A FRAMEWORK FOR DESIGN OF PARTIALLY REPLICATED DISTRIBUTED DATABASE SYSTEMS WITH MIGRATION BASED GENETIC ALGORITHMS

Sukkyu Song

Department of Hotel Management, Youngsan University, Busan, Korea

ABSTRACT

For partially replicated distributed database systems to function efficiently, the data (relations) and operations (subquery) of the database need to be located, judiciously at various sites across the relevant communications network.The problem of allocating relations and operations to the most appropriate sites is a difficult one to solve so that genetic algorithms based on migration are proposed in this research. In partially replicated distributed database systems, the minimization of total time usually attempts to minimize resource consumption and therefore to maximize the system throughput. On the other hand, the minimization of 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. Workload balancing implies the reduction of the average time that queries spend waiting for CPU and I/O service at a network site, but its effect on the performance of partially replicated distributed database systems cannot be isolated from other distributed database design factors. In this research, the total cost refers to the combination of total time and response time. This paper presents a framework for total cost minimization and workload balancing for partially replicated distributed database systems considering important database design objectives together. The framework incorporates both local processing, including CPU and I/O, and communication costs. To illustrate its suitability, experiments are conducted, and results demonstrate that the proposed framework provides effective partially replicated distributed database design.

KEYWORDS

Partially replicated distributed database, Total time minimization, Response time minimization, Workload balancing, Operation allocation, Data allocation,Genetic algorithm

1. INTRODUCTION

Two important aspects for design of partially replicated distributed database systems are data allocation and operation allocation. Data allocation is to allocate relations to sites so that the performance of distributed database are improved. Operation allocation refers to query execution plan indicating which operations (sub queries) should be allocated to which sites in a computer network, so that query processing costs are minimized. The allocation of relations can be either non-replicated or replicated. In non-replicated allocation, each relation is mapped to exactly one site. In replicated allocation each relation is mapped to one or more sites. Replicated allocation can be either fully replicated or partially replicated to a system. A fully replicated database system refers to a system where a complete copy of the relation exists at every site in the system. In a partially replicated database system, copies of relations exist at some of the sites in the system. The replication of relations can improve data availability and reliability in the event of site failures and decrease the response time for queries partly through parallel query execution. The replication of relations, however, will increase the data storage cost and the transmission cost

This work was supported by a 2017 research grant from Youngsan University, Republic of Korea.

DOI : 10.5121/ijdms.2017.9301

for updates because some relations need to be updated simultaneously, which also incurs a communication overhead for concurrency control. Obviously, the advantage of full replication is that queries may be completely executed at one of the sites in the network without incurring any communication overhead.

In partially replicated distributed database systems, the minimization of total time usually attempts to minimize resource consumption and therefore to maximize the system throughput. In other words, if the usage of resources (CPUs, I/Os, and communication channels) for a given transaction is minimized, more transactions can be processed for a given time period, which in turn means that 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 as shown by [5]. In this research, the total cost refers to the combination of total time and response time. The workloads represent the amount of services at a network site expressed in terms of the CPU and I/O loads when executing a query. Workload balancing implies the reduction of the average time that queries spend waiting for CPU and I/O services at a site so that the queuing delays at the network site are minimized as shown by [3], [4], [6] and [13].

In a fully replicated distributed database system, arriving transactions can be serviced at any sites. In a partially replicated distributed database system, however, arriving transactions should be routed (allocated) to the site owning the referenced relation for processing. In other words, the data allocation scheme gives to some extent limitations on decision choices in developing workload balancing schemes. In a partially replicated distributed database system, this means that data allocation and workload balancing are not independent. We, therefore, believe that in a partially replicated distributed database system the workload balancing issue should be taken into account in the early design step of data allocation according to [12]. In this paper, we propose a framework for total cost minimization and workload balancing as important design factors together. In this research, the total cost refers to the combination of total time and response time. In order to construct the framework for total cost minimization and workload balancing, we consider total cost minimization as the primary objective and workload balancing as the secondary objective for partially replicated distributed database design.

In this research, we propose migration based genetic algorithms to explore the interactions not only between total cost minimization operation allocation and data allocation, but also between workload balancing and data allocation. During the last-three decades there has been a glowing interest in algorithms which rely on analogies to natural processes. The emergence of massively parallel computers made these algorithms of practical interest. The best known algorithms in this class include genetic algorithms, simulated annealing, classifier systems, and neural networks as shown by [8]. Genetic algorithms are heuristic solutions that have been used to solve intractable problems in distributed database systems as shown by [1],[2], [9] and [11]. When compared to other heuristic algorithms as demonstrated in [8] and enumerative techniques, genetic algorithms provide global ‘optima’ in much less time as shown by [10].

This paper is organized as follows. Section 2 has discussion of cost models for genetic algorithms. In Section 3, the framework for total cost minimization and workload balancing is described in detail using the migration based genetic algorithms. In Section 4, the illustration of genetic algorithm procedures is presented. In section 5, experiments and their results are presented. Section 6 provides conclusions.

2. DEVELOPMENT OF COST MODELS FOR GENETIC ALGORITHMS

2.1. TOTAL TIME MODELS

The total time for each query is the sum of local processing times and communication times for all subqueries. Total Time= $\sum_j (LP_j^k + COM_j^k)$, where LP_j^k represent the local processing time of the subquery j (a node in the query tree in Figure 2) of a query k. COM_j^k represents the communication time of transmitting the input relation(s) to the site at which the sub query j of a query k is being executed.

2.1.1. LOCAL PROCESSING TIME (LP_j^k)

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:

$$LP_j^k = \sum_t Y_{jt}^k \ (IO_t \ \sum_i Z_{ij}^k B_{ij}^k + CPU_t \ \sum_i Z_{ij}^k B_{ij}^k) \quad (1)$$

where B_{ij}^k 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 mainmemory,
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:

$$LP_j^k = \sum_t Y_{jt}^k \ IO_t \ \sum_m \sum_i \rho_m \ Z_{ijp[m]}^k B_{ijp[m]}^k + \quad (2a)$$
$$\sum_t Y_{jt}^k \ (IO_t \ \prod_i Z_{ij}^k B_{ij}^k + CPU_t \ \prod_i Z_{ij}^k B_{ij}^k) \quad (2b)$$

where ρ_m represents the selectivity of the two previous operations (m = 1 or 2), and where the selectivity is the ratio of output relation size and input relation size, $B_{ijp[m]}^k$ 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 $B_{ijp[m]}^k$ (the size of intermediate results being stored at the join site) to B_{ij}^k (the size of same intermediate results being retrieved for the current join operation) for notational convenience so that B_{ij}^k will be used for the next join operation with the join selectivity of the current join operation.

