International Journal of Advances in Computer Science & Its Applications – IJCSIA
Volume 4: Issue 3 [ISSN 2250-3765]
Publication Date : 30 September,2014

Design of Partially Replicated Distributed Database Systems with Combination of Total Cost and Workload Balancing

[Sukkyu Song]

***Abstract*—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 design factors 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, Response time, Workload, Operation (subquery) allocation, Data allocation, Genetic algorithms

I. Introduction

This Two important aspects for design of partially replicated distributed database systems are operation allocation and data allocation. Operation allocation refers to query execution plan indicating which operations (subqueries) should be allocated to which sites in a computer network, so that query processing costs are minimized. Data allocation is to allocate relations to sites so that the performance of distributed database are improved. 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 (Johansson et al. 2003). In this research, the total cost refers to the combination of total time and response time. Workload balancing among network sites is an important decision factor in designing partially replicated distributed database systems. Workload balancing, in general, refers to a query transaction scheduling scheme when used in distributed systems. To execute a query, a query should be routed (allocated) to a network site where the physical copy of the relation referenced by the query is available. In workload balancing, the objective of a query transaction scheduling scheme is to balance the workloads among network sites as opposed to total or response time minimization (Lin 2009). The execution of a query requires a certain amount of services from the system's resources such as CPU, I/O, and communication channel. From the local site's point of view, each site can be modeled as providing CPU and I/O services; that is, the service at a site can be expressed in terms of the CPU and I/O workloads. One of the implications of workload balancing is that it can lead to significant reductions in the average time that queries spend waiting for CPU and I/O services, and as a result, the average query execution time is significantly reduced; that is, the queuing delays at the network site are minimized (March and Rho 1995; Kossmann 2000; Cheng et al. 2002; Menon 2005; Gu et al. 2006; hababeh et al. 2007; Wangand Jea 2009).

Song, Sukkyu
Youngsan University
Busan, Korea

The workload balancing problem is commonly studied under the name of task allocation problem for general purpose distributed computer systems (Verma and Tamhankar, 1997; Jiang and Jiang, 2009). In the task allocation problem, it is assumed that incoming tasks (such as queries) can be serviced completely at any processing site. Workload balancing also has been applied in distributed database systems for solving the operation allocation problem. In a fully replicated distributed database system, as in the general purpose distributed computer 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, as considered in this research, the workload balancing issue should be taken into account in the early design step of data allocation (Tamhankar and Ram, 1998). In this paper, we propose a framework for total cost minimization and workload balancing considering important design factors together. 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 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 (Michalewicz and Fogel, 2004). Genetic algorithms are heuristic solutions that have been used to solve intractable problems in distributed database systems (Song and Gorla 2000; Cheng et al. 2002; Du et al. 2006; Sevince and Cosar 2011). When compared to other heuristic algorithms (Li and Jiang 2000) and enumerative techniques (Song and Gorla 2000), genetic algorithms provide global 'optima' in much less time.

This paper is organized as follows. Section 2 defines the cost models for total time, response time and workload. In section 3, the framework for total cost minimization and workload balancing is described in detail using the genetic algorithm procedures, in which we explain how the interactions are employed in this research. In section 4, experiments and their results are presented. Section 5 provides the conclusion for this research.

II. Development of Cost Models

A. *Query Tree Model for Query Execution Order*

In this section, we introduce a query tree model representing the query execution order. 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. In other words, each query, expressed in the SQL statement (or relational calculus) by users against a database, can be decomposed into a sequence of subqueries, which are equivalent to relational algebra operations, each of which works on a relation stored at the site. 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 assume that the relational algebra operations considered are projection, selection and join. Other operations could be included without altering any of the operation allocation algorithm proposed in this research. Also note that we assume that the structure of the query, i.e., the ordering of the operations, is fixed prior to

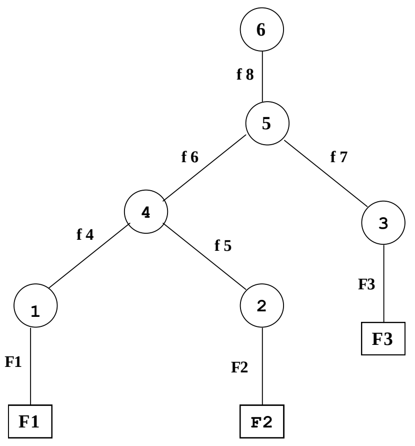

Figure 1. Query Tree

operation allocation. Our assumption is consistent with those of previous researchers in distributed databases design (Ozsu and Valduriez 1991).

