STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIX**, Number 1, 2014

DATA TRANSFER OPTIMIZATION IN DISTRIBUTED DATABASE QUERY PROCESSING

LEON ŢÂMBULEA, ADRIAN SERGIU DĂRĂBANT, AND VIORICA VARGA

Abstract. Query execution in a distributed database requires data transfers between the processing nodes of the system. An important step in the query optimization is the minimization of the data transfers which often incur larger latencies than local data processing. In this paper we propose a new method and algorithms for determining the processing nodes for the evaluation of each relational operator of a query so that data transfer is minimal. We model our method as a data transfer minimal cost problem.

1. Introduction

Distributed database system design and query processing is an active research area. The main topics are fragmentation, the allocation of the fragments to various sites, generation of subqueries for sites. Query processing includes designing algorithms that analyze queries and convert the queries into a set of data manipulation operations. An important aspect of query processing is query optimization. The distributed query optimization problem is NP-hard ([10]), which makes finding efficient solution methods and effective heuristics a high priority. Good surveys on query optimization can be found in [5], [11], [20] and [8].

The primary task of query processing is to find a strategy for executing each query over the network in the most efficient way. Query processing takes into consideration the distribution of the data, the communication cost, and the lack of sufficient locally available information. Query optimization refers to the process of ensuring that either the total cost or the total response time for a query is minimized. The choices to be made by a query optimizer sub-module of the query processor include: the order of executing relational operations; the access methods for the relating relations; the algorithms for carrying out the relational algebra operations; the order of data movements

Received by the editors: April 4, 2014.

2010 *Mathematics Subject Classification.* 68P15.

1998 *CR Categories and Descriptors.* H.2.4 [**D**atabase Management]: Systems – Distributed Databases.

Key words and phrases. distributed databases, query optimization, data transfer cost.

between sites. A good measure of resource consumption is the total cost that will be incurred in processing the query. In a distributed database system, the total cost to be minimized usually includes: CPU, I/O and communication costs.

Query optimization and the selection of join order are critical to the performance of practical relational database management systems. A query optimizer selects among the many alternative query execution plans, the one with the least estimated execution cost, according to a given cost function. The objective functions of query optimization may take many different forms. One may try to find a query evaluation plan that optimizes the most relevant performance measures, such as the response time, CPU, I/O, and network time and efforts, memory or storage costs, resources usage. The complexity of query optimization is basically determined by the number of alternative query execution plans, which grows exponentially with the number of relations involved in the query. Consequently, enumerative optimization strategies are prohibitively expensive and therefore unacceptable as the query sizes grow. Moreover, a database management system usually supports a variety of join algorithms for processing joins and a variety of indices for accessing individual relations, which increase the complexity. All query optimization algorithms primarily deal with the join queries.

Let's consider a distributed database with data and processing on m nodes $S = \{s_i | i = \overline{1,m}\}$. On each node of the system there is a subset of the data that is subject to query and update operations. The database is composed of a set of n fragments $F = \{f_j | j = \overline{1,n}\}$. A given fragment $f \in F$ has a size $dim(f)$ (given in bytes, pages, etc). Let $S(f)$ be the nodes where fragment $f \in F$ is stored and $F(s)$ the fragments stored on node $s \in S$.

A set Q of operations needs to be executed against the database in a given time interval. We assume for simplicity that most operations are read-queries. In order to execute a query $q \in Q$, the system generates an execution plan that can be represented as a tree. The leaf nodes are actual fragments of the distributed database, while internal nodes correspond to relational operators (unary or binary) to be evaluated.

A query execution needs data access to different fragments and intermediary results obtained by evaluating relational operators of the query. These intermediary results need to be transferred between processing nodes of the system. The fragment allocation order could have a large impact on the query execution time. Various fragment allocation methods are described in [2, 4, 6, 9, 12, 14, 15, 16, 17, 19].

During each query execution the system gathers various statistical information about the evaluation of relational operators in the execution plans. Using statistics and knowledge about data distribution, one could generally

propose an optimal query execution plan that minimizes the cost of inter-node data transfer.

