Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. III (2008), No. 4, pp. 384-394

Redistributing Fragments into a Distributed Database

Leon Ţâmbulea, Manuela Horvat-Petrescu

Abstract: A distributed system database performance is strongly related to the fragment allocation in the nodes of the network. An heuristic algorithm for redistributing the fragments is proposed. The algorithm uses the statistical information relative to the requests send to a distributed database. This 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.
Keywords: distributed database, fragment allocation, allocation algorithm, transfer cost, heuristic algorithm, redistribution algorithm

1 Introduction

Let's consider a distributed database C, formed by n nodes (sites) S_i, $0 <= i < m$. Each node contains a local database and has the capability to evaluate requests. The distributed database C contains a set of n fragments F_j, $0 <= j < n$. Each F_j fragment has a specific dimension $dim(F_j)$ - the dimension can be measured in bytes, pages, so on. Different fragments noted with L_j can be stored in a S_i node.

Let's assume that a user is sending a request q (a select, an update, a query). For solving this request, are necessary a set of fragments $r(q)$ and for each fragment an access right (read/write - depending of the request) is required.

The great majority of requests received by a distributed database are requests used for data retrieving. The fragments that should be transferred between the nodes of the network are required when evaluating a request. An optimal allocation of the fragments in the nodes of the network is necessary for the request evaluation time to be minimum. In [1, 3, 4, 7, 9, 10, 11] are mentioned other solutions for solving the allocation problem. Because of the complexity of this problem (NP-complete problem) more heuristic algorithms were proposed, algorithms with a lower complexity that provide only an approximate solution. In [8] is described a model that propose a dynamic redistribution of the fragments using the statistical information gathered in a specific period of time. Queries and information about the performed fragment transfer can be obtained for a specific period of time in a distributed database. Using this data obtained over a longer period of time, a redistribution of the fragments can be performed for minimizing the size of the data transfer.

This article is organized as follows: In section 2 is described the model for evaluating a query, and also the useful information that can be obtained when evaluating this query. The third section describes the problem of redistributing the fragments in the nodes of a distributed database. For solving this problem is proposed an heuristic algorithm that minimizes the size of the data transferred between the network nodes. In section 4 is proposed an algorithm for generating a transfer plan of the fragments when evaluating a query. This transfer plan is obtained using the query evaluation plan and the fragments distribution in the nodes of the network. In the final section of the paper is presented a conclusion section.

2 Query Evaluation

An execution plan can be determined for each request q. In centralized databases, lots of studies regarding how to find an execution plan with a minimum cost were performed, and some results were implemented in commercial database management systems.

Using the execution plan, a request q can be split in a number of sub requests $\{ q_i, i \in I_q\}$, and each sub request q_i corresponds to an operator in the relational algebra. An operator requires one fragment (if it's an unary operator) or two fragments (if is a binary operator). These arguments (fragments) can be stored in a node in the distributed database or can be the result of evaluating other operators.

In a distributed system, a request q can be evaluated as follows [5, 6]:

- Centralized, in a specific node S from the network. All the fragments $r(q)$ necessary for solving the request q should be transferred in this node S. The node S in which the evaluation is performed can be determined as the total fragments $r(q)$ transfer cost to be minimum.
- Distributed: each sub request q_i is evaluated in a separate node S_j in the network and in the S_j node should be found only the $r(q_i)$ fragments. The $r(q_i)$ fragments can be fragments stored in the S_j node, can be the result of a previous evaluation in this node, or can be transferred from other node of the network.

In order to evaluate the cost of a request q, there has to be evaluated the cost of the operators and the cost of the fragments transfer to the nodes where these operators are evaluated ([5, 6]). In this section as in the next one, we will analyze the fragment redistribution problem in the nodes of a distributed database in order to obtain a minimum cost for data fragments transfer when evaluating requests.

Let's note with $c_{i,j}$ the cost of the transferring a data unit from the node S_i to the node S_j. For a fragment F (that is stored in the node or is a fragment resulted from evaluating previous requests) transferred from node S_i to the node S_j the total cost is $c_{ij} * dim(F)$. Because the c_{ij} cost is relative but the difference between them (in absolute terms) is relatively small, we'll suppose in this paper that they are constants; $c_{i,j} = 1, 0 <= i, j < m$. This assumption simplifies the solution for the proposed problem.