A query tree is illustrated in Fig. 1. A node is called a leaf node (F1, F2 and F3) 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, 3, 4 and 5) if it has incoming and outgoing arcs. The operation nodes represent the relational operations. The operation nodes such as 1, 2 and 3 represent an unary operation such as selection, projection or a combination of both, and the operation nodes such as 4 and 5 represent 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 6). An arc represents the transmission of a (intermediate) relation into the operations, such as f4, f5, f6, f7 and f8.

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. There is no local processing time associated with leaf nodes and dummy operation nodes. The cost is associated with the arcs representing estimated communication times, transmitting the output relation of the source node from the site of source node to the site of the receiving node. We assume that each site has different I/O and CPU processing speeds (or costs) and communication channel speeds (or costs) for different pairs of sites are varied. Thus, both the local processing and communication times depend on the sites selected for the operation executions.

B. *Total Time Model*

The total time for each query is the sum of local processing times and communication times for all subqueries. Total Time $= \sum_j (LP_j^k + COM_j^k)$, where LP_j^k represent the local processing time of the subquery j (a node in the query tree) 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.

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:

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

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 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:

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

$$\sum_t Y_{jt}^k \left(IO_t \prod_i Z_{ij}^k B_{ij}^k + CPU_t \prod_i Z_{ij}^k B_{ij}^k\right) \quad (2b)$$

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

$B_{ijp[m]}^k$ 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. 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:

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

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 $Z_{ijp[2]}^k$ 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 -- we ignore this synchronization time, since this is usually a fixed overhead cost and it is not variable like data transfer cost.

C. *Response Time*

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 parallelism (Johansson et al. 2003). 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 $RT_j^k = COM_j^k(p[1]) + LP_j^k(p[1]) + RT_j^k(p[1])$

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. Fig. 2 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.

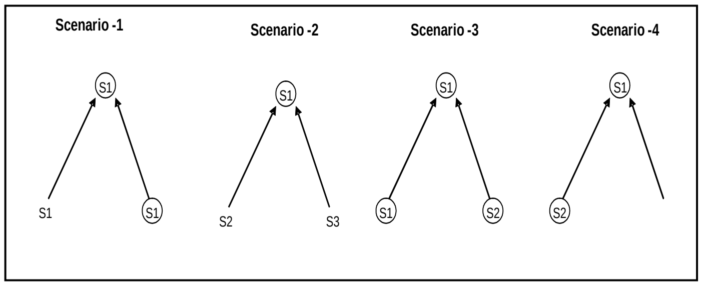

Figure 2. Four Joining Scenarios

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, $Y_{jp[1]t}^k Y_{jp[2]t}^k C_{tp} = 0$, $Y_{jp[1]t}^k Y_{jt}^k C_{tp} = 0$

and $Y_{jt}^k Y_{jp[2]t}^k C_{tp} = 0$

then RT_j^k can be calculated by using the equation.

$$LP_j^k + \sum_m LP_j^k(p[m] + RT_j^k(p[m]))$$

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. 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

$$\text{Max} \{ LP_j^k, \tag{3a}$$

$$LP_j^k(p[1]) + RT_j^k(p[1]), \tag{3b}$$

$$LP_j^k(p[2]) + RT_j^k(p[2]) \tag{3c}$$

$$COM_j^k(p[1]) + COM_j^k(p[2]) \} \tag{3d}$$

where $COM_j^k(p[1]) = Y_{jp[1]t}^k Y_{jp}^k C_{tp} (\sum_i Z_{ijp[1]}^k B_{ijp[1]}^k)$

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

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).

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,

$$Y^k_{jp[1]t} Y^k_{jt} C_{tt} = 0, \quad Y^k_{jp[2]p} Y^k_{jt} C_{tp} \neq 0$$

and $Y^k_{jp[1]t} Y^k_{jp[2]p} C_{tp} \neq 0$, then RT^k_j is given by:

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

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

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

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

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.

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

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

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

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

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

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

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.