The remaining of this paper is organized as follows. Section 2 presents relevant research on this subject, Section 3 describes the various information an evaluation engine could obtain while evaluating execution plans. In section 4 we provide an algorithm for finding the optimal node where each operator needs to be evaluated in order to minimize the cost of inter-node data transfer, for a given fragment allocation. In the last section we present the conclusions and future developments.

2. Related work

Many algorithms have been proposed for various aspects of query optimization, fragmentation, data and operation allocation. These algorithms may be divided into three major categories: deterministic search algorithms, randomized algorithms and genetic algorithms. The most prevalent technique in first category is dynamic programming, as used in System R [10]. The algorithm exhaustively searches through all plans for the query and prunes away bad sub-plans as early as possible. This algorithm is not time efficient for large number of joins and relations as it is exponential. There are many variations of this classical algorithm. Randomized algorithms make random choices as they walk thru the state space to find a local minima. The most successful of these algorithms called Two Phase Optimization [7] combines iterative improvement (a variant of hill climbing) with simulated annealing. The query optimization problem is a difficult combinatorial optimization problem with complicated objective functions. Genetic algorithms may be very effectively applied to search for solutions in the query optimization problem with a very large number of relations, see [21], [2].

Apers has discussed in detail the data allocation problem and their fragmentation in [1]. An heuristic algorithm for redistributing the fragments is proposed in [16]. The algorithm minimizes the size of the data transferred for solving a request. Assuming that a distribution of the fragments in the nodes of a network is known, the algorithm generates a plan to transfer data fragments, plan that will be used to evaluate a request. Other fragment allocation methods are described in [2, 4, 6, 9, 12, 14, 15, 16, 19].

3. Query Evaluation

Before actual execution and data retrieval for a given query q, the system generates a query execution plan. The execution plan decomposes q in a set of sub-queries, each sub-query corresponding to an evaluation algorithm for an operator from the relational algebra. Generally, and in our approach, the

execution plan is represented as a tree where leaf nodes are fragments and internal nodes are relational operators. Some of these operators need a single argument (unary - projection, selection), while others need two arguments (binary - join). The operator arguments in various phases of the execution are either fragments or the results of evaluating some other operators.

For a given query q we can determine a cost for the evaluation of all its sub-queries, a cost for transferring the required data between the nodes where sub-queries are executed and a cost for transferring the intermediate results from the where the root operator from the query execution plan is evaluated to the node where the final result of q is needed [5].

Let c_{ij} be the cost for transferring a data unit from a given node s_i to a different node s_j. When transferring an entire fragment, table or intermediary result T from s_i to s_j the actual transfer cost is given by $c_{ij} \times dim(T)$. We can assume without any over-simplifying that the transfer costs are constant $c_{ij} = 1$ if $i \neq j, c_{ii} = 0, 1 \leq i, j \leq m$.

The execution of a query q implies the postorder traversal of the associated execution plan (tree) and the evaluation of the relational operators from the internal nodes. Each operator is evaluated on a node of the system and implies a potential data transfer from other nodes of the system where its arguments are evaluated. After the operator in an internal node is evaluated we obtain a result of a given size that is independent of the nodes that provide the argument data. In order to assess the computation of the data transfer cost when executing a query q, we associate a *tag* to each node of the execution plan $(node, dim, ct)$, where:

- **node** - is the node where the fragment is stored (if a leaf), or where the relational operator is evaluated (if an internal node);
- **dim** - the size if the fragment or intermediate results;
- **ct** - the cost of data transfer required to evaluated the current operator. It is obtained as the sum of data transfer cost for the arguments of the current node and the required costs for the evaluation of these arguments at their origin.

For a leaf node, corresponding to the fragment f, the *tag* is $(node, dim, cost)$, where dim is the fragment size and $ct = 0$ if $node \in S(f)$, or $ct = dim$ if $node \notin S(f)$.

In the following paragraphs we introduce the *tag* computation methods for a binary and unary operators.

