Algorithm for Dynamic Partitioning and Reallocation of Fragments in a Distributed Database

Nicoleta Magdalena Ciobanu (Iacob)¹

(1) Computer Science Department, University of Pitesti, Pitesti, Targu din Vale Street, No.1, Arges and Postcode 110040, ROMANIA
E-mail: nicoleta.iacob_2007@yahoo.com

Abstract
For globally expanding organizations, applications generate dynamic workflows with frequent changes in database access models (write, read) at different sites. In these situations the integration of data fragmentation, replication and dynamic allocation of data fragments on different sites based on user needs is highly recommended. Static data proposed in model will help to determine dynamic histograms of data access and also the decision regarding the opportunity of a fragment reallocation. In this paper is proposed a heuristic algorithm for fragmentation and reallocation of new fragments to other sites in an unbalanced system in order to obtain an optimal dynamic distribution of data fragments with the smallest possible cost. The overall cost is formed from fragments partitioning cost, network cost, data storage cost, etc. The proposed model can be applied also in parallel databases because every site takes decisions about their own fragments and the decisions are taken without any site synchronization.

Keywords: Distributed Database, Fragmentation, Allocation, Algorithm

1. Introduction
The evolution of information technology generated also a tremendous evolution of database systems and in this context distributed database technology has changed the centralized point of view by offering major advantages.

Definition 1.1: According to “(Elmasri and Navathe, 1999)”, we can define a distributed database (DDB) as a collection of multiple logically interrelated databases distributed over a computer network, and a distributed database management system (DDBMS) as a software system that manages a distributed database while making the distribution transparent to the user. According to “(Khan and Hoque, 2010)” distribution of data is a collection of fragmentation, allocation and replication processes.

Definition 1.2: Fragmentation. The system partitions the relation into several fragments, and stores each fragment at a different site. The fragmentation is the partitioning of a global relation R into fragments R₁, R₂,…,Rₘ containing enough information to reconstruct the original relation R “(Iacob Ciobanu, 2011)”.
Informations regarding fragmentation, the nodes where the copies are stored and the rights of the fragments (read/write) in nodes are realized by a common catalog service using a distributed hash table.

Final methods monitor continuously the database and adapt to the recent workload. The static methods are based on offline analysis of database access.

Database data access is continuously made. Statistics are stored using dynamic histograms which are progressively maintained. The histogram is a collection of buckets and every bucket is
stored as a triplet \((bp, R[bp], W[bp])\), where \(p\) is the bucket number, \(R[bp]\) – number of read accesses from a histogram bucket (interval) and \(W[bp]\) – number of write accesses from a histogram bucket. Those histograms are an approximation, because the details are limited to the histograms buckets. In order to improve the performance of histogram operations in this algorithm are utilized so-called *equi-length histograms*. Because all the buckets have the same length \(W\), finding the correct bucket for a value is very simple. Every site, according to “(Hauglid et al, 2010)” offers a collection of histograms for every fragment which has a local replica. In order to store only the recent accesses, two sets of histograms are used: the old and the actual. The algorithms use both collections to make them have the same length of the bucket. All the operations are placed in the actual collection. Periodically evaluation algorithms are executed and as a result the old collection \(H_o\) is deleted and replaced with the content of the actual collection \(H_c\) and then \(H_c\) is deleted and then utilized for new statistics. In this way it is guaranteed the fact that the replicas were updated.

**Definition 1.3:** A *histogram* is an approximation of data distribution “(Donjerkovic et al, 2000)”, realized by partitioning in \(k (k \geq 1)\) disjoint two by two collections named buckets and analyzing every bucket by some concise information regarding the attribute values which are in a bucket and the frequencies corresponding to those.

The creation of histogram \(H\) related to replica \(R\) from the local site \(N\) consists of the following steps, according to “(Ioannidis, 2003)”: \(N\) distinct points are read and the following are determined: Minimum value \((X_{min})\) and maximum value \((X_{max})\) from the histogram; Length \((W)\) of a bucket; Number \((kh)\) of buckets: \(kh = \lfloor(X_{max}-X_{min})/W\rfloor\); Division points (borders) between buckets \(v_j(d) = X_{min} + j*W, j = 0,...,kh-1\), where \(j\) is the bucket number.

The replica access is made at tuple level. Every time a tuple is accessed in one of the local replicas, the histogram is updated accordingly. The data update in the histogram is done like this: A new point \(x\) is read (the access of a tuple); The bucket in which \(x\) resides \((bp = x/W)\) is determined; If in every moment the access of a tuple is made beyond the range of buckets of the actual histogram \((bp \notin H)\), a new bucket is created for \(x\); Value \(x\) is inserted in the corresponding bucket; The type of access for \(x\) is verified. If \(x\) is a read operation then the number of reads from the \(bp\) bucket increase; otherwise the number of writes increase; The points are uniformly redistributed in the values range of the bucket; At the end, the data from hash table \(TH\) from local site are updated.

2. Partitioning of fragments and reallocation. Algorithm

The re-fragmentation and data allocation algorithm on different nodes from database where are frequently accessed, have the role to minimize the network traffic by identifying parts of a table which must be extracted to form a new fragment and migrate them to a remote site by taking into consideration the number of fragments on a node and their dimensions.