D. Update Tree Model 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 3. The simplified SQL statement for the update query tree (Fig. 3) 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)
```

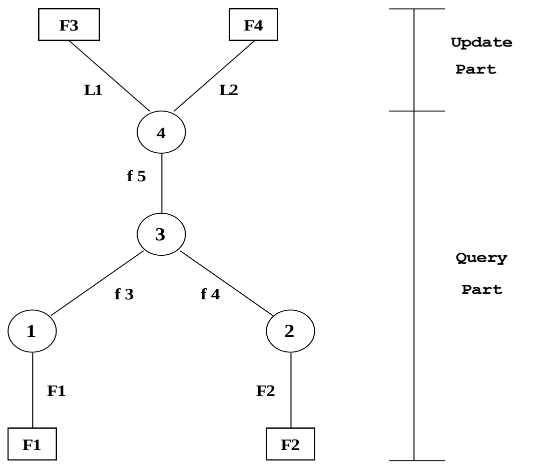

Figure 3. Query Tree for Update Transaction

International Journal of Advances in Computer Science & Its Applications – IJCSIA
Volume 4: Issue 3 *[ISSN 2250-3765]*
Publication Date : 30 September,2014

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 3) resulting from the first part must be sent from the update initiation site (site for operation 4 in Figure 3) 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 3), which incurs CPU and I/O costs at each site. In Figure 3, 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.

E. *Cost Models for Update Trasaction*

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:

$$U(u,t) = Q(u,t) +$$

$$\sum_t \sum_p C_{tp} \sum_i U_i X_{it} L_i + \qquad (1)$$

$$\sum_t (IO_t \sum_i U_i X_{it} B_i^u + CPU_t \sum_i U_i X_{it} B_i^u) + \qquad (2)$$

$$\sum_t IO_t \sum_i U_i X_{it} L_i \qquad (3)$$

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 occur 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 (Li) 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.

<u>Calculation of Q(u,t)</u>

$$Q(u,t) = \sum_j (LP_j^k + COM_j^k) \qquad (1)$$

$$LP_j^k = \sum_t Y_{jt}^k (IO_t \sum_i Z_{ij}^k B_{ij}^k + CPU_t \sum_i Z_{ij}^k B_{ij}^k) \qquad (2)$$

$$LP_j^k = \sum_t Y_{jt}^k IO_t \sum_m \sum_i \rho_m Z_{ijp[m]}^k B_{ijp[m]}^k + \qquad (3a)$$

$$\sum_t Y_{jt}^k (IO_t \prod_i Z_{ij}^k B_{ij}^k + CPU_t \prod_i Z_{ij}^k B_{ij}^k) \qquad (3b)$$

$$COM_j^k = \sum_m \sum_t \sum_p Y_{jp[m]t}^k Y_{jp}^k C_{tp} (\sum_i Z_{ijp[m]}^k B_{ijp[m]}^k) \qquad (4)$$

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

$Y^k_{jp[m]t}$ 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.

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

$Z^k_{ijp[m]}$ 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 $Z^k_{ijp[2]}$ for all i is 0 (also $B^k_{ijp[2]}$ = 0).

F. ***Workload Model***

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

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

subject to

$$LI_{av} = \frac{1}{N}\sum_t LI_t$$

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

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,

$$LI_t = \sum_k F(k,t) \sum_j Y^k_{jt}\, IO_t \sum_i Z^k_{ij}\, B^k_{ij}$$

$$LC_t = \sum_k F(k,t) \sum_j Y^k_{jt}\, CPU_t \sum_i Z^k_{ij}\, B^k_{ij}$$

(2) For a join,

$$LI_t = \sum_k F(k,t) \sum_j Y^k_{jt}\, IO_t \sum_m \sum_i \rho_m Z^k_{ijp[m]}\, B^k_{ijp[m]} + \sum_k F(k,t) \sum_j Y^k_{jt}\, IO_t \prod_i Z^k_{ij}\, B^k_{ij}$$

$$LC_t = \sum_k F(k,t) \sum_j Y^k_{jt}\, CPU_t \prod_i Z^k_{ij}\, B^k_{ij}$$

where

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

Y^k_{jt} represents operation allocation, and is 1 if subquery j of query k is done at 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,

$Z^k_{ijp[m]}$ 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^k_{ij} is the number of blocks of relation i accessed by subquery j of query k,

$B^k_{ijp[m]}$ 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 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.

III. 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 accomplish this 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 Fig. 4.

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.

(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.

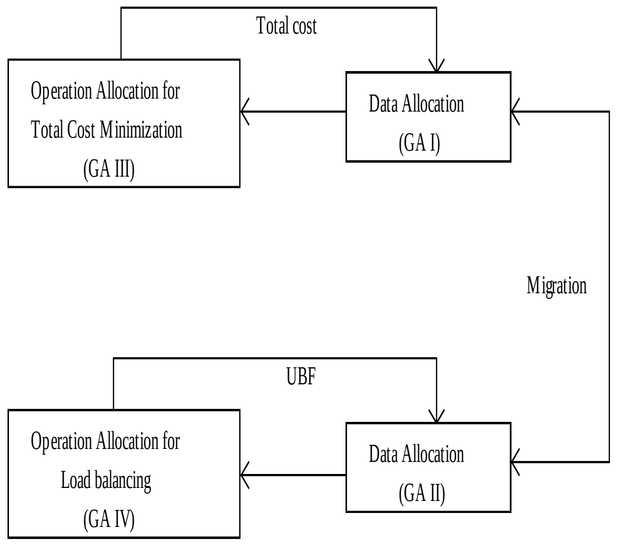
Total cost