Let Θ_1 be an unary relational operator. Figure 1 shows the computation values involved in the computation of the *tag* for the Θ_1 node.

Using the *tag* for node A we obtain for Θ_1:

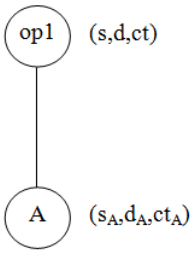

FIGURE 1. Unary operator tag

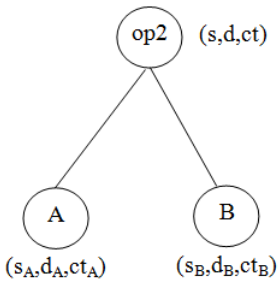

FIGURE 2. Binary operator tag

$$ct_{\Theta_1} = \begin{cases} ct_A & \text{if } s = s_A \\ ct_A + d_A & \text{if } s \neq s_A \end{cases}$$

The case of a binary operator is depicted in Figure 2:

if $s_A = s_B$ then: $ct_{\Theta_2} = \begin{cases} ct_A + ct_B, & \text{if } s = s_A \\ (ct_A + ct_B) + (a_A + d_B), & \text{if } s \neq s_A \end{cases}$

otherwise if $s_A \neq s_B$, then: $ct_{\Theta_2} = \begin{cases} (ct_A + ct_B) + d_B, & \text{if } s = s_A, \\ (ct_A + ct_B) + d_A, & \text{if } s = s_B, \\ (ct_A + ct_B) + (d_A + d_B), & \text{if } s \notin s_A, s_B \end{cases}$

A and B are the relations in our database fragmented horizontally as follows:

$$A = A_1 \cup A_2; \quad B = B_1 \cup B_2$$

Let us consider the following query:

$$q = A \times \sigma_C(B) \tag{1}$$

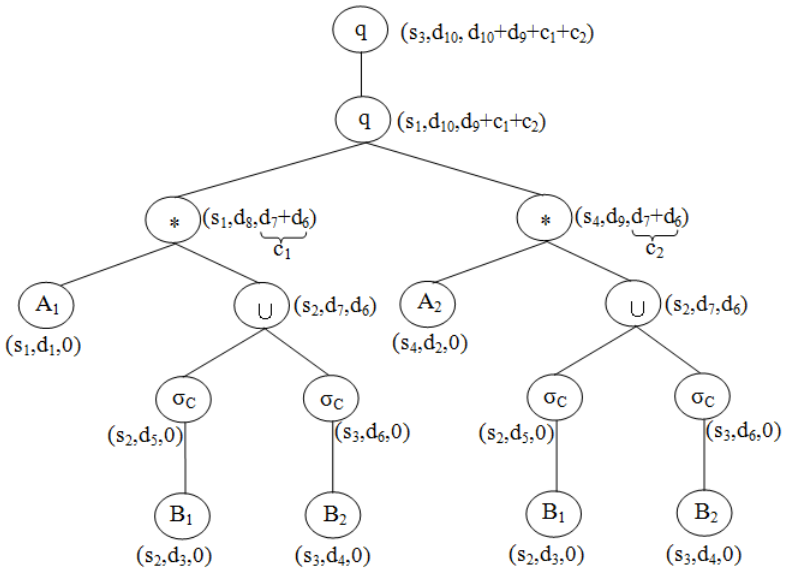

FIGURE 3. Node tags for query q.

where "$\times$" is the join operator and σ_C is a selection operator over C. This query can be transformed as follows:

$$\begin{aligned} q &= A \times \sigma_C(B) = \\ &= (A_1 \cup A_2) \times \sigma_C(B_1 \cup B_2) = \\ &= [A_1 \times \sigma_C(B_1 \cup B_2)] \cup [A_2 \times \sigma_C(B_1 \cup B_2)] = \\ &= [A_1 \times (\sigma_C(B_1) \cup \sigma_C(B_2))] \cup [A_2 \times (\sigma_C(B_1) \cup \sigma_C(B_2))] \end{aligned}$$

Suppose the database fragments are stored on four nodes:

$$\begin{aligned} &F(s_1) = \{A_1, B_1\}; F(s_2) = \{B_1\}; \\ &F(s_3) = \{B_2\}; F(s_4) = \{A_2\}. \end{aligned} \tag{2}$$

Supposing that the results for the query q needs to be returned on s_3, figure 3 presents the tags that appear in the nodes of the execution plan.