A request q can be split into elementary queries (operations) that execute over different fragments stored in the node N. If such an elementary request (a join operator for example) uses fragments that are not stored in the same node, than all the fragments should be transferred to the same node.

In the following example, the distributed database is composed by two relations: A and B, and has horizontal fragmentation, such as:

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

The request/query q for this database is:

$$q = A * \sigma_c(B)$$

where c is a condition and "*" represents a join operator.

If all the fragments are stored in the same node than the request q can be evaluated in the node without requiring other data transfers. If all the fragments (A_1, A_2, B_1 and B_2) are stored in different nodes (noted with S_0, S_1, S_2 and S_3) then evaluation of the request q implies a data transfer cost. We'll consider that the requests q is required in a node S different from the previous mentioned nodes (S_0, S_1, S_2 and S_3) where the data fragments are stored. Two evaluation strategies can be considered:

1. The evaluation is performed in the node S, so each required fragment is transferred from the node where is stored into the S node. The size of the whole transfer is:

$$dim(A_1) + dim(A_2) + dim(B_1) + dim(B_2).$$

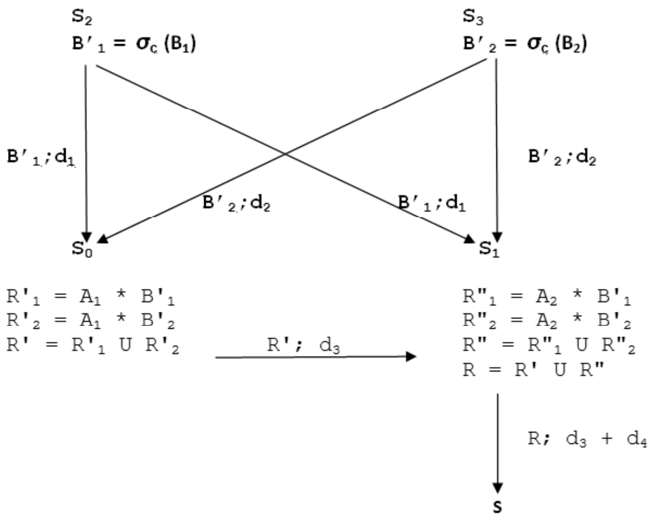
2. The evaluation is performed in a distributed manner: for this case the request q can be transformed as follows:

$$q = A * \sigma_c(B) = (A_1 \cup A_2)* \sigma_c(B_1 \cup B_2) =$$
$$= [A_1 * \sigma_c(B_1 \cup B_2)] \cup [A_2 * \sigma_c(B_1 \cup B_2] =$$
$$= [A_1 * (\sigma_c(B_1) \cup \sigma_c(B_2))] \cup [A_2 * (\sigma_c(B_1) \cup \sigma_c(B_2))]$$

The request *q* can be evaluated using the following graph, where we consider that the result of evaluating $\sigma_c(B_i)$ will have a size/dimension d_i smaller that $dim(B_i)$, $i=1,2$.

Let's note with $B'_i = \sigma_c(B_i)$, i=1,2 the results of the selection from the S_2 and S_3 nodes and $d_3 = dim(B'_1)$, $d_4 = dim(B'_2)$.

In the nodes of the following graph (the nodes of the graph represent the nodes of the network) are presented the operators (unary and binary) that are evaluated and on the links are shown the data transfers required in the evaluation process.

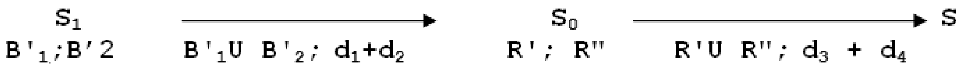


The total cost of the transfer will be $2(d_1 + d_2 + d_3) + d_4$.

In a distributed database the fragments can be replicated - a fragment can be stored in more than one site/node. For the distributed database presented above, we'll exemplify for two fragment distribution plans the corresponding execution plan.

3.

Node	S_0	S_1	S_2	S_3
Fragments	A_1 , A_2	B_1 , B_2	A_1 , B_1	A_2 , B_2



4.