Operation Allocation for Total Cost Minimization (GA III)
Data Allocation (GA I)
Migration
UBF
Operation Allocation for Load balancing (GA IV)
Data Allocation (GA II)

Figure 4: Framework for Total Cost Minimization and Workload Balancing with Four Genetic Algorithms

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. This migration can be compared to the concepts of migration demonstrated by Potts et al. (1994).

(6) Steps 3, 4, and 5 are repeated until the GAs I and II have reached the maximum number of generations.

IV. Experiments and Results

In this section, we investigate how not only the data allocation pattern but also the unbalanced factor is 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.

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 thee 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.

A. *Effect of Objective Function*

The research questions investigated 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 means 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 1. And Table 2 shows solution patterns for all three minimization problems converge around the 20th generation. The results at the 20th generation are shown as follows: 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 response time minimization, one or two

Table 1: Site and Frequency for Transactions

Transaction	q1	q2	q3	q4	Q11	Q12	Q13	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 2: 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

relations are dispersed among five sites, so UBF (83285.8) is much 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.

B. *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 (WorkLoad Balancing 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, Fig. 5 plotted against 20 chromosomes shows that 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. These results are illustrated in Fig. 6.

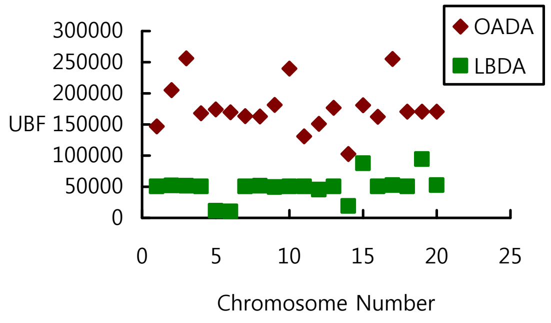

Figure 5: UBF by Total Time Minimization

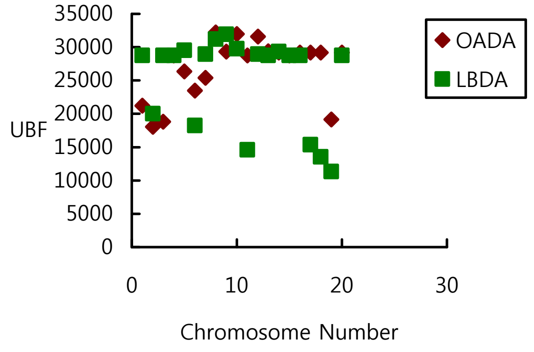

Figure 6: UBF by Response Time Minimization

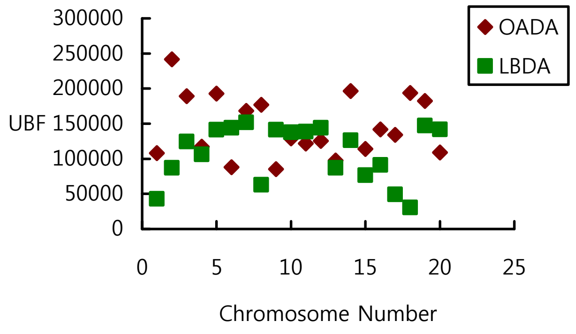

Figure 7: UBF by Minimization of Combination of Both

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 resulted from total time minimization. But the total cost of LBDA is 238835, which is better than that of OADA, which is 239757. Fig. 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.

C. *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 4; 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 (Carey and Lu 1986) 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 3 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 as compared 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).

In case of response time minimization, the response times are almost the same according to two genetic algorithms, whereas UBF of UBFDA is reduced as compared to OADA (7,694 -> 3,353). The reason is that response time minimization naturally spreads the workloads among sites in order to maximize the parallel executions of subqueries so that there is not much difference of response time itself between two genetic algorithms. In case of the combination of both, there is not much difference between OADA and UBFDA, but the interaction between the total time and response time minimizations leads to a little bit more reduction of total cost as compared to the reduction of response time.

The execution results of total time, response time, and combination of both up to the 20th generations are shown in Fig. 8, 9, and 10, respectively. All three genetic algorithms are

Table 3: Comparison between Two Genetic Algorithms