The figure also presents an additional node corresponding to the entire query q. Node tags (in the tree) correspond to a possible execution plan and allows to infer the data transfer costs. For each node of the execution plan a database node has been chosen for evaluating its operator.

We note here that the size of the fragments and intermediate results does not change if the operator evaluation is done on different nodes of the database system. In the next section we propose a method for finding the database node (station) where each operator needs to be evaluated such that the total data transfer cost be minimal.

4. Finding the query evaluation plan with minimal data transfer cost

For each node in the query evaluation tree, with its attached tags (like in figure 3) we add an additional root node corresponding for the whole query result and we determine the following values:

- $\underline{\mathrm{d}}$ - size of data associated with a node;
- A vector $\underline{c} = [c_1, \ldots, c_m]$, where m is the number of stations composing the distributed database. A component of the vector c_i has the following meaning:
  - for a leaf node,corresponding to a fragment f, is the cost of data transfer for the given fragment to station (node) s_i;
  - for an internal node, corresponding to a relational operator, is the *minimum data transfer cost* if the evaluation of this operator takes place on s_i;
- $\underline{\mathrm{sr}}$ - a database station where is recommended to evaluate the current operator (for an internal node of the execution plan), or where the fragment needs to be read (for a terminal node).

In the first step of our method we compute the vector $\underline{\mathrm{c}}$ for each node of the query evaluation plan using a post-order traversal order. For a given current node, the computation of the $\underline{\mathrm{c}}$ uses the values of $\underline{\mathrm{d}}$ and $\underline{\mathrm{c}}$ associated with the operands nodes. If the current node is a leaf node, corresponding to a fragment f, then: $c_i = \begin{cases} 0, \text{ if } s_i \in S(f), \\ dim(f), \text{otherwise.} \end{cases}$

The second step starts from the root of the query evaluation plan. The network station where the response of the query q needs to be returned will be the $\underline{\mathrm{sr}}$ value associated to this node. For a given current node, starting from the root, we will compute the values $\underline{\mathrm{sr}}$ for the associated operand nodes.

There are two ways for computing the values of the $\underline{\mathrm{c}}$ vector for the current node depending if the associated operator is binary or unary.

Let Θ_1 be an unary operator. The c_i values $i = \overline{1, m}$ are the minimum data transfer costs if Θ_1 is evaluated on station s_i. Figure 4 shows the construction of the $\underline{\mathrm{c}}$ vector for an unary operator. If the argument (operand) A is evaluated (or stored) in a station $s_j, i \neq j$, then overall cost of evaluating the node A on s_j is incremented with d_A, the cost of data transfer from station s_j to station

s_i. The total transfer cost for A becomes thus $c_j^A + d_A$. If $i = j$, then the cost for the node A is c_i^A. In the general case, the formula for a node is given by:

$$c_i = \min\left\{c_1^A + d_A, \ldots, c_{i-1}^A + d_A, c_i^A, c_{i+1}^A + d_A, \ldots, c_m^A + d_A\right\} \tag{3}$$

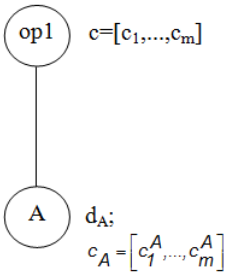

FIGURE 4. Cost vector attached to an unary operator node.

For a binary operator we use a similar computation method as for the unary operator. Figure 5 shows the intervening values and their computation is explained bellow.