Node	S_0	S_1	S_2	S_3
Fragments	A_1 , B_1	A_1 , B_2	A_2 , B_1	A_2 , B_2

$$\begin{tikzcd}
S_1 \arrow[d, "A_1*B'_2;\ d_5"] & S_3 \arrow[d, "A_2*B'_2;\ d_6"] \\
S_0 \arrow[r, "R';d3"'] & S_2 \arrow[d, "R'U\ R'';\ d_3 + d_4"] \\
 & S
\end{tikzcd}
\qquad
\begin{tikzcd}
S_0 \arrow[d, "A_1*B'_1;\ d_7"] & S_2 \arrow[d, "A_2*B'_2;\ d_8"] \\
S_1 \arrow[r, "R';d3"'] & S_3 \arrow[d, "R'U\ R'';\ d_3 + d_4"] \\
 & S
\end{tikzcd}$$

3 Redistribution of the fragments

Let's assume the following situation: in a distributed database for a specific period of time *T* is required to obtain the answers to a set of requests *Q*. The problem raised is the redistribution of the fragments as the size of the data transfers between the nodes for the same set of requests Q (if they are repeated) to be decreased. The size of the transfer will be zero if the database in completed distributed (so each fragment is stored in each node).

We'll assume that the size of each node in the network is limited by a maximum value noted with *DMax*, and the number of the replica r_j for a fragment F_j is between two limits:
$1 <= r_j <= RMax_j$, $0 <= j < n$.

The problem of optimal redistribution of the fragments is quite difficult as the optimization performed by the query optimizer must be taken into account and also must be taken into account the size of the data transfer between the nodes. Due to the complexity of this problem, more heuristic algorithms were proposed (for example [1, 3, 4, 7, 9, 10, 11]). We describe a new and much simpler algorithm which offers an approximate solution for the proposed problem.

The result of generating an execution plan for a query q is a list of operators that should be evaluated: $op_1(X)$ - for an unary operator or $op_2(X, Y)$ in case of binary operators, where *X* and *Y* are fragments stored in the database or are the result of evaluating previous operators. If is an unary operator, then the evaluation can be performed in the node where the operand is stored and no data transfer is required. In case of a binary operator $op_2(X, Y)$, the evaluation can be performed in the node where the *X* fragment is stored or is determined, or where the *Y* fragment is stored or is determined. The node is chosen in order to minimize the data transfer size.

From the information obtained when evaluating the set of requests *Q*, in a fragment redistribution is used only the information related to the data transfer: "the fragment *X* (or the fragment obtained from an evaluation in a node where the fragment *X* is stored) is transferred in a node where is stored the fragment *Y* (or the fragment obtained from an evaluation in a node where the fragment *Y* is stored)".

The data transfers occur only when evaluating binary operators. For example, for the evaluation strategy *b* from the previous section the following transfers occur:

Binary operator	Data Fragment	Observations
$R'_1 = A_1 * B'_1$	B_1, A_1, d_1	The meaning of the first line is: d_1 is transferred from a node where B_1 is stored to a node where A_1 is stored
$R'_2 = A_1 * B'_2$	B_2, A_1, d_1	
$R''_1 = A_2 * B'_1$	B_1, A_2, d_1	
$R''_2 = A_2 * B'_2$	B_2, A_2, d_2	
$R' = R'_1 \cup R''_2$	A_1, A_1, d_3	Does not require transfer
$R'' = R''_1 \cup R''_2$	A_2, A_2, d_4	Does not require transfer
$R = R' \cup R''$	A_1, A_2, d_3	

The final result where the fragment A_2 with size $d_3 + d_4$ is stored must be transferred to the node S.

For a set of requests Q that were evaluated in a period of time, a journal with two types of information can be created:

1. A set of tuples: $T_1 = \{ (F_i, F_j, d_i), i \in I \}$, where a tuple $t = (F_i, F_j, d_i)$ has the meaning that from a node that stores the F_i fragment, some data with size d_i is transferred in a node where the fragment F_j is stored.
2. A set of tuples: $T_2 = \{ (F_q, S_q, d_q), q \in Q\}$, where a tuple $t = (F_q, S_q, d_q)$ corresponds to a query $q \in Q$ and has the following meaning: from the node where the fragment F_q is stored (and where the query evaluation was finalized) some data with size d_q are transferred in the node S_q where the result for the query q is required.

The algorithm has two phases:

First phase: the distribution of the fragments in the virtual nodes is performed such as the total size of the data transfer to be minimum.