A fragment reallocation or the transfer of the write right (only in the case where a replica have read access and the other write access) between the fragment replicas situated on two sites, with the smallest load and the biggest benefit, will be made by talking into consideration function cost.

Given the statistics existent on every site, the proposed algorithm examine access for every replica and evaluate the possible reallocations and refragmements based on recent history using a cost function (UtilEM). The algorithm is executed at given time intervals, individually for every replica. If the site does not have a positive useful value, then no change is made on that site.

Given the fact that the number of accesses of replica include only the recent history, the actual number of tuples from fragment is scaled (fragment dimension) with the weight of function cost (FSC) and in this way is independent of the history length. Because migrations cause delay of accessing tables, the migration must not be permitted when the number of local accesses is higher than the number of remote accesses (this fact can result in an unstable situation in which a
fragment is migrated continuously between sites). To reduce this problem, we scale the benefit part of the functions cost by \( FSB \in [0...1] \).

**Optimizations.** In order to reduce the costs, some optimizations can be made. At the beginning of an update transaction the data integrity is assured by sending some “warning” messages to the nodes that contain replicas with read rights which are going to be updated. After finishing the update transaction, only the nodes marked with “warning” will receive the updated value (thus limiting the network traffic) to update the database catalogue and then the “warning” will be deleted.

The algorithm can be further optimized: using the control sum on the same data collection, if a hash table was modified (the control sum is different) the table can be split in smaller fragments which will be verified and in this way only the modified data will be updated, avoiding the traffic which would be needed to update the entire table.

The proposed system model is an improvement of the system model from “(Hauglid et al, 2010)”: if in “(Hauglid et al, 2010)” the sites have equal computing and communication capacities, the proposed system is unbalanced (the nodes execute the same operation on the same data collection at a different time, latency between the two nodes depend on the network and the load on a node is not constant); storage space on every site is continuously monitored; it offers a high degree of availability (According to “(Vasileva et al, 2007)” the best response time is obtained when there are two data copies). We have to mention that the number of replicas with write access is proportional with the update costs. Based on recommendations “(Weikum and Vossen, 2000; Ghemawat et al, 2003)” maximum three replicas with write access is proposed. The proposed model can be customized, permitting the change of the number of replicas.

The benefit of replica migration (write access replica) from the local site \( N \) to remote site \( N1 \) consist of the fact that the remote writes will become local operations. The cost will consist of writes at local sites and the cost of migration.

Reallocation of a fragment \( F \) from node \( N \) in node \( N1 \), according to “(Horvat-Petrescu, 2009)” is necessary when:

a) The number of update requests received from node \( N1 \) (remote) is bigger than the number of requests received directly from node \( N \) (local):
\[ nwr(N1) > nwl(N) \quad \Rightarrow \quad nwr(N1) - nwl(N) > 0 \]

b) The cost of extraction and migration of fragment \( F_{new} \) is smaller than the difference between the cost of requests received in node \( N1 \) and retransmitted to node \( N \) and the cost of requests received from node \( N \) and transmitted to node \( N1 \).

b1) The extraction and migration cost of the fragment \( F_{new} \) is determined like this:

Let \( P \) be the collection of sites on which reside the replicas with read rights of fragment \( F \) and \( Q \) the collection of sites on which reside the replicas with write rights of fragment \( F \). The matrix of transfer cost and latency between the nodes \( P \) and \( Q \) from database, noted with \( C \), is:
\[ C = \{ C_{P,Q} \mid P, Q \text{ nodes in database} \} \]

**Observation:** The transfer and latency cost from node \( P \) to node \( Q \) is the same with the transfer and latency cost from node \( Q \) to node \( P \):
\[ C_{P,Q} = C_{Q,P} \]

The notification cost of replica \( P \) with read right and the cost of obtaining an answer is:
\[ C_{w} \leftarrow 2 \times \sum_{Q} C_{N,Q} \]

(We have “2” because it is taken into consideration the cost of received queries in node \( Q \) and the retransmit of those to node \( N \).)

The notification cost of replica \( P \) with read right and the cost of updating their values is:
The cost of a fragment extraction $F_{\text{new}}$ from the local site $N$ is defined as the product between the extraction cost ($C_{\text{et}}$) for attribut, number of attributes ($N_{\text{at}}$) and the actual number of tuples ($\text{card}(F_{\text{new}})$) from fragment $F_{\text{new}}$ (transferred quantity):

$$[6] \quad C_{\text{et}} = \text{card}(F_{\text{new}}) \times N_{\text{at}} \times C_{\text{et}}$$

The cost for migrating a fragment $F_{\text{new}}$ from site $N$ to site $N_1$, being the principal cost of refragmentation and reallocation is:

$$[7] \quad C_{\text{M}} = 2 \times \sum_{Q} C_{N,Q} + 2 \times \sum_{Q} C_{N,1,Q} + \sum_{N} C_{N,P} + \sum_{N_1} C_{N_1,P} + \sum_{N} C_{s}$$

$C_{s}$ – represent the cost of storing the fragment $F_{\text{new}}$ at site $N_1$ and is defined as the product between the storage cost ($C_{\text{st}}$) of the tuple at site $N_1$, actual number of tuples from fragment $F_{\text{new}}$ and the decision variable $y$:

$$[8] \quad C_{s} = C_{\text{st}} \times \text{card}(F_{\text{new}}) \times y,$$

$$y = \begin{cases} 1, & \text{if the fragment } F_{\text{new}} \text{ is store on site } N_1; \\ 0, & \text{otherwise.} \end{cases}$$

The decision variable $y$ is used to select only those cost values for the sites where the fragments are stored.