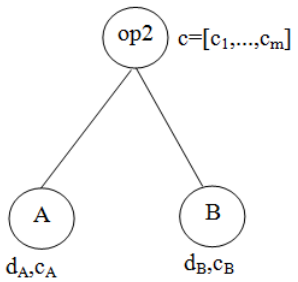

FIGURE 5. Cost vector attached to a binary operator node.

The value of the $c_i, i = \overline{1,m}$ components in the case of a binary operator Θ_2 are the minimum transfer costs if the operator is evaluated on station s_i. They are obtained as follows:

$$\begin{aligned} c_i = &\min\left\{c_1^A + d_A, \ldots, c_{i-1}^A + d_A, c_i^A, c_{i+1}^A + d_A, \ldots, c_m^A + d_A\right\} + \\ &+ \min\left\{c_1^B + d_B, \ldots, c_{i-1}^B + d_B, c_i^B, c_{i+1}^B + d_B, \ldots, c_m^B + d_B\right\} \end{aligned} \tag{4}$$

In the following we assume that the value of the $\underline{\mathrm{d}}$ and the components of the $\underline{\mathrm{c}}$ vector have been computed for a given query q on all nodes of the query

evaluation plan (tree). The value for $\underline{d}$ is usually obtained from statistics and estimations stored in the database dictionary.

Let c_q be the vector associated with query q, represented in the evaluation plan as the root node (or node q). The value sr associated to node q is given by the station sq where we need to return the result of the query execution, so $sr_q = s_q$. The value c^q_{sq} from the $c_q = \left[c^1_1, \ldots, c^q_m\right]$ vector is the minimum data transfer cost if q is evaluated on station sq. This value is computed according to equation 3 using the operand node of q like depicted in figure 4.

We start with this initial value for the root node and $\underline{sr}$. We traverse the nodes of the evaluation tree from the root node to the leafs (this can be a pre-order traversal) and compute the values for $\underline{sr}$ for each operand (argument) of a given current node.

For an unary operator Θ_1, as in figure 4, the values for c and sr are known. This operator needs to be evaluated on station s_{sr}. The c_{sr} value is the minimum data transfer cost if Θ_1 is evaluated on station s_{sr} and is computed according to equation 3. The minimum value (according to equation 3) could be obtained on multiple stations. For simplicity, in the case we obtain the minimum value on multiple stations we choose one (any one) station that achieves the minimum. Let this station be sr_A - the station attached to node A. Figure 6 show a numerical example for c computation on unary operators.

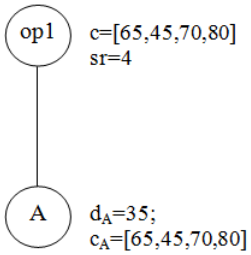

FIGURE 6. Example: Cost vector computed for an unary operator.

$sr = 4 \Rightarrow c_4 = 80 = \min\{65+35, \underline{45+35}, 70+35, \underline{80}\} \Rightarrow sr_A \in \{2,4\}$. Stations s_2 and s_4 are the stations that incur the lowest data transfer cost when evaluating Θ_1.

For a binary operator node, like in figure 5, we can compute the values of sr_A and sr_B using the Θ_2's sr value and sizes of operands A and B. Figure 7 shows a numerical example for computing these values.

$$\begin{aligned} sr = 4 \rightarrow d_4 = 80 \quad &= \quad \min\{50+30, 25+30, 40+30, \underline{30}\} + \\ &+ \quad \min\{\underline{15+35}, 20+35, 30+35, \underline{50}\}. \end{aligned}$$

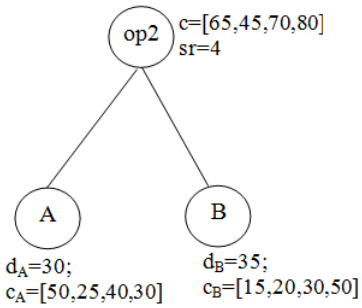

FIGURE 7. Example: Cost vector computed for a binary operator.

We obtain thus: $sr_A = 4, sr_B \in \{1, 4\}$.

