
\end{aligned}$$

In the above 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.

2.4.4 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, $Y^k_{jp(1)p} Y^k_{ji} C_{tp} \neq 0$, $Y^k_{jp(2)p} Y^k_{ji} C_{tp} \neq 0$ 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_j^k is given by

$$\begin{aligned}
\text{Max } \{ & \ LP_j^k, && \text{(5a)}\\
& LP_j^k(p[1]) + LP_j^k(p[2]) + RT_j^k(p[1]) + RT_j^k(p[2]), && \text{(5b)}\\
& COM_j^k(p[2]) + COM_j^k(p[2])\} && \text{(5c)}\\
\text{where } & COM_j^k(p[1]) = Y^k_{jp(1)p} Y^k_{ji} C_{tp} (\textstyle\sum_i Z^k_{ijp[1]} B^k_{ijp[1]})\\
& COM_j^k(p[2]) = Y^k_{jp[2]p} Y^k_{ji} C_{tp} (\textstyle\sum_i Z^k_{ijp[2]} B^k_{ijp[2]})
\end{aligned}$$

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 (4b). 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.

2.5 Cost Coefficients

We use the relative cost coefficients associated with three cost components instead of using the actual measures since the purpose of this research is to evaluate the relative performance of operation allocation schemes in a

given distributed database environment. We assume that the communication network is relatively high-speed wide area networks whose data transmission speed is almost equivalent to local area networks [Atkins and Norris, 1995]. A typical ratio of communication cost to local I/O cost is 1:1.6 [Kossmann, 2000]. Furthermore, the typical average I/O speed for processing one page block is 20 msec and the CPU speed 1-5 msec [Kossmann, 2000]. So throughout this research we use: I/O cost coefficient (IO_i) is 20 per page, CPU cost coefficient (CPU_i) 1 per page, and communication cost coefficient (C_{ip}) 12 per page, unless otherwise mentioned.

3. Methodology

As stated earlier, we will develop our solution procedure using genetic algorithm due to intractability of the distributed database design problems. Genetic algorithms have been used by other researchers [Kumar and Pathak, 1995; Cheng et al, 2002; Gorla, 2001; Johansson et al, 2003; Du et al, 2006] to solve difficult optimization problems in database design. When compared to other heuristic algorithms [Li and Jiang, 2000], Genetic Algorithm (GA) provides global 'optima' with less time. Furthermore, the distributed database design problems addressed in this research can be classified into combinatorial optimization problems [Kossmann, 2000]. Most combinatorial optimization problems are NP-hard, and so enumeration algorithms are inefficient to solve large scale NP-hard problems. Thus heuristics such as genetic algorithms, which can obtain nearly optimal solutions within a reasonable time, are proposed as alternative solution approaches [Goldberg, 1989; Michalewicz and Fogel, 2004].

Any genetic algorithm must have the following five components [Goldberg, 1989]: (1) A genetic representation of a solution to the problem, (2) A way to create an initial population of solutions, (3) An evaluation function that evaluates solutions, (4) Genetic operators that affect the population of offspring, and (5) Values for the parameters that the genetic algorithm uses (population size, probabilities of applying genetic operators).

Each solution (chromosome) in the GA is a string of integers, where the string length represents number of operations and each integer at a particular position in the string represents the site number selected for the operation in that position. The fitness of each individual member in the population is the query execution cost calculated according to the equations presented in the previous section. To select a member, we adopt a technique termed "stochastic remainder without replacement" [Goldberg, 1989]. Its basic idea is that chromosomes with higher-than-average fitness generate more than one offspring at the next generation, and it works as follows: 1) The fitness is normalized with the average value of the fitness. The normalized fitness of a chromosome is equal to the fitness of that chromosome divided by the average value of the fitness of all chromosomes in the population. 2) Chromosomes with higher-than-average fitness will have more than one offspring, and those with below-average fitness will have less than one offspring on the average. 3) After the number of offspring has been determined as above, the remainder of the new population is then filled up by choosing another offspring for each of the remaining chromosomes with probability equal to the fractional part of the normalized fitness until the total number of offspring equals the population size.