Second phase: the m virtual nodes created in the first phase are associated with the real nodes in such a manner as the size of the data transfer involved in answering the queries to be minimum.

A pair of fragments F_i and F_j can be found in more than one tuple from T_1 set, and can correspond to a set of binary operators required by the queries from the set Q. If these two fragments are stored in the same node, than there is no data transfer for neither one of the binary operators. In the proposed algorithm, a matrix $V = (v_{i,j})$, $0 <= i, j <n$ will be determined. The $v_{i,j}$ represents the size/dimension of the data fragments that don't have to be transferred when evaluating the queries from Q in the case when the fragments F_i and F_j are stored in the same node.

The V matrix can be easily determined using the following algorithm:

1. $v_{i,j} = 0, 0 <= i, j <n;$
2. *For each* $t = (F_i, F_j, d_i) \in T_1$: $v_{i,j} = v_{i,j} + d_i;$
3. *For each* $0 <= i <n, 0 <= j <i$: $w = v_{i,j} + v_{j,i}$; $v_{i,j} = w, v_{j,i} = w$

The V matrix is symmetrical and the values have the mentioned significance. The values: $v_{i,j}$, $0 <= j <= i <n$ represent the inferior block of the matrix V.

These values will be sorted in a descending order and will be used (in this order) to generate a new data fragments distribution in the distributed database.

The proposed algorithm is given in the next paragraph and contains four phases:

1. For each node S_i , $0 <= i <m$ the following initializations will be performed:

   (a) $s_i = 0$; *(the size/dimension used by the node S_i)*

   (b) $L_i = \Phi$; *(the set of the fragments stored in the node S_i)*

   (c) For each fragment F_j, $0 <= j <n$: $r_j = 0$; (the number of replicas of the fragment F_j)

2. The values from the V matrix are retrieved in a descending order. A tuple $t = (F_i, F_j, v_{i,j})$ corresponding to a value $v_{i,j}$ from the V matrix represents a gain obtained for the total transfer size/dimension if the fragments F_i and F_j are stored in the same node. For this tuple t will be determined the node that offers the best advantage in the case that the mentioned fragments will be added there. For finding this node, the algorithm will compute the gain c_k obtained if these fragments are attached to the node S_k, and will keep track of an indicator sit_k. The value from sit_k is referring to the state of the fragments $\{F_i, F_j\}$ relative to the set L_k , for $0 <=k <m$. (L_k is the set of the nodes existing in S_k).

```
(a) If $F_i \in L_k$ and $F_j \in L_k$, (the fragments $F_i$, $F_j$ are already added to the node $S_k$, then:
    $c_k = 0$; $sit_k = 1$;
(b) If $F_i \in L_k$ and $F_j \notin L_k$, then:
    If $r_j < RMax_j$ and $S_k + dim(F_j) <= DMax$, then
        $sit_k = 2$, $c_k = 0$;
     For $F_p \in L_k$ : $c_k = c_k + v_{jp}$;
    Note: the fragment F$_j$ can be added to the node S$_k$ and the gain obtained will be c$_k$;
    Else $sit_k = 1$, $c_k = 0$;
(c) If $F_i \notin L_k$ and $F_j \in L_k$, then:
    If $r_i < RMax_i$ and $S_k + dim(F_i) <= DMax$, then
        $sit_k = 3$, $c_k = 0$;
        For $F_p \in L_k$ : $c_k = c_k + v_{ip}$;
    Else $sit_k = 1$, $c_k = 0$;
(d) If $F_i \notin L_k$ and $F_j \notin L_k$, then:
    If $r_i < RMax_i$ and $r_j < RMax_j$, and $S_k + dim(F_i) + dim(F_j) <= DMax$, then
        $sit_k = 4$, $c_k = v_{ij}$;
        For $F_p \in L_k$ : $c_k = c_k + v_{ip} + v_{jp}$;
    Else $sit_k = 1$, $c_k = 0$;
```

   A node S_k with c_k maxim will be determined. Depending on the value sit_k the following data are modified: L_k, S_k, r_i, r_j.