5. Experiments and numerical test

In the following, we are going to compute the tags associated with the evaluation plan for the query expressed in equation 1, on the evaluation tree presented in figure 3. Using actual values for fragment sizes and intermediate results, we traverse in the first step the tree in post-order, from the leafs to the root in order to compute the $\underline{c}$ values on each node. The second traversal of the evaluation tree, from the root to the leafs (pre-order) is used to compute the $\underline{sr}$ values, and thus the stations where each query operator needs to be evaluated in order to minimize the data transfer cost. The obtained results are depicted in figure 8.

The $\underline{c}$ values for the leaf nodes (fragments) are computed from the the information on the fragment allocation to database stations given by the equation 2.

6. Conclusions and future work

In this paper we propose a cost based optimization of a query in a distributed relational database, by applying some metrics to its evaluation tree. We start from the general assumption that the evaluation plan is represented as a tree with the leaf nodes storing the actual database fragments and the internal nodes representing operators of the query. Once a candidate evaluation plan is determined, our method tries to minimize the cost of data transfers when executing the query against the real database by dynamically computing the stations of the system that are optimal for each query operator evaluation. We also take in account the data transfer to the station where the results needs to be obtained. The proposed method mathematically chooses the optimal node for each query operator evaluation, based on some usual statistical

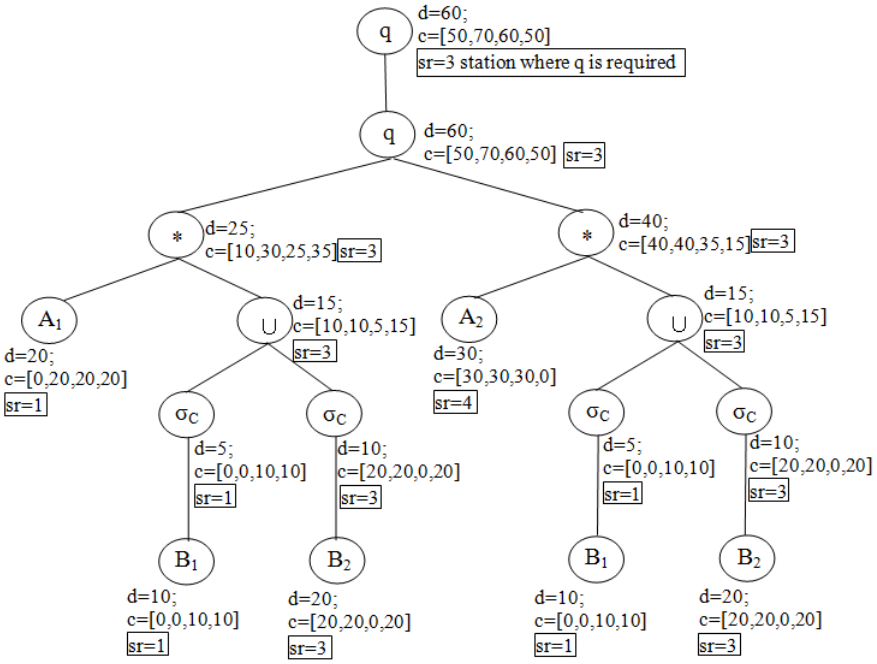

FIGURE 8. Evaluation of the query in equation 1

database information and by attaching *tags* to operator nodes such that overall data transfer cost is minimal. The algorithm uses two traversals of the query evaluation tree in order to extract the *operator-to-station* execution affinity. We aim to use the results of the proposed method to propose dynamic re-fragmentation of the database or data replications in order to minimize data transfer and processing costs in distributed databases.

REFERENCES

[1] P.M.G. Apers, *Data Allocation in Distributed Database Systems*, ACM Trans. on Database Systems, 13(3), (1988), pp. 263-304.

[2] C.H. Cheng, W.K. Lee, K.F. Wong, *A Genetic Algorithm-Based Clustering Approach for Database Partitioning*, IEEE Transactions on Systems Man and Cybernetics Part C-Applications and Reviews, 32, (2002), pp. 215-230.