2.1.2. COMMUNICATION TIME (COM_j^k)

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:

$$COM_j^k = \sum_m \sum_t \sum_p Y_{jp[m]t}^k \; Y_{jp}^k \; C_{tp} \; (\sum_i Z_{ijp[m]}^k B_{ijp[m]}^k)$$

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 $Z_{ijp[2]}^k$ for all i is 0 (also $B_{ijp[2]}^k$ = 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. In this research, we ignore this synchronization time, since this is usually a fixed overhead cost and it is not variable like data transfer cost.

2.2. RESPONSE TIME MODEL

In a partially replicated 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 parallelismas shown by [5]. 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. Inter-operation parallelism refers that several subqueries in a single query can be executed in parallel. In this research we assume the join operation is performed using the sequential processing method, and we are concerned only with parallelism in a single query, not among multiple queries.

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:

$$\text{Response time } RT_j^k = COM_j^k(p[1]) + LP_j^k(p[1]) + RT_j^k(p[1])$$

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

The first term $COM_j^k(p[1])$ is to calculate the communication time sending the results to the query originating site ($Z_{ijp[2]}^k$ for all i is 0 and $B_{ijp[2]}^k$ = 0) and the $LP_j^k(p[1])$ refers to the local processing time of the final operation. For the recursive function $RT_j^k(p[1])$ (but we will use RT_j^k 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.

2.2.1. SCENARIO – 1

The join operation j and the sites two preceding operators p[1] and p[2] areexecuted 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 processingtime 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.

2.2.2. 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) [6]. Then RT^k_j is given by

$$\text{Max } \{ LP^k_j, \tag{3a}$$
$$LP^k_j(p[1]) + RT^k_j(p[1]), \tag{3b}$$
$$LP^k_j(p[2]) + RT^k_j(p[2]), \tag{3c}$$
$$COM^k_j(p[1]) + COM^k_j(p[2]) \} \tag{3d}$$

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 sub query j and (3b) and (3c) are the processing times for the two preceding operations of sub query j. The communication costs will be additive, since those are the overheads on the receiving node, as represented by (3d).

2.2.3. SCENARIO –3

The sites at which two preceding operations of sub query j are performed are different and the join sub query 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:

$$\text{Max } \{ LP^k_j + LP^k_j(p[1]) + RT^k_j(p[1]), \tag{4a}$$
$$LP^k_j(p[2]) + RT^k_j(p[2]), \tag{4b}$$
$$COM^k_j(p[2]) \} \tag{4c}$$

$$\text{where} \quad COM_j^k(p[2]) \;=\; Y_{jp[2]p}^k Y_{jt}^k C_{tp} \left(\sum_i Z_{ijp[2]}^k B_{ijp[2]}^k\right)$$

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.

2.2.4. 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, $Y_{jp[1]p}^k Y_{jt}^k C_{tp} \neq 0$, $Y_{jp[2]p}^k Y_{jt}^k C_{tp} \neq 0$ and $Y_{jp[1]p}^k Y_{jp[2]p}^k C_{pp} = 0$. Also, there will be no parallelism between the operations p[1] and p[2]. Then RT_j^k is given by

$$\text{Max} \quad \{ LP_j^k, \tag{5a}$$

$$LP_j^k(p[1]) + LP_j^k(p[2]) + RT_j^k(p[1]) + RT_j^k(p[2]), \tag{5b}$$

$$COM_j^k(p[2]) + COM_j^k(p[2]) \} \tag{5c}$$

$$\text{where} \quad COM_j^k(p[1]) \;=\; Y_{jp[1]p}^k Y_{jt}^k C_{tp} \left(\sum_i Z_{ijp[1]}^k B_{ijp[1]}^k\right)$$

$$COM_j^k(p[2]) \;=\; Y_{jp[2]p}^k Y_{jt}^k C_{tp} \left(\sum_i Z_{ijp[2]}^k B_{ijp[2]}^k\right)$$

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 $Z_{ijp[m]}^k$ 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. $Z_{ij}^k(p[1]) * Z_{ij}^k(p[2]) = 0 \;\forall_i$, otherwise local processing times can be additive in the worst case.

2.3. QUERY TREE AND UPDATE TREE MODEL FOR UPDATE TRANSACTION

A query tree is illustrated in the query part in Figure 1. A node is called a leaf node (F1 and F2) if it has no incoming arcs; that is, it represents the relations in the database. A node is called an operation node (nodes 1, 2 and 3) if it has incoming and outgoing arcs. The operation nodes represent the relational operations. The operation nodes such as 1 and 2 represent a unary operation such as selection, projection or a combination of both, and the operation node such as 3 represents a binary operation such as join or union. Sometimes a binary operation is performed on an input relation directly without any unary operation(s), and in this case the unary operation node connected to the corresponding input relation is called a dummy operation node. An operation node without any outgoing arcs is called a result node (node 4). An arc represents the transmission of a (intermediate) relation into the operations, such as f3, f4 and f5.

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.

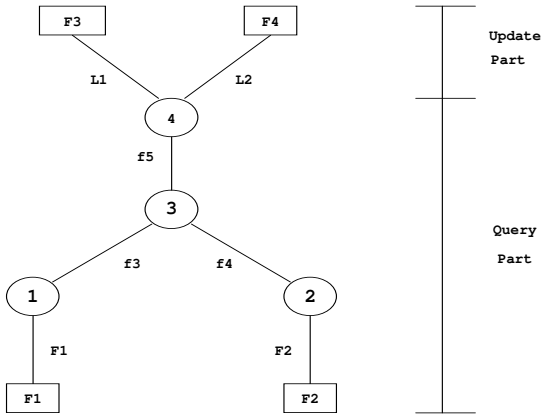

Figure 1. Query Tree for Update Transaction

An update transaction may be viewed as a two-part action, wherein the first part corresponds to a query transaction, followed by the second part which updates the value of a set of relations, as shown in Figure 1. The simplified SQL statement for the update query tree Figure 1 may be as follows:

- UPDATE F3, F4 Alias F
- SET F.z = F.z * 1.1
- WHERE F.k In (SELECT k
  FROM F1, F2
  WHERE F1.x = F2.y)

In the second part of an update transaction, the update values (L1 and L2, which are the same as the intermediate relation f5 resulting from the final operation 3 in Figure 1) resulting from the first part must be sent from the update initiation site (site for operation 4 in Figure 1) to all sites that have a copy of the relation being updated, and then the relation must be updated at each site (for example, two copies of F3 and three copies of F4 in Figure 1), which incurs CPU and I/O costs at each site. In Figure 1, two relations 1 and 2 are referenced by the query part of update transaction, and then both relations are updated according to the update value resulting from the query part.