The parameters for crossover rate and mutation rate were adapted primarily based on a large empirical study by [Schaffer et al., 1989]. We also incorporate "elitism" [Davis, 1991], in which the GA keeps track of the best fitness chromosome in the population. More description about GA can be found from [Goldberg, 1989]. A general sketch of the Genetic Algorithm based on the procedures/ parameters outlined in (Fig. 2).

```
t is the iteration number
P(t) is the solution pool at the iteration t
p(t) is a candidate solution

t ← 0
Generate initial solution pool, P(t);
Calculate the fitness of each solution E(t) based on total time or response time;
While t<50 or there is improvement in consecutive generations do
        Select p(t) from P(t) using 'stochastic reminder without replacement'
        Produce off springs by applying crossover and/or mutation with probability
        Replace the worst fitness chromosomes by the best fitness chromosomes;
        t ← t+1;
end;
```

(Fig. 2) General Sketch of the Genetic Algorithm

4. Illustration

Consider a replicated distributed database with 4 sites and 3 relations from a university database: Students (S#, Sname, Major), Courses (C#, Cname, Dept, Credits), and Enrolls (S#, C#, Grade), the database statistics of which is provided in Table 1A. It is also assumed that the size of page block is 4k bytes, and the length of attributes measured in bytes are: S# (15), Sname (20), Major (10), C# (8), Cname (20), Dept (30), Credits (2), Grade (2). The allocation of the relations to sites is as follows: Relation (F1) is stored in sites 1 and 2, relation (F2) at sites 2 and 3, relation F3 is stored at sites 3 and 4. As stated earlier, the average cost coefficients are assumed to be in the ratio of 20:1:12 for I/O, CPU, and Communication, respectively and actual cost coefficients are shown in Table 1B.

The following SQL statement will be used and the corresponding query tree is shown in Figure 3. It is assumed that the query-originating site is 4, and it is the node 6 in (Fig. 3).

```
SELECT STUDENTS.S#, STUDENTS.Sname, COURSES.
Cname
        FROM    STUDENTS, COURSES, ENROLLS
        WHERE   STUDENTS.Major = 'CIS'
        AND     ENROLLS.Grade  > 'C'
        AND     STUDENTS.S# = ENROLLS.S#
        AND     ENROLLS.C# = COURSES.C#
```

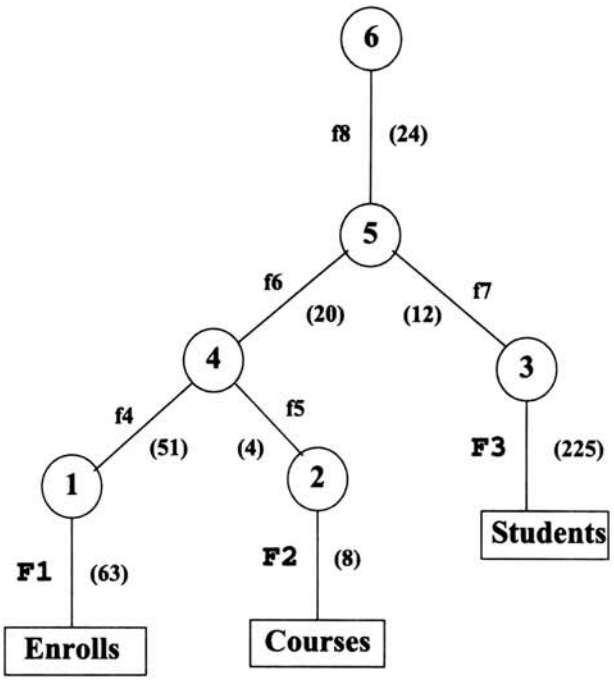
By using simple estimation techniques, the results of each operation execution are as follows: (Note that ρ_s, ρ_p and ρ_j: the selectivity for selection, projection and join respectively)