3. If a fragment F_i was not used in the binary operators obtained from Q in the period of time T used to analyze and to evaluate the requests from set Q, then the result in the precedent step will be $r_i = 0$, and that means that the fragment F_i was not stored in the nodes of the network. In this case, a node from the network with enough space to store the fragment F_i without removing other fragments will be selected.

```
(a) $sit = 0$;
(b) For each $0 <= k <n$:
    If $S_k + dim(F_i) <= DMax$, then
        $F_i$ is added to the node $S_k$, ($S_k$, $L_k$ will change; $r_i = 1$);
```

```
      sit = 1;
    Endif
```

If *ind=0* after running the previous algorithm it means that the fragment F_i could not be added to a node then other fragment $G_k \in L_k$ will be searched for. The fragment G_k is chosen as the node S_k will have the smaller lost when removing it:

```
(c) For $0 <= k < m$:
      $c_k = \infty$;
      For each $Fp \in L_k$:
          $cc = \sum v_{i,j}$; $F_j \in L_k, j \neq p$
          If $cc < c_k$ then $c_k = cc$, $G_k = F_p$
```

(d) Is determined the S_k node having a minimum value for c_k. In the S_k node the G_k fragment is replaced by the F_i fragment. The value from L_k, r_i, and r position corresponding to G_k will be updated.

4. The R_i set containing the nodes where the fragment F_i is stored is computed for each fragment F_i. (R_i is the set of the F_i replicas)

At the end of this algorithm were determined the value of L_k for each node S_k, $0 <= k < n$. The virtual nodes can be allocated to the real nodes using the information obtained from the set of requests Q. The virtual nodes can be allocated to the real nodes according to the node where evaluation is required. This allocation will be done using a second set of tuples: $T_2 = \{(F_q, S_q, d_q), q \in Q\}$. For a request q, if the fragment F_q is stored in the node S_q (S_q is the node where the answer to the request q is required) then the data with the size d_q are not transferred.

We'll assume that the virtual nodes where the fragments are allocated are S_k, $0 <= k < m$, and the real nodes from the distributed database are SR_k, $0 <= k < m$.
With $c_{i,j}$ is noted the total gain obtained when transferring the answers of the requests from Q if the virtual node S_i is stored in the real node SR_j. ($c_{i,j}$ is the size of the data from all the answers for all the requests that are transferred from S_i to SR_j.) The $c_{i,j}$ values can be computed as follows:

```
$c_{i,j} = 0$; $0 <= i, j < m$;
For each $t = (F_i, SR_j, d) \in T_2$,
    For each $S_k \in R_i$ (the nodes containing $F_i$ replicas):
        $c_{k,j} = c_{k,j} + d$
```

Is build a graph with the nodes S_k, SR_k, $0 <= k < m$, and the value $c_{i,j}$ will be associated to the line from S_i to SR_j. For this graph is determined the maximum cupling with the maximum value [2]. Using this coupling, a virtual node S_i will correspond to a real node SR_j where the L_i replicas are stored. Using this redistribution the R_i set could be recomputed.

4 Fragments transfer plan when evaluating requests

A request q is split by the execution plan in as set of operators $\{op_0, \ldots op_{n-1}\}$. The result of the request q is the result of evaluating the last operator, as the other operators generate intermediate results. Evaluation plans can be created for each operator, these plans detailing all the data transfers required in the evaluation process and the size of the transfers. If more than one evaluation plan (that evaluates in the same manner) can be created for an operator, than the plan that requires less data (in terms of size) for transfer will be used.

Coming next is an evaluation plan for a request showing the data transfers. The plan can be noted with:

$$P = (\ evaluationList;\ dim),$$

where *"evaluationList"* is a succession of actual evaluation of the operators:

$$evaluationList : (nod_i,\ op_1,\ nod_f)\ [,\ (nod_i',\ op_2,\ nod_f')]...$$

A tuple $(nodi,\ op_1,\ nod_f)$ has the following meaning: the operator *"op_1"* is evaluated in the *"nod_f"* after a data transfer from *"nod_i"*. If $nod_i = nod_f$ then there is no data transfer when the operator is evaluated (the unary operators always encounter this situation). The fragments used from the nodes nod_i and nod_f are effectively stored in the nodes or are the result of previous evaluations performed in these nodes.