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

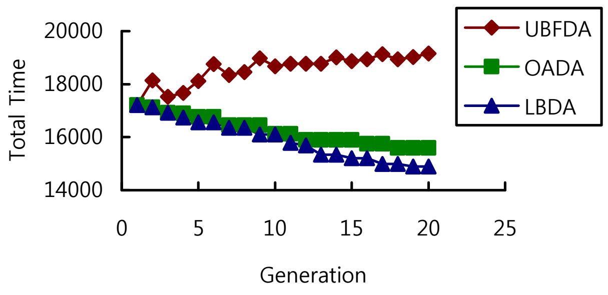

Figure 8: Comparison of Three Genetic Algorithms (Total Time Minimization)

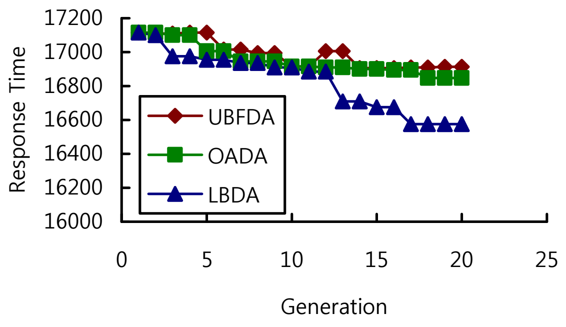

Figure 9: Comparison of Three Genetic Algorithms (Response Time Minimization)

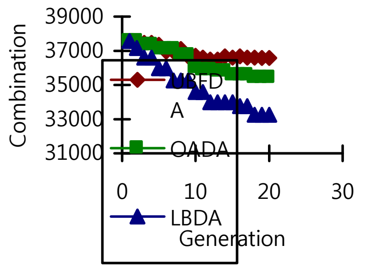

Figure 10: Comparison of Three Genetic Algorithms

(Minimization of Combination of Both)

stopped at the 20th generation since the solutions are all conversed around the 20th generation. All time cost for two genetic algorithms, OADA and LBDA, continue to decrease as the number of generation is increased, but those of UBFDA are not since its objective is not cost (time) minimization. One notable thing in case of total time minimization is that the total time tends to be increased as the number of generations is increased.

This result indicates that the total time minimization and workload balancing have a counter effect on each other. For all three cases, the results also show that LBDA always finds better solutions compared to OADA.

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

References

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

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

[3] 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 and Distributed Systems, Vol. 17 Issue 9, pp. 1001-1013, Sep 2006.

[4] I. Hababeh, R. Omar, and B. Nicholas, "A high-performance computing method for data allocation in distributed database systems," Journal of Supercomputing, Vol. 39 Issue 1, pp. 3-18, Jan 2007.

[5] Y. Jiang and J. Jiang, "Contextual resource negotiation-based task allocation and load balancing in complex software systems," IEEE Transactions on Parallel & Distributed Systems., Vol. 20 Issue 5, pp. 641-653, May 2009.

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

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

[8] M. Lin, "An optimal workload-based allocation approach for multidisk databases," Data & Knowledge Engineering, Vol. 68 Issue 5, pp. 499-508, May 2009.

[9] S. March and S. Rho, "Allocating data and operations to nodes in distributed database design," IEEE Transactions on Knowledge and Data Engineering, Vol. 7(2), pp. 305–317, Feb 1995

[10] S. Menon, "Allocating fragments in distributed databases," IEEE Transactions on Parallel & Distributed Systems, Vol. 16(7), pp. 577–585, July 2005

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

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

[13] J. Potts, T. Giddens, and S. Yadav. "The development and evaluation of an improved genetic algorithm based on migration and artificial selection." IEEE Transactions on Systems, Man, and Cybernetics, Vol. 24, pp. 73-86, January 1994.

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

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

[16] A.Tamhankar and S. Ram, "Database fragmentation and allocation: an integrated methodology and case study," IEEE Transactions on Systems, Man, and Cybernetics: Part A., Vol. 28 Issue 3, pp. 288-295, May 1998.

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

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

[19] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*

[20] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[21] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[22] K. Elissa, "Title of paper if known," unpublished.

[23] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[24] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[25] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.