〈Table 1A〉 Database Statistics

Relation	Cardinality	Tuple Size	Relation Size (bytes)	Relation Size (blocks)
Students (F3)	20 000	45	900 000	225
Enrolls (F1)	10 000	25	250 000	63
Courses (F2)	500	60	30 000	8

〈Table 1B〉 Communication, I/O, and CPU Cost Coefficients

		Site			
		1	2	3	4
Communication	1	0	13	12	11
Coefficients	2	13	0	11	12
	3	12	11	0	13
	4	11	12	13	0
I/O Coefficients		20	19	18	21
CPU Coefficients		1	1	1	1



(Fig. 3) Example Query Tree

operation 1: $\sigma_{\text{grade} > \text{'C'}}$ (Enrolls) $\Rightarrow$ f4
(63 x 0.8 (ρ_s) = 51 blocks)

operation 2: $\Pi_{\text{C\#, Cname}}$(Courses) $\Rightarrow$ f5
(8 x 0.47 (ρ_p) = 4 blocks)

operation 3: $\sigma_{\text{major='CIS'}}$ (Students) $\Rightarrow$ f7
(225 x 0.05 (ρ_s) = 12 blocks)

operation 4: f4 $\triangleright\triangleleft_{\text{c\#=c\#}}$ f5 $\Rightarrow$ f6
(f4 x f5 x ρ_j = 51 x 4 x 0.1 = 20 blocks)

operation 5: f6 $\triangleright\triangleleft_{\text{S\# = S\#}}$ f7 $\Rightarrow$ f
(f6 x f7 x ρ_j = 20 x 12 x 0.1 = 24 blocks)

The results of the above size estimation are also shown in (Fig. 3).

4.1 Total Time Minimization

The fitness function used in the GA is based on the total time. The optimal solution was obtained in the third generation. The final solution, based on the total time minimization objective function, is 22323 (operations 1,2, and 4 are assigned to site 2, and operations 3 and 5 are assigned to site 3). The cost calculations are shown in <Table 2>. We can see that the total time to execute the query is 16488 time units, which comprises 15216 of I/O, 740 of CPU, and 532 of communication time.

〈Table 2〉 I/O, CPU and Communication Times for Total Time Minimization Problem

Subquery	Assigned Site	I/O Time	CPU Time	Local Processing Time	Comm. Time 1	2
1	2	1197	63	1260	-	-
2	2	152	8	160	-	-
3	3	4050	225	4275	-	-
4	2	4921	204	5125	-	-
5	3	4896	240	5136	220	-
* 6	4	0	0	0	312	-
Total		15 216	740	15956	532	-

* 6 is the query originating site

4.2 Response Time Minimization

The same example is used for the purpose of finding optimal operation allocation with the objective function of minimizing response time. The optimal solution from running the genetic algorithm is 23423 (i.e. operation 1 is assigned to site 2, operations 2, 4, and 5 are assigned to site 3, and operation 3 is assigned to site 4). The I/O, CPU, and Communication costs are given in <Table 3>. For the given optimal operation allocation, the response time is 7,708 units.

4.3 Analysis of Solutions

In the above example, as an illustration we used two copies for each of the basic relations: Enrolls at sites 1 and 2, Courses at sites 2 and 3, and Students at 3 and 4. In the total time minimization objective, both operation 1 (σ(Enrolls)) and operation 2 (Π(Courses)) were assigned to site 2. In the response time minimization, operations 1 and 2 are assigned to different sites, 2 and 3 respectively - this resulted in communication time. Similarly the algorithm assigned operation 3 (σ(Students)) to site 3, the same site used for operation 5 ($f6 \triangleright\triangleleft_{S\#=S\#} f7$) in total time minimization case. On the other hand, in response time minimization case, operation 3 and operation 5 were assigned to different sites (sites 4 and 3 respectively) in order to allow parallelism. In this analysis, the time unit is msec based on processing one page block as stated in section 2.5.