The total cost for the extraction and migration of a fragment $F_{\text{new}}$ from the site $N$ to the site $N_1$ is:

$$[9] \quad C_{T} = C_{\text{et}} + C_{\text{M}} = C_{\text{et}} + 2 \times \sum_{Q} C_{N,Q} + 2 \times \sum_{Q} C_{N,1,Q} + \sum_{N} C_{N,P} + \sum_{N_1} C_{N_1,P} + \sum_{N} C_{s}$$

The necessary time to update a fragment $F$ on node $N$ is very small and can be neglected ($T_{a\rightarrow 0}$).

b2) Difference between costs (the cost of requests received in node $N_1$ and retransmitted to node $N$ and the cost of the requests received from node $N$ and transmitted to node $N_1$), which in fact is the benefit, is determined like this:

$$[10] \quad D_{\text{C}} = 2 \times \sum_{Q} C_{N,1,Q} \times (\text{nwr}(N_1) - \text{nwl}(N)) + 2 \times \sum_{Q} C_{N,Q} \times (\text{nwr}(N_1) - \text{nwl}(N))$$

The difference between the queries solving cost on the two sites, $N$ and $N_1$, and the extraction and migration cost of the fragments ($\text{UtilEM} = \text{benefit} - \text{cost}$) must be positive:

$$[11] \quad \text{UtilEM} = D_{\text{C}} - C_{T} = (2 \times \sum_{Q} C_{N,1,Q} \times (\text{FSB} \times \text{nwr}(N_1) - \text{nwl}(N))) + 2 \times \sum_{Q} C_{N,Q} \times (\text{FSB} \times \text{nwr}(N_1) - \text{nwl}(N))$$

$$+ (F_{\text{S}} \times C_{\text{et}} + 2 \times \sum_{Q} C_{N,Q} + 2 \times \sum_{Q} C_{N,1,Q} + \sum_{N} C_{N,P} + \sum_{N_1} C_{N_1,P} + \sum_{N} C_{s} > 0$$

The extraction and migration algorithm of a fragment from node $N$ to node $N_1$ consist of the following steps:

- The collection of nodes $P$ with read access and the collection of nodes $Q$ with write access are determined.
- All the possible new fragments $F_{\text{new}}$ and the possible sites $N_1$ are evaluated using the function $\text{UtilEM}$. The proposed heuristic consist of transmitting the fragment which have the smallest communication cost at the sites where is util - Migrate($F_{\text{new}}, N_1$). The selection of node $N_1$
will take into consideration the best response time with the best benefit. In this way the load of every node and distance from node N will be considered.

- Then all the compatible fragmentations with positive values will be performed. Two fragmentations are compatible if the extracted fragments do not overlap. In case we have two incompatible fragmentations, the fragmentation with the highest UtilEM value is chosen.

- If a refragmentation decision is taken, all the sites with read rights are notified in order to be able to perform the same refragmentation. Because the update of read replicas is made after all the replicas with write rights are updated, problems related to data consistency can appear. To resolve this, in the beginning of an update transaction the data consistency is assured by transmitting some warning messages to the sites that contain read replicas of the data that will be updated.

- To keep the number of fragments low, any adjacent fragments which have replicas on the same site are joined \( F \leftarrow F_1 \bigcup F_2 \). If two fragments are joined, the replicas with write rights of those fragments must be updated (those sites must delete their replicas or obtain a replica of the joined fragment).

- In the end, old access statistics from the local replicas with write access are eliminated (histogram is reorganized).

The algorithm for refragmentation and reallocation can be formalized as:

```plaintext
START [REFRAGMENTATION(F,R)]  // R is the master replication of F and it is on the local site N
INPUT { The sites of DDBs: S_1,...,S_n; Sites communication cost matrix; Loading sites: load(S_1),...,load(S_n); The current histogram corresponding to replica R from the local site N is:

\[ H \leftarrow H_c[N,R], \text{where} \ H = \bigcup_{ih=1}^{kh} (b_{ih}, R[b_{ih}], W[b_{ih}]), \ ih=1,kh \]

BEGIN

// The load of every site is computed, “(Ozsu and Valduriez, 2011)”
For each Sr ∈ S, r ← 1 to n Do compute (load(Sr)) End For
N ← getLocalSite(F) // N is the site where fragment F is located
// The write right is transferred from site P to site Q
P ← Ø
For each Sr ∈ S, r ← 1 to n Do
If getRight(R,Sr) = ‘read’ and getRight(R,Sr) ≠ ‘write’ Then // replica R has the right