2.3.1. COST MODELS FOR UPDATE TRANSACTION

As mentioned in the previous section, the total cost for executing all query (either OLTP or decision-support) and update transactions against a particular data allocation scheme will determine the goodness of its data allocation scheme, and it is represented as follows:

$$\text{Total Cost} = \sum_k \sum_t F(k,t)Q(k,t) + \sum_u \sum_t F(u,t)U(u,t)$$

Where F(k,t) and F(u,t) are the frequencies of query k originating at site t and update u originating at site t per unit time, and Q(k,t) and U(u,t) are the cost of query k and update u transactions originating at site t. Our objective is to minimize this total cost.

We now define the update transaction cost model. Before describing the cost model, we first introduce one more variable U_i, specifying relations updated by the update transaction. U_i is 1 if relation i is updated by the update transaction; otherwise, it is 0. The update transaction cost is defined as follows:

$$U(u,t) = Q(u,t) +$$
$$\sum_t \sum_p C_{tp} \sum_i U_i X_{it} L_i + \quad (1)$$
$$\sum_t (IO_t \sum_i U_i X_{it} B_i^u + CPU_t \sum_i U_i X_{it} B_i^u) + \quad (2)$$
$$\sum_t IO_t \sum_i U_i X_{it} L_i \quad (3)$$

Where

B_i^u is the number of blocks of relation i updated by update u,
L_i is the update value in number of blocks for the relation i, which is the same as the final result from the query part Q(u,t),
IO_t is the I/O cost coefficient (speed) of site t in msec per page (4k bytes),
CPU_t is the CPU cost coefficient (processing speed) of site t in msec per page (4k bytes),
C_{tp} is the communication cost coefficient (channel speed) between site t and site p in msec per page (4k bytes),
X_{it} represents data allocation; relation i is stored at site t.

Note that calculation of query execution time for the query part Q(u,t) of the update transaction is exactly the same as that of the total time model (see below for details). The reason for using the total time model for Q(u,t) is that the update transactions typically occurred in the DEBIT/CREDIT type of transactions in the banking industry, which in general require high throughput. Therefore, the calculation of Q(u,t) is the same as Q(k,t) of total time introduced in Chapter V. In the formula, (1) represents the communication cost for sending the update values (L_i) from the update initiation site to all sites that have the copy of the relation being updated; (2) represents I/O cost for reading the required relation into main memory and CPU cost for processing the update; and (3) represents the update cost for writing the updated values back to disk. Calculation of Q(u,t) is as follows:

$$Q(u,t) = \sum_j (LP_j^k + COM_j^k) \quad (1)$$
$$LP_j^k = \sum_t Y_{jt}^k (IO_t \sum_i Z_{ij}^k B_{ij}^k + CPU_t \sum_i Z_{ij}^k B_{ij}^k) \quad (2)$$
$$LP_j^k = \sum_t Y_{jt}^k IO_t \sum_m \sum_i \rho_m Z_{ijp[m]}^k B_{ijp[m]}^k + \quad (3a)$$
$$\sum_t Y_{jt}^k (IO_t \prod_i Z_{ij}^k B_{ij}^k + CPU_t \prod_i Z_{ij}^k B_{ij}^k) \quad (3b)$$
$$COM_j^k = \sum_m \sum_t \sum_p Y_{jp[m]t}^k Y_{jp}^k C_{tp} (\sum_i Z_{ijp[m]}^k B_{ijp[m]}^k) \quad (4)$$

Where

LP_j^k represents the local processing time of the subquery j of a query k.

COM_j^k 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.
B_{ij}^k is the number of blocks of relation i accessed by subquery j of query k.
$B_{ijp[m]}^k$ 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 previous operation, and 2 for the right previous operation.
ρ_m represents selectivity of the two previous operation (m = 1 or 2), and selectivity refers to the ratio of relation size reduction after an operation.
Y_{jt}^k represents operation allocation and is 1 if subquery j of query k is done at site t; otherwise, it is 0.
$Y_{jp[m]t}^k$ 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.
Z_{ij}^k is 1 if input (or intermediate) relation(s) i is referenced by subquery j of query k.
$Z_{ijp[m]}^k$ is 1 if input (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.

(1) represents the total query execution time for the query part Q(u,t) of the update transaction and is the sum of all local processing times and communication times. (2) represents the local processing time for the subquery j of the query k when the subqueries are unary operations such as the selection or projection operation. (3a) represents the I/O time in storing the intermediate results of previous operations to the site of the current join operation before the execution of the join. (3b) represents the I/O and CPU processing times for the current join operation. (4) represents the communication time for join operations when either of the (intermediate) relation(s) to be joined is not produced at the site at which the join operation is performed. (4) is also used for the communication time for sending the final result 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 $Z_{ijp[2]}^k$ for all i is 0 (also $B_{ijp[2]}^k = 0$).

2.4. COST MODEL FOR WORKLOAD BALANCING

We define the unbalanced factor (UBF) as the sum of the absolute deviation of site workloads from the average network load. The objective function for workload balancing is then defined to minimize UBF. Minimization of UBF gives a load distribution that has approximately balanced the network load. Note that if the network load among sites is balanced totally (all site have the same workload), the absolute deviation becomes zero. The objective function is defined as follows.

$$\text{Minimize UBF} = \sum_t \left| LI_t - LI_{av} \right| + \sum_t \left| LC_t - LC_{av} \right|$$

$$\text{subject to} \quad LI_{av} = \frac{1}{N} \sum_t LI_t$$

$$LC_{av} = \frac{1}{N} \sum_t LC_t$$

Where LI_t and LC_t represent the I/O and CPU workloads (I/O and CPU times), respectively, at the site t; LI_{av} and LC_{av} represent the average I/O and CPU workloads (I/O and CPU times), respectively, in the entire database; N represents the number of sites. We now define LI_t and LC_t as follows:

(1) For a selection or projection,

$$LI_t = \sum_k F(k,t) \sum_j Y_{jt}^k IO_t \sum_i Z_{ij}^k B_{ij}^k$$

$$LC_t = \sum_k F(k,t) \sum_j Y_{jt}^k CPU_t \sum_i Z_{ij}^k B_{ij}^k$$

(2) For a join,

$$LI_t = \sum_k F(k,t) \sum_j Y_{jt}^k IO_t \sum_m \sum_i \rho_m Z_{ijp[m]}^k B_{ijp[m]}^k +$$

$$\sum_k F(k,t) \sum_j Y_{jt}^k IO_t \prod_i Z_{ij}^k B_{ij}^k$$

$$LC_t = \sum_k F(k,t) \sum_j Y_{jt}^k CPU_t \prod_i Z_{ij}^k B_{ij}^k$$

Where

$F(k,t)$ represents the frequency of query k originating at site t,

Y_{jt}^k represents operation allocation, and is 1 if subquery j of query k is done at site t, otherwise it is 0,

Z_{ij}^k is 1 if input (or intermediate) relation(s) i is referenced by subquery j of query k,

$Z_{ijp[m]}^k$ is 1 if input (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,

IO_t is the I/O cost coefficient (speed) of site t in msec per page (4k bytes),

CPU_t is the CPU cost coefficient (processing speed) of site t in msec per page (4k bytes),

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

$B_{ijp[m]}^k$ 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 previous operation, and 2 for the right previous operation, and

ρ_m represents the selectivity of the two previous operation (m = 1 or 2), and the selectivity refers to the ratio of relation size reduction.

(3) For the update part of an update transaction,

$$LI_t = \sum_u F(u,t)\ IO_t \sum_i U_i X_{it} B_i + \sum_u F(u,t)\ IO_t \sum_i U_i X_{it} L_i$$

$$LC_t = \sum_u F(u,t)\ CPU_t \sum_i U_i X_{it} B_i$$

Where

$F(u,t)$ represents the frequency of update originating at site t,

X_{it} represents data allocation; relation i is stored at site t,

B_i^u is the number of blocks of relation i updated by update u, and

L_i is the update value in number of blocks for the relation i, which is the same as the final result from the query part of an update transaction.

Note that the query part of an update transaction is the same as (1) and (2) above.

3. FRAMEWORK FOR TOTAL COST MINIMIZATION AND WORKLOAD BALANCING

As described in the previous section, workload balancing can be used as the sole objective for the operation allocation as opposed to total cost minimization, and in our case, the total cost is the

combination of total time and response time. Our purpose, however, is to use workload balancing as the secondary objective for the data allocation while keeping total cost minimization as the primary objective. In order to accomplishthis objective, we employ four algorithms: one for the operation allocation whose objective is minimizing the total cost, one for workload balancing whose work workload depends on the optimized operation allocation resulted from the operation allocation algorithm, and two for the data allocation. The framework is proposed that these four algorithms interact with each other as shown in Figure 2.

In order to obtain better data allocation in terms of total cost as well as workload balancing, each step in the framework is adopted to use the genetic algorithm. Four genetic algorithms interact with each other according to the following steps:

(1) GA I produces the initial data allocation population by using binary strings. Note that the fitness of GA I is the total cost.
(2) GA II also produces the initial data allocation population, but by using a different random number seed (for example, 0.5) from the one (for example, 0.1) used for GA I. Note that the fitness of GA II is UBF.
(3) For each chromosome (data allocation scheme) from GA I, find the best operation allocation for each query (or query part of an update) by using GA III. In this step we obtain the fitness for each data allocation scheme in terms of the total cost.
(4) For each chromosome (data allocation scheme) from GA II, find the best operation allocation for each query (or query part of an update) by using GA IV. In GA IV, the best operation allocation for each query is obtained in terms of the total cost like GA III. But, once the best operation allocation for each query has been obtained, UBF is calculated for each data allocation scheme (chromosome of GA II) based on the best operation allocation obtained. So in this step we obtain the fitness for each data allocation scheme in terms of UBF based on the best operation allocation.

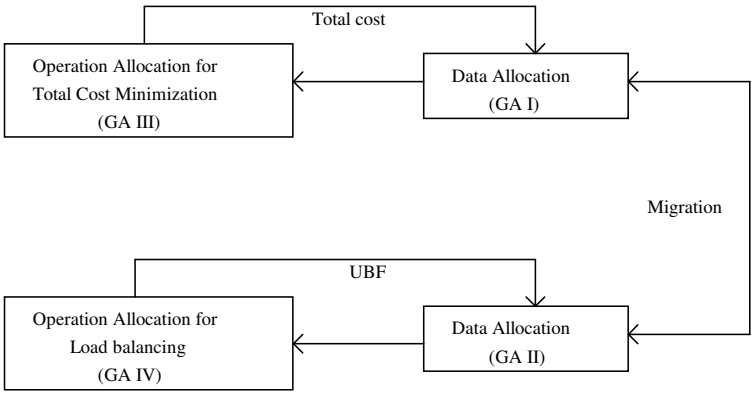

Figure 2. Framework for Total Cost Minimization and Workload Balancing Using Migration Based Genetic Algorithms

(5) Once all fitnesses (total costs) for GA I and (UBFs) GA II have been determined, the migration of selected chromosomes between GA I and GA II takes place. The number of chromosomes to be migrated is selected according to the random number which is always less than one-half of the total number of population, and these chromosomes are then selected based on their fitness (from the best one) in the current population of GA I and GA II respectively. Then the best chromosomes selected from GA I are migrated into the population of GA II, and at the same time the same number of the worst fitness chromosomes in GA II are removed from the population in GA II. The same migration

procedure occurs from GA II to GA I. The forced migration occurs at each generation during one-half of the total number of generations, and at the generation when the best fitness is not changed for three consecutive generations during subsequent generations. The reason to make the forced migration occur at the higher frequency during early generation and slow subsequent generation is that during early generation, there are not many differences between GA I and GA II in terms of total cost and UBF; but during later generation, since chromosomes in GA I and GA II are already optimized in terms of total cost and UBF, respectively, the migrated chromosomes do not make any significant contribution. The migration employed in this research, therefore, allows GA I to create as many diverse chromosomes as possible during early generations.

(6) Steps 3, 4, and 5 are repeated until the GAs I and II have reached the maximum number of generations. The intended data allocation we are looking for is the best fitness chromosome in GA I. The best solution in GA I will have not only the optimized total cost but also the better UBF, if not the optimized one, than the solution derived from the genetic algorithms.

4. ILLUSTRATION OF GENETIC ALGORITHM PROCEDURE

We use the same example used in [11] to illustrate the migration based genetic algorithms proposed in this research.We consider five relations as follows:

Faculty(F#, Fname, Dept)
Students(S#, Sname, Major)
Courses(C#, Cname, Dept, Credits)
Enrolls(S#, C#, Grade)
Advises(F#, S#)

We assume that the length of attributes measured in bytes is as follows:

F# (15), Fname (20), S# (15), Sname (20), Major (10),
C# (8), Cname (20), Dept (30), Credits (2), Grade (2)

The database statistics for five relations are as shown in Table 1 and the size of page block is assumed to be 4k bytes. We assume that the database consists of four sites and that the cost coefficients for each component are as shown in Table 2.The following SQL statements are used to illustrate the genetic algorithm procedure.

SQL Statement 1:

```
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#
```

SQL Statement 2:

```
SELECT FACULTY.Fname, STUDENTS.Sname
FROM   FACULTY, STUDENTS, COURSES, ENROLLS
WHERE   ADVISES.S# = STDUENTS.S#
AND      COURSES.C# = ENROLLS.C#
AND   ENROLLS.Grade< 'C'
OR      STUDENTS.Major = 'CIS'
```

SQL Statement 3:

```
UPDATE STUDENTS
```

```
SET     Major = 'CIS'
WHERE   FACULTY.F# = ADVISES.F#
AND       STUDENTS.S# = ADVISES.S#
AND   STUDENTS.Major = 'UNDECLARED'
AND   FACULTY.Fname = 'BOB SMITH'
```

As explained in the previous section, the high-level SQL statement can be transformed into the query tree, and the corresponding query tree for each SQL statement is shown in Figure 3. For the purpose of showing the calculation of the size of intermediate results for SQL statement 1, suppose that 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.It is assumed that the query-originating site is 4, and it is the node 6 in Figure 3.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)

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

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

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

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

operation 5: $f6 \rhd\lhd_{\text{S\# = S\#}} f7 \Rightarrow f8$ (f6 x f7 x ρ_j = 20 x 12 x 0.1 = 24 blocks)

Table 1. Relation Statistics

Relation No.	RelationName	Cardinality	Tuple Size	Size (bytes)	Size (blocks)
F1	Faculty	500	65	32,500	9
F2	Students	20,000	45	900,000	225
F3	Courses	500	60	30,000	8
F4	Enrolls	50,000	25	1,250,000	313
F5	Advises	20,000	30	600,000	150

Table 2. Cost Coefficients for Example

	Site				
		1	2	3	4
Communication Coefficients	1	0	13	12	11
	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

The results of the above size estimation for SQL statement 1 are shown in Figure 3. From the query tree, input(intermediate) relations used for each subquery are shown in Table 3. For example, the Enrolls relation is used by the operation 1 (Z_{ij}^{k} where i = 1 and j = 1), and the intermediate relations 4 and 5 are used by the operation 4 ($Z_{ijp[m]}^{k}$ where i = 4 and j = 4 for p[1], and i = 5 and j = 4 for p[2]), etc.In case of SQL statement 2, the size estimation of intermediate results is as follows:

operation 1: $\sigma_{cname='Database\$}$ (*Courses*) & $\pi_{c\#}$ (Courses) -> f6;
8 x 0.01 (ρ_s) x 0.03 (ρ_p) = 1
operation 2: $\sigma_{grade<'c'}$ (Enrolls) & $\pi_{s\#,c\#}$ (Enrolls) -> f7
313 x 0.2 (ρ_s) x 0.92 (ρ_p) = 58
operation 3: $\sigma_{major='cis'}$ (Students) & $\pi_{s\#,sname}$ (Students) -> f8
225 x 0.05 (ρ_s) x 0.78 (ρ_p)= 9
operation 4: Dummy -> f9 (size of Advises: 150)
operation 5: f6 $\triangleright\triangleleft_{c\#=c\#}$ f7 -> f10
1 (f6) x 58 (f7) x 0.01 (ρ_j) = 1
operation 6: f8 $\triangleright\triangleleft_{s\#=s\#}$ f9 -> f11
9 (f8) x 150 (f9) x 0.01 (ρ_j) = 14
operation 7: f10 $\triangleright\triangleleft_{s\#=s\#}$ f11 -> f12
1 (f9) x 14 (f10) x 0.01 (ρ_j) = 1

We assume that SQL statement 1 is an OLTP (Online Transaction Processing) type query so that its execution order results in the left deep query tree; that SQL statement 2 is a DSS (Decision Support System) type query so that its execution order results in the bushy query tree; and that SQL statement 3 is the update query tree (the query part of update is the OLTP type, so the left deep tree is used), and their corresponding query trees are as shown in Figure 4. So for this simple example problem, the objective function is to minimize a linear combination of total time andresponse time, in which the ratio of total time minimization and response time minimization is one to one. We assume that the frequencies of queries 1, 2, and update 3 are 3, 1, and 2, and the transaction originating site is assumed to be 4, 3, and 1 respectively.

4.1. INITIAL SOLUTION POOL

We first generate the initial solution pools (populations) for GA I and II using the probability 0.2 in order to get the genetic algorithms started with the best initial solution pools. Table 4 shows

Table 3. Input relation(s) used by subqueries ($Z_{ij}^k$and$Z_{ijp[m]}^k$)

(Intermediate) Relation	Subquery							
	1	2	3	4		5		6
				1	2	1	2	1
1	1	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0
4	0	0	0	1	0	0	0	0
5	0	0	0	0	1	0	0	0
6	0	0	0	0	0	1	0	0
7	0	0	0	0	0	0	1	0
8	0	0	0	0	0	0	0	1

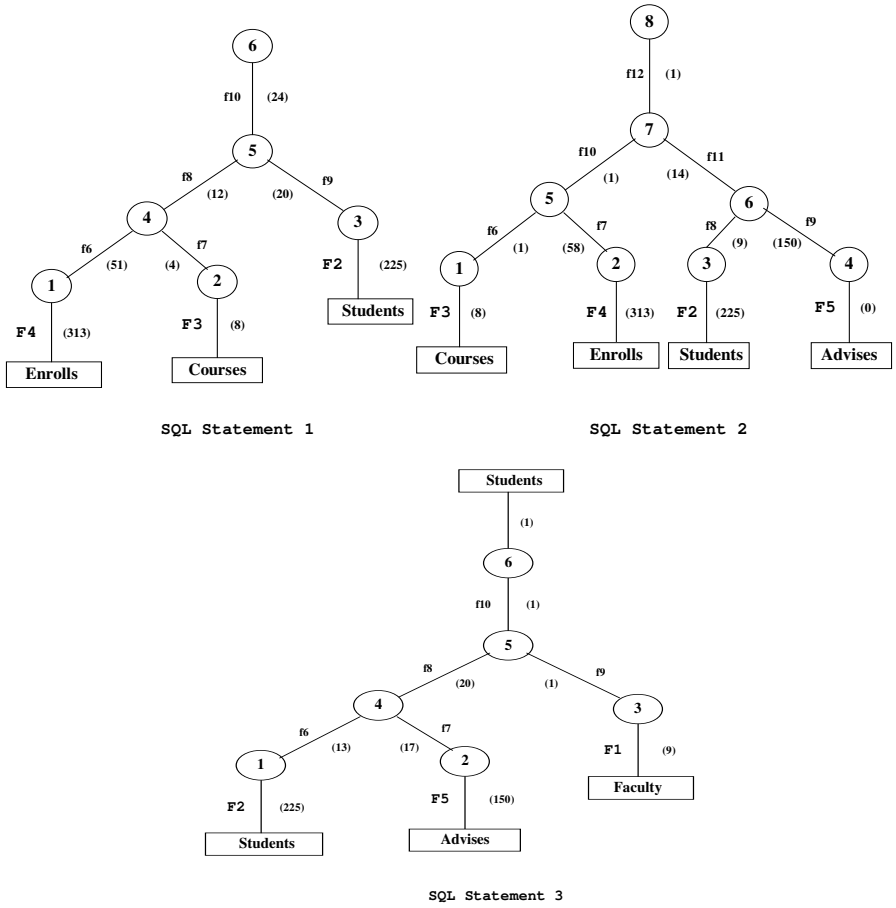

Figure 3. Query Trees for SQL Statements

the initial solution pools for GA I, and Table 5 shows those for GA II. Note that the fitness in Table 4 is calculated in terms of total cost while that in Table 5 is in terms of UBF.

Table 4. Initial Solution Pool for GA I

Solution #	Solution	Total Cost	UBF	Fitness
	F1 F2 F3 F4 F5			
1	0001 1000 1000 1000 1000	136396	138241	0.733
2	0001 1000 1000 0010 0001	133166	123071	0.750
3	0100 0001 1000 0010 0011	131528	135748	0.760
4	1000 0010 1010 0001 0010	130198	150728	0.768
5	1000 0100 0100 1000 0001	135178	71058	0.739
6	1010 0001 1100 0100 0001	136724	72802	0.731
7	1000 0100 1000 1000 0100	133934	95256	0.746
8	0010 0001 0010 0010 0010	131296	142643	0.761
9	0011 0101 0001 0100 0001	143308	117358	0.697
10	0010 0010 1010 0001 0001	133430	139326	0.749

Table 5. Initial Solution Pool for GA II

Solution #	Solution	Total Cost	UBF	Fitness
	F1 F2 F3 F4 F5			
1	1000 0010 0001 1000 0010	150148	129334	0.066
2	1000 1001 0001 0010 0100	58677	142037	0.170
3	1010 0010 0010 1010 1000	176918	128140	0.056
4	0001 0110 0100 0001 1010	144968	139438	0.068
5	0010 1000 0100 1000 0100	72417	135664	0.138
6	0100 0010 0100 1000 0100	83986	131524	0.119
7	0010 0001 1000 1000 0100	21050	137376	0.475
8	0100 0001 0100 1000 0100	20743	137380	0.482
9	0010 0001 0010 0001 0001	147299	139394	0.067
10	1000 0100 1000 0001 0010	80677	133014	0.123

Table 6. Operation Allocation for Solution #5

SQL Number	Frequency	OperationAllocation	Time	Remarks
1	3	12233	22,423	Total Time
2	1	2124133	31,097	Response Time
3	2	24133 -----	13,874 4,532	Query Part Update Part

The total cost for the data allocation scheme (solution #5 in Table 6) is the sum of all transaction execution costs, which is 135178. Based on these operation allocations, we then can calculate UBF for solution #5. Table 7 shows I/O and CPU loads for each operation at each site. The resulting I/O and CPU loads are given as: LI_{av} = 33939, LI_1 = 6199, LI_2 = 907, LI_3 = 32931, LI_4 = 27639, and LC_{av} = 1659.5, LC_1 = 331.5, LC_2 = 172.5, LC_3 =1518.5, LC_4 = 1359.5. Based on these results, the unbalanced factor (UBF) for the solution #5 is equal to 71058.

4.2. MIGRATION

As explained in the previous section, the forced migration of selected chromosomes between GA I and GA II occurs at each generation during the first half of the total number of generations. As an example, suppose that the number of chromosomes to be migrated is three. Then the solution pools in GA I and GA II are sorted according to their fitnesses. Solutions 4, 8, and 3 are the three best chromosomes in GA I, whereas solutions 8, 7, and 2 are the three best chromosomes in GA II, and solutions 9, 1, and 5 are the worst chromosomes in GA I, whereas solutions 3, 1, and 9 are the worst chromosomes in GA II. Therefore, chromosomes 4, 8, and 3 are migrated into GA II while removing chromosomes 3, 1, and 9 in GA II, and at the same time chromosomes 8, 7, and 2 are migrated into GA I while removing chromosomes 9, 1, and 5 in GA I. This completes the migration procedure, and the selection, crossover, and mutation are performed based on these new populations at the next generation.

4.3. FINAL SOLUTION POOL

Tables 8 and 9 illustrate the final solution pools for GA I and GA II at the 20th generation, respectively. They show that the total costs resulting from GA I are less than those from GA II whereas the UBFs resulting from GA II are much less than those from GA I. Since the purpose of this research is to obtain the best data allocation whose primary objective is total cost

minimization while load balancing is the secondary objective, the solutions 5, 7, and 10, which are the same, are the best data allocation scheme, and its total cost is 125332 and UBF 200884.

5. EXPERIMENTS AND RESULTS

In this section, we investigate how the data allocation pattern and the unbalanced factor are changed when a different objective function is used.We also investigate the effect of migration between two genetic algorithms. We will discuss this effect in terms of the total cost as well as the unbalanced factor.

Table 7. I/O and CPU Loads at Sites (I/O Load / CPU Load)

SQL	Operation No.	site			
		1	2	3	4
1	1	18780/939			
	2		456/24		
	3		12825/675		
	4			13986/612	
	5			14688/720	
2	1		152/8		
	2	6260/313			
	3		4275/225		
	4				0/0
	5	2340/58			
	6			27162/1350	
	7			522/14	
3	1		8550/450		
	2				6300/300
	3	360/18			
	4			9036/442	
	5			1476/40	
	Update		4294/225		
Total Load		27740/1328	34864/1832	66870/3178	6300/300
UBF at Site		6199/331.5	907/172.5	32931/1518.5	27639/1359.5

Table 8. Final Solution Pool for GA I

Solution #	Solution	Total Cost	UBF	Fitness
	F1 F2 F3 F4 F5			
1	0100 0010 0010 0010 0011	125372	200132	0.797
2	0010 0010 0001 1000 0010	129274	150792	0.773
3	1000 0010 1100 0010 0010	125548	199008	0.796
4	0100 0010 0010 0010 0100	127654	134034	0.783
5	0010 0010 0010 0010 0010	125332	200884	0.797
6	1000 0010 1000 1000 0010	129298	150106	0.773
7	0010 0010 0010 0010 0010	125332	200884	0.797
8	1000 0010 0100 1000 0010	129262	150063	0.773
9	0100 0010 1000 0010 0010	125564	198943	0.796
10	0010 0010 0010 0010 0010	125332	200884	0.797

Table 9. Final Solution Pool for GA II

Solution #	Solution	UBF	TotalCost	Fitness
	F1 F2 F3 F4 F5			
1	1000 0001 1000 1000 0100	15417	137436	0.648
2	1000 0001 1000 1000 0100	19979	137436	0.500
3	0100 0001 1000 1000 0100	20014	137416	0.499
4	0100 0001 1000 1000 0100	20365	137416	0.491
5	1000 0001 1000 1000 0100	15417	137436	0.648
6	1000 0001 1000 1000 0100	19979	137436	0.500
7	1000 0001 1000 1000 0100	15417	137436	0.648
8	1000 0001 1000 1000 0100	15417	137436	0.648
9	1000 0001 1000 1000 0100	20330	137436	0.491
10	1000 0001 1000 1000 0100	19979	137436	0.500

Finally, we compare two data allocation genetic algorithms, one using only interaction between total cost minimization operation allocation and data allocation (referred to as GA I/III) and one using only interaction between workload balancing operation allocation and data allocation (referred to as GA II/IV), using three different objective functions: total time, response time, and the combination of both.

For all experiments, we assume that the communication speed between any two pairs of sites is identical, which is set at 2.0. The processing speeds of all sites are also assumed to be identical, and are set I/O and CPU at 0.1 and 1.2, respectively. The configuration of the distributed database is assumed to consist of five sites and seven relations.

5.1. EFFECT OF OBJECTIVE FUNCTION

The research questionsinvestigated are as follows, and they are reiterated in terms of the unbalanced factor:

(1) for the total time minimization problem, the execution time can be minimized when queries are executed by using the smallest set of sites, which in turn means data themselves should be allocated to as few sites as possible.
(2) response time minimization can be obtained by having a large number of parallel local processing and transmissions at different sites as much as possible, which in turn mean data should be allocated to as many sites as possible.
(3) When the two objectives above are combined, data allocation should find a compromise suitable for total time minimization and response time minimization.

The above statements imply that the unbalanced factor for the data allocation scheme resulting from total time minimization should be larger than that of the data allocation scheme from response time minimization. And the unbalanced factor for the data allocation scheme resulting from minimization of a combination of total time and response time should be between those from total time minimization and response time minimization.

In order to investigate the effect of objective functions in terms of the unbalanced factor, the query and update originating site and their frequency are set as shown in Table 10. Table 11 shows solution patterns for all three minimization problems converge around the 20th generation. As expected, in the case of total time minimization, four relations are allocated to site 3 while two relations are allocated to sites 1 and 2, which in turn means that UBF is high. In case of

responsetime minimization, one or two relations are dispersed among five sites, so UBF (83285.8) is much

Table 10. Site and Frequency for Transactions

Transaction	Q1	Q2	Q3	Q4	Q5	Q6	Q7	U1	U2	U3	U4
Site	5	3	2	1	4	1	2	1	5	3	4
Frequency	50	50	50	50	2	2	2	10	10	10	10

Table 11. Solution Patterns

T + U	R + U	C + U
10000	01000	10000
00001	00010	00001
00110	10000	00100
01000	10000	10000
11100	00100	00100
00100	00001	01001
00100	00100	00001

Notation: T (total time minimization)
R (response time minimization)
C (combination of both)
+ U (with update transactions)
Note: Column: Sites; Row: Relations
Total Time Minimization: Time = 223727, UBF = 170779.6
Response Time Minimization: Time = 207027, UBF = 83285.8
Combination: Time = 216242.5, UBF = 133183.6

less than that of total time minimization (170779.6). In case of the combination of total time and response time minimization, the UBF is in between those of total time and response time minimization. This results show that the genetic algorithm finds solutions in a reasonable way according to its objective function.

5.2. EFFECT OF MIGRATION

The effect of the forced migration is investigated in this section. We first run the data allocation genetic algorithm without workload balancing, naming it OADA (Operation Allocation with Data Allocation). Then the genetic algorithms explained in this paper are run using the same query and update transactions, named LBDA (WorkLoadBalancing with Data Allocation including cost minimization operation allocation), for convenience.

As in the previous experiment, the genetic algorithms converge around the 20th generation. So all results are obtained at the 20th generation, and the number of chromosomes (the population size) is 20. First, in the case of total time minimization, the UBFs of LBDA are much less than those of OADA, while the best total time of OADA is 223727 (UBF = 170779) and that of LBDA is 223137 (UBF = 52583). This result shows that LBDA not only gives better total time but also much better UBF. Since OADA attempts only to minimize the total time, as a result the total time is minimized but UBF actually may be increased, as explained in the previous section. LBDA, however, not only attempts to minimize the total time but also UBF, and since the migration leads to more diverse chromosomes, LBDA results in better total time and UBF. This result shows the superiority of LBDA over OADA.

Second, in the case of response time minimization, there is not much difference between OADA and LBDA in terms of response time and UBF. As we described in the previous section, the response time minimization naturally disperses data among sites, and as a result, UBF is also minimized.

Third, in the case of minimization of the combination of total time and response time, the difference of UBF between OADA and LBDA is not as much as those resulting from total time minimization. But the total cost of LBDA is 238835, which is better than that of OADA, which is 239757. Figure 7 shows the slightly improved UBF of LBDA over OADA when we visually inspect the patterns between two results, although we do not prove that statistically.In summary, the above results show LBDA is superior to OADA.

5.3. COMPARISON BETWEEN THREE GENETIC ALGORITHMS

In this section we compare three genetic algorithms, OADA, LBDA, and one more genetic algorithm employing GA II and IV in Figure 2; that is, its objective is to minimize UBF, and we name it as UBFDA. The comparison is made in terms of total time, response time, and the combination of both. Three genetic algorithms start with the same initial populations. Since the objective of OADA is to minimize total cost, a combination of total time and response time, whereas that of UBFDA is to minimize the unbalanced factor, even though three genetic algorithms start with the same initial populations, the final results will be different in terms of total time, response time, and the combination of both respectively. One more issue we are investigating in this experiment is the implication of workload balancing; that is, workload balancing can lead to significant reduction in the average query response time since the waiting time for CPU and I/O services at sites of queries is reduced when queries are executed at the dynamic (run-time) environment. But since we employ only a static (compile-time) workload balancing in this research, it is hard to see the effect of actual response time (run-time) reduction of queries due to workload balancing unless we actually run simulation models or use mathematical queuing models based on data allocations and operation allocations (or workload balanced operation allocation) resulting from two genetic algorithms. Applying simulation or queuing models is, however, out of scope of this research. We, therefore, merely compare two genetic algorithms in terms of how total time, response time, and the combination of both are changed.

Table 12 shows the results based on two genetic algorithms, OADA and UBFDA. In case of total time minimization, the total time of UBFDA (19,195) is increased ascompared to that of OADA (15,595) even though UBF of UBFDA is significantly reduced. The main reason is that since UBFDA tends to spreads the workloads among sites, the total time is increased due to increased communications (note that the total time is minimized when subqueries are executed at the same site as much as possible).

6. CONCLUSION

This paper proposes the framework for total cost minimization and workload balancing. It is more realistic to solve the integrated problem of both data and operation problem based on total cost minimization and workload balancing than solve each problem separately.

To the best of our knowledge, this paper is the first attempt to consider total cost minimization and workload balancing in determination of data allocation and operation allocation.

Table 12: Comparison between Two Genetic Algorithms

Objective	Genetic Algorithm	Time	UBF
Total Time	OADA	15,595	11,967
UBF	UBFDA	19,159	2,200
Response Time	OADA	16,848	7,694
UBF	UBFDA	16,914	3,953
Combination	OADA	35,437	38,098
UBF	UBFDA	36,179	3,121

Computational results show the effectiveness of the framework. The proposed framework is more likely to provide a better data allocation and operation allocation for the performance of partially replicated distributed database systemsand also provides a better understanding of the underlying mechanisms for design of partially replicated distributed database systems.

REFERENCES

[1] C. Cheng, W. Lee, and K. Wong, (2002) "Genetic algorithm-based clustering approach for database partitioning," IEEE Transactions on Systems, Man, and Cybernetics, Vol. 32(3), pp. 215–230.

[2] J. Du, R. Alhajj, and K. Barker, (2006) "Genetic algorithms based approach to database vertical partitioning," Journal of Intelligent Information Systems, Vol. 26(2), pp. 167–183.

[3] X. Gu, W. Lin, and V. Bharadwaj,(2006) "Practically realizable efficient data allocation and replication strategies for distributed databases with buffer constraints", IEEE Transactions on Parallel and Distributed Systems, Vol. 17, issue 9, pp. 1001-1013.

[4] I. Hababeh, R. Omer, and B. Nicholas, (2007) "A high-performance computing method for data allocation in distributed database systems", Journal of Supercomputing, vol. 39, issue 1, pp. 3-18.

[5] J. Johansson, S. March, and J. Naumann, (2003) "Modeling network latency and parallel processing in distributed database design," Decision Sciences, Vol. 34(4), pp. 677–706.

[6] D. Kossmann, (2000) "The state of the art in distributed query processing," ACM Computing Surveys," Vol. 32(4), pp. 422–469.

[7] B. Li and W/ Jiang, (2000) "A novel stochastic opitimizationalgorirhm," IEEE Transactionn on Systems, Man, and Cybernetics: Part B, Vol 30 Issue 1, pp. 191-198.

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

[9] E. Sevince and A. Cosar, (2011) "An evolutionary genetic algorithm for optimization of distributed database queries," Computer Journal, Vol. 54 Issue 5, pp. 717–725.

[10] S. Song and N. Gorla, (2000) "Genetic algorithm for vertical fragmentation and access path selection," Computer Journal, Vol. 43 Issue 1, pp. 81–93.

[11] S. Song, (2015) "Design of distributed database systems: an iterative genetic algorithm," Journal of Intelligent Information Systems, (2015) " Journal of Intelligent Information Systems, Vol. 45, No. 1, pp. 29-59.

[12] A. Verma and M. Tamhankar, (1997) "Reliability-based optimal task-allocation in distributed-database management systems,"IEEE Transactions on Reliability, Vol. 46 Issue 4, pp. 452-459

[13] J. Wang and K. Jea,(2009) "A near-optimal database allocation and replication strategies for distributed databases with buffer constraints," Information Sciences, Vol. 179 Issue 21, pp. 3772-3790.

AUTHORS

Sukkyu Song
Professor, Youngsan University
Manager, Posdata Co., Seoul, Korea