With the total time minimization objective function, the total resource consumption is 16,488 time units. On the other hand, with response time minimization, the total resource consumption is 17,355 units, which represents an increase of more than 5% over the execution plan with total time minimization. Furthermore, with total time minimization, the total time that is needed to execute the query is 16,488 units. With the same execution plan, the response time is 11,143. With the response time objective function, the given execution plan results in a response time of 7,708 time units. This represents a reduction of 31% in response time compared to the solution with total time minimization objective. This implies that using total time minimization objective function will be very inefficient with respect to response time. Thus, the query execution plans should be designed with the appropriate objective function into consideration.

In the example, we allowed two copies of each relation. In effect, with total time minimization the five operations were allocated to two sites (sites 2 and 3) whereas in response time minimization case they were assigned to three sites (sites 2,3, and 4). In a fully replicated database, the "total time minimization allocation" will result in the operations being allocated to only one site (the one with least I/O and CPU cost coefficient); in this case total time and response time will be the same, since there is no parallelism possible. On the other hand, the "response time minimization allocation" will result in many sites being used for the operations; this will result in lot higher total time compared to the total time in the case of "total time minimization allocation", because of increased communication costs.

〈Table 3〉 I/O, CPU, and Communication Times for Response Time Minimization Problem

Subquery	Assigned Site	I/O Time	CPU Time	Local Total	Comm. Time 1	2
1	2	1197	63	1260	-	-
2	3	152	8	160	-	-
3	4	4050	225	4275	-	-
4	2	4921	204	5125	0	44
5	3	4896	240	5136	220	156
* 6	4	0	0	0	312	-

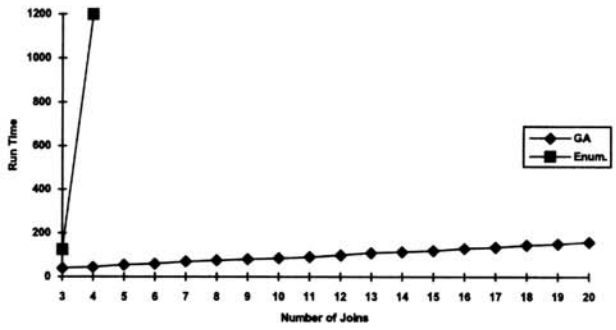
* 6 is the query originating site

In summary, total time minimization is achieved when queries use as few sites as possible, which will result in minimum data communication costs. In the extreme case, all subqueries can be executed at the same site. Response time minimization can be achieved through a large number of parallel executions and parallel transmissions when subqueries are assigned to as many sites as possible. Thus, there is performance difference with the execution plans under these two conflicting objective functions.

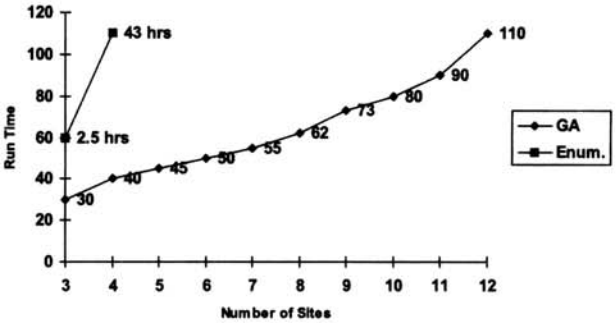
4.4 Time Performance of Algorithm

In order to compare the results from GA with optimal (through exhaustive enumeration), 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 at 20, 1, and 12, respectively. 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 is 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 4 shows run time performance of GA varying number of joins. Exhaustive enumeration was performed for only two cases (3-&4-joins) since cases for more than 4-join were meaningless in terms of run-time comparison. 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 5 shows the run time efficiency of GA with a 10-join query, varying the number of sites.



(Fig. 4) Execution Time (in seconds) of GA vs. Number of Joins



(Fig. 5) Execution Time (in seconds) of GA vs. 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.