"dim" is the total size/dimension of the data transferred in order to evaluate the plan from the evaluationList. This size can represent:
a) the number of the transfers involved in evaluation
b) an average size/dimension of the transferred data. The average size can be found by using values $v'_{i,j}$ computed in the same time with the values $v_{i,j}$ from section 3. ($v'_{i,j}$ would be the average size of the data transferred between the node S_i and S_j when the request set Q is evaluated.)

In the following algorithm will be used for simplicity the first meaning of the *"dim"* (explained in the paragraph a) from above).

We will note with *P(op)* the set of evaluation plans for an operator *op*. If $p = (t_0,\ t_1,\ ....\ t_{n-1};\ d) \in P(op)$ where $ti = (S^1{}_i,\ op_i,\ S^2{}_i)$, then the following notations can be used:
$dim(p) = d;$
$plan(p) = t_0,\ t_1,\ ....\ t_{n-1}$ = the transfers list required by the plan p
$result(p) = S^2{}_{n-1}$ = the node where can be found the result of the plan p evaluation

An algorithm for generating the plan *P(op)* is presented in the following paragraph (the data transfers are taken into consideration).

1. If *"op"* is an unary operator, so it has the form *op(x)* then:

   (a) If x is a fragment stored in the distributed database then
   $P(op) = \{((S,op,s);o)|S \in R(X)\}$,
   where *R(X)* represents the set of the nodes where the fragment x is stored.

   (b) If x is a fragment obtain from a previous evaluation of an operator *op'* then:
   $P(op) = \{(plan(p),(result(p),op,result(p));dim(p))|p \in P(op)\}$

2. If *"op"* is a binary operator, so it has the form *op(x,y)* then:

   (a) If x and y are fragments stored in the distributed database then:
   $P(op) = \{\ (\ (S_1,\ op,\ S_2)\ ;\ 1)$ where $S_1 \in R(X) - R(Y)\ ,\ S_2 \in R(Y)\}$
   $\cup\ \{\ (\ (S_1,\ op,\ S_2)\ ;\ 1)$ where $S_1 \in R(Y) - R(X)\ ,\ S_2 \in R(X)\}$
   $\cup\ \{\ (\ (S_1,\ op,\ S_1)\ ;\ 0)$ where $S_1 \in R(X) \cap R(Y)\}$.

   The evaluation of the operator *"op"* can be made in a node where the fragment x is stored and fragment y is transferred, or in a node where the fragment y is stored and fragment x is transferred, or in a node where both fragments are stored.

(b) If *x* is a fragment stored in the distributed database and *y* is a fragment obtained from a previous evaluation of a *op'* operator, then:

$P(op) = P_1(op) \cup P_2(op) \cup P_3(op)$ where:
$P_1(op) = \{(plan (p), (result(p), op, S); dim(p) + 1)$
  where $p \in P(op')$, $S \in R(X) - \{ result(p) \}\}$

$P_2(op) = \{(plan (p), (S, op, result(p)); dim(p) + 1)\}$
  where $p \in P(op')$, $S \in R(X) - \{ result(p) \}\}$

$P_3(op) = \{(plan(p), (result(p), op, result(p)); dim(p))$
  where $p \in P(op')$, if result(p) *in* R(X)}

The evaluation of the operator *"op'"* can be made in:
- a node where the fragment *x* is stored by transferring the result from a node where *op'* can be evaluated
- in a node where *op'* is evaluated by transferring the *x* fragment
- a node there *op'* can be evaluated and where the fragment *x* is stored

(c) If *x* is a fragment obtained from a previous evaluation of a *op'* operator and *y* is a fragment stored in the distributed database, the plan is build as in b2.

(d) If *x* and *y* are the results of evaluation the operators op_1 and op_2 then:

$P(op) = P_1(op) \cup P_2(op) \cup P_3(op)$ where:
$P_1(op) = \{(plan (p_1), plan (p_2), (result(p_1), op, result(p_2)); dim(p_1) + dim(p_2) + 1)$
  where $p_1 \in P(op_1)$, $p_2 \in P(op_2)$, $result(p_1) \neq result(p_2)\}$

$P_2(op) = \cup \{(plan(p_1), plan(p_2), (result(p_2), op, result(p_1)); dim(p_1) + dim(p_2) +1)$
  where $p_1 \in P(op_1)$, $p_2 \in P(op_2)$, $result(p_1) \neq result(p_2)\}$

$P_3(op) = \cup \{(plan(p_1), plan(p_2), (result(p_1), op, result(p_1)); dim(p_1) + dim(p_2))$
  where $p_1 \in P(op_1)$, $p_2 \in P(op_2)$, $result(p_1) = result(p_2)\}$.