[3] R. Diestel, *Graph Theory*, Springer-Verlag, Heidelberg 2000, Electronic Edition.

[4] J. Graham, *Efficient Allocation in Distributed Object Oriented Databases*, Proceedings of the ISCA 16th International Conference on parallel and Distributed Computing Systems, Reno Nevada, August (2003), pp. 407-412.

[5] G. Graefe, *Query Evalution Techniques for a Large Database*, ACM Computing Surveys, 25, (1993) pp. 73-90.

[6] Y. Huang, J. Chen, *Fragment Allocation in Distributed Database Design*, Journal Of Information Science And Engineering, 17, (2001) pp. 491-506 .
[7] Yannis E. Ioannidis, Youngkyung Cha Kang, *Randomized Algorithm for Optimizing Large Join Queries*. Procedings of ACM SIGMOD International Conference on Management Of Data, (1990), pp. 312-321 .
[8] D. Kossmann, *The State of the Art in Distributed Query Processing*, ACM Computing Surveys, Vol. 32, Issue 4, (2000) pp. 422-469.
[9] S.T. March, S. Rho, *Allocating Data and Operations to nodes in a distributed database design*, IEEE Trans. on Knowledge and Data Engineering, Vol. 7 (2), (1995), pp. 305-317.
[10] M. T. Ozsu, P. Valduriez, *Principles of Distributed Database Systems*, 2nd ed., Prentice-Hall International Editions, 1999.
[11] M. S. Sacco, S. B. Yao: *Query Optimization in Distributed Database Systems*, Advances in Computers, Vol. 21, Academic Press, New York, (1982), pp. 225-273.
[12] Rajinder Singh, Gurvinder Singh, Varinder Pannu, *Optimized Access Strategies for a Distributed Database Design*, International Journal of Data Engineering (IJDE), Vol. 2, Issue 3, (2011), pp. 102-110, .
[13] Rajinder Singh Virk, Gurvinder Singh, *Optimizing Access Strategies for a Distributed Database using Genetic Fragmentation*, International Journal of Computer Science and Network Security, VOL.11 No.6,(2011), pp. 180-183.
[14] A. Sleit, W. AlMobaideen, S. Al-Areqi, A. Yahya, *A Dynamic Object Fragmentation and Replication Algorithm in Distributed Database Systems*, American Journal of Applied Sciences 4 (8), (2007) pp. 613-618.
[15] L. Tambulea, M. Horvat, *Dynamic Distribution Model in Distributed Database*, Int. J. of Computers, Communications & Control, Vol. III (2008), Suppl. issue: Proceedings of ICCCC (2008), pp. 512-515.
[16] L. Tambulea, M. Horvat, *Redistributing fragments into a distributed database*, Int. J. of Computers, Communications & Control, Vol. III (2008), No. 4, pp. 384-394.
[17] T. Ulus, M. Uysal, *Heuristic Approach to Dynamic Data Allocation in Distributed Database Systems*, Pakistan Journal of Information and Technology 2 (3),(2003) pp. 231-239.
[18] S. Upadhyaya, S. Lata, *Task allocation in Distributed computing VS distributed database systems: A Comparative study*, IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.3, (2008), pp. 338-346.
[19] O. Wolfson, S. Jajodia, *An Algorithm for Dynamic Data Distribution*, Proceedings of the 2nd Workshop on the Management of Replicated Data (WMRD-II), Monterey, CA, (1992), pp. 62-65.
[20] C.T. Yu, C.C. Chang, *Distributed Query Processing*, ACM Computing Surveys, Vol. 16, (1984), pp. 399-433.
[21] Zehai Zhou, *Using Heuristics and Genetic Algorithms for Large Scale Database Query Optimization*, Journal of Information and Computing Science Vol. 2, No. 4, (2007), pp. 261-280

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Kogălniceanu 1, 400084 Cluj-Napoca, Romania
E-mail address: leon@cs.ubbcluj.ro
E-mail address: dadi@cs.ubbcluj.ro
E-mail address: ivarga@cs.ubbcluj.ro