5. Conclusions

In this paper we have presented a solution technique for designing query execution plans in distributed databases. Our solution technique solves 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. The OLTP type transactions require high throughput, hence total time minimization objective function is appropriate. The Decision Support type transactions require low response time, thus response time minimization objective

is appropriate. Our results show that the optimal allocations are quite different with the two objective functions: total time and response time minimization. Response time minimization could be achieved through a large variety of parallel execution and parallel transmission. In order to maximize these parallelisms, subqueries were allocated to as many sites as possible. On the other hand, 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.

In our research, we assumed that query execution order as given and determined the operation allocations. It should be noted that the query execution order and the operation allocation are two interdependent decisions. The research can be extended by integrating both the sub-problems and providing the optimal query execution plans. The GA algorithm developed in this research can be extended to include the additional sub-problem.

References

[1] P.M.G Apers, "Data Allocation in Distributed Database Systems," ACM Trans. on Database Systems, Vol.13, No.3, pp.263-304, Sep., 1988.

[2] J. Arcangeli, A. Hameurlain, E. Migeon and F. Morvan, "Mobile Agent Based Self-Adaptive Join for Wide-Area Distributed Query Processing," Journal of Database Management, Vol.15, No.4, pp.25-44, 2004.

[3] J. Atkin and M. Norris, Total Area Networking: ATM, Frame Relay and SMDS Explained, John Wiley & Son, New York, N.Y., 1995

[4] F. Baiao, M. Mattoso and G. Zaverucha, "A Distribution Design Methodology for Object DBMS," Journal of Distributed and Parallel Databases, Vol.16, No.1, pp.45-90, 2004.

[5] B. Bergsten, M. Couprie and P. Valduriez, "Overview of Parallel Architectures for Database," The Computer Journal, Vol.36, pp.734-740, Aug., 1993.

[6] C-H Cheng, W-K Lee and K-F Wong, "A Genetic Algorithm-Based Clustering Approach for Database Partitioning," IEEE Transactions on Systems, Man, and Cybernetics, Vol.32, No.3, pp.215-230, 2002.

[7] D.W. Cornell and P.S. Yu, "On Optimal Site Assignment for Relations in the Distributed Database Environment," IEEE Transactions on Software Engineering, Vol.15, No.8, pp.1004-1009, Aug., 1989.

[8] J. Cuadrado, Optimize Database Queries, Byte, pp.57-63, July, 1995.

[9] L. Davis, Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, N.Y., 1991.

[10] J. Du, R. Alhajj and K. Barker, "Genetic Algorithms Based Approach to Database Vertical Partitioning," Journal of Intelligent Information Systems, Vol.26, No.2, pp.167-183, 2006.

[11] W. Du, M. Shan and U. Dayal, "Reducing Multidatabase Query Response Time by Tree Balancing," Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, pp.293-303, May, 1995.

[12] R. Florin and D. Alin, "Sketches for Size of Join Estimation," ACM Transactions on Database Systems, Vol.33, No.3, pp.1-46, 2008.

[13] O. Frieder and C. Baru, "Site and Query Scheduling Policies in Multicomputer Database Systems," IEEE Transactions on Knowledge and Data Engineering, Vol.6, No.4, pp.609-619, Aug., 1994.

[14] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing, 1989.

[15] N. Gorla, "An Object-Oriented Database Design for Improved Performance," Data and Knowledge Engineering, Vol.37, pp. 117-138, 2001.

[16] G. Graefe, "Query Evaluation Techniques for Large Databases," ACM Computing Surveys, Vol.25, pp.73-90, June, 1993.

[17] X. Gu, W. Lin and V. Bharadwaj, "Practically Realizable Efficient Data Allocation and Replication Strategies for Distributed Databases with Buffer Constraints," IEEE Transactions on Parallel & Distributed Systems, Vol.17, No.9, pp.1001-1013, Sep., 2006.

[18] J. M. Johansson, S. T. March and J. D. Naumann, "Modeling Network Latency and Parallel Processing in Distributed Database Design," Decision Sciences, Vol.34, No.4, pp.677-706, 2003.