After the plans were generated in the cases a2, b2, b3, b4 there is the chance to obtain for an operator two different plans $p1 \neq p2$ but with the same final evaluation node $final(p1) = final(p2)$. In this case is chosen the plan with smaller dim. Using this transformation an operator will have maximum one plan for a final node.

Example: Considering the following distribution of the fragments:

Node	S_1	S_2	S_3
Fragments	A	B	A, C

Assuming the request: $q = (A * \sigma_c (B)) \cup C$.
The operator's plans are:

$P(\sigma_c) = \{((S_2, \sigma_c, S_2); 0)\}$

$P(*) = \{((S_2, \sigma_c, S_2), (S_2, *, S_1); 1), ((S_2, \sigma_c, ,S_2), (S_2, *, S_3); 1),$

$((S_2, \sigma_c, S_2), (S_1, *, S_3); 1), ((S_2, \sigma_c, S_2), (S_3, *, S_2); 1)\}$

$P(\cup) = \{((S_2, \sigma_c, S_2), (S_2, *, S_1), (S_1, \cup, S_3); 2),$

$((S_2, \sigma_c, S_2), (S_2, *, S_3); (S_3, \cup, S_3); 1),$

$........ \}$

In the previous example can be noticed that exists a request evaluation plan with only one transferred fragment.

5 Conclusion

In section 3 is proposed an algorithm for redistribution of the fragments of a distributed database. The algorithm uses the request execution plan and some information that can be obtained when evaluating the relational determinant operators from the execution plan. This algorithm minimizes the size of the data transferred between the nodes of the network when the requests are evaluated.

For evaluating a request several fragments are needed, the fragments are stored in the nodes of a distributed database. An algorithm that determines an execution plan which minimizes the number of the fragments transferred between the nodes is proposed in section s4.

The results can be extended as follows:
-In the data transfer take into consideration the cost of the transfer instead of the size of the transferred data.
-In redistribution algorithm use a maximum size/dimension for each node instead a unique maximum size for all nodes.
-In the algorithm that finds the transfer plan between the nodes (algorithm used to evaluate a request) use a transfer cost instead of the number of the tranferred fragments.

Bibliography

[1] 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: 215-230, 2002.

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

[3] 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.

[4] Y. Huang, J. Chen, "Fragment Allocation in Distributed Database Design", *Fragment Allocation in Distributed Database Design, Journal Of Information Science And Engineering,* 17, 491-506 (2001).

[5] I. Lungu, A. G. Fodor, "Optimizing Queries in Distributed Systems," *Revista Informatica Economica nr. 1* (37), 67-72, 2006.

[6] M. T. Õzsu and P. Valduriez, "Principles of Distributed Database Systems", *2nd ed.,* Prentice-Hall International Editions, 1999.

[7] 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),* 613-618, 2007.

[8] L. Tâmbulea, M. Horvat, "Dynamic Distribution Model in Distributed Database", *Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844, Vol. III (2008),* Suppl. issue: Proceedings of ICCCC 2008, pp. 512-515.

[9] T. Ulus, M. Uysal, "Heuristic Approach to Dynamic Data Allocation in Distributed Database Systems", *Pakistan Journal of Information and Technology 2 (3)* , 231-239, 2003.

[10] 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, March 2008.

[11] 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, Nov. 1992.

Leon Ţâmbulea
Babeş Bolyai University, Cluj-Napoca
Faculty of Mathematics and Computer Science
Department of Computer Science
M.Kogãlniceanu No.1
E-mail: leon@cs.ubbcluj.ro

Manuela Horvat
Babeş Bolyai University, Cluj-Napoca
Faculty of Mathematics and Computer Science
Department of Computer Science
M.Kogãlniceanu No.1
E-mail: manuela.petrescu@gmail.ro

Manuela Horvat-Petrescu is a PhD.C. in Computer Science Department at Faculty of Mathematics and Computer Science of Babes Bolyai university of Cluj-Napoca.She received the Batchelor Degree in Computer Science, Master Degree in Component Based Programming and is working now as a Software Engineer in Montran Corporation.