[19] D. Kossmann, "The State of the Art in Distributed Query Processing," ACM Computing Surveys, Vol.32, No.4, pp.422-469, Dec., 2000.

[20] U R. Kulkarni and H. K. Jain, "Interaction Between Concurrent Transactions in the Design of Distributed Databases," Decision Sciences, Vol.24, No.2, pp.253-277, 1993.

[21] A. Kumar, and R. Pathak, "Genetic Algorithm Based Approach for File Allocation on Distributed Systems," Computers & Operations Research, Vol.22, No.1, pp.41-55, 1995.

[22] B. Li and W. Jiang, "A novel stochastic optimization algorithm," IEEE Trans. on Systems, Man, and Cybernetics, Part B, Vol.30, No.1, 2000.

[23] S-J. Lim and Y-K Ng, "Vertical Fragmentation and Allocation in Distributed Deductive Database Systems," Information Systems, Vol.22, No.1, pp.1-24, 1997.

[24] S.T. March and S. Rho, "Allocating Data and Operations to Nodes in Distributed Database Design," IEEE Trans. on Knowledge and Data Engineering, Vol.7, No.2, April, 1995.

[25] T. Martin, K. Lam and J. Russel, "An Evaluation of Site Selection Algorithms for Distributed Query Processing," The Computer Journal, Vol.33, No.1, pp.61-70, 1990.

[26] Z. Michalewicz and D. Fogel, How to Solve It: Modern Heuristics, 2nd edition, Springer, Berlin, 2004.

[27] M. Ozsu and P. Valduriez, Principles of Distributed Database Systems, Englewood Cliffs, Prentice-Hall Inc., 1991.

[28] S. Seshadri and B. Cooper, "Routing Queries through a Peer-to-Peer InfoBeacons Network Using Information Retrieval Techniques," IEEE Transactions on Parallel & Distributed Systems, Vol.18, No.12, pp.1754-1765, Dec., 2007.

[29] S.K. Song and N. Gorla, "A Genetic Algorithm for Vertical Fragmentation and Access Path Selection," The Computer Journal, Vol.43, No.1, pp.81-93, 2000.

[30] J. D. Schaffer, R. A. Caruana, L. J. Eshlman and R. Das, "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, pp.51-60, 1989.

[31] J. Srivastava and G. Elsesser, "Optimizing Multi-Join Queries in Parallel Relational Databases," Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems, pp.84-92, 1993.

[32] M. Syam, "Allocating Fragments in Distributed Databases," IEEE Transactions on Parallel & Distributed Systems, Vol. 16, No.7, pp.577-585, Jul., 2005.

[33] A.M. Tamhankar and S. Ram, "Database Fragmentation and Allocation: An Integrated Methodology and Case Study," IEEE Trans. on Systems, Man, and Cybernetics, Vol.28, No.3, pp.288-305, May, 1998.

[34] L. The, "Distributing Data Without Choking the Net," Datamation, Vol.40, pp.35-36, Jan. 7, 1994.

[35] C. T. Yu, C. Chang, M. Templeton, D. Brin and E. Lund, "Query Processing in a Fragmented Relational Distributed System: Mermaid," IEEE Transactions on Software Engineering, Vol.11, pp.795-809. Aug., 1985.

[36] M. Ziane, M. Zait and P. Borla-Salamet, "Parallel Query Processing in DBS 3," Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems, pp.93-102. 1993.

송 석 규

e-mail : sksong@ysu.ac.kr

1977년 성균관대학교 화학공학과(학사)

1990년 미국 Arizona State Univ. MIS (석사)

1997년 미국 Cleveland State Univ. MIS (박사)

1995년~1997년 ㈜포스데이타(POSDATA) 차장

1997년~현 재 영산대학교 호텔경영학과 교수

관심분야 : 데이터베이스, 소프트웨어공학, 유전자알고리즘, IT경영전략컨설팅, 호텔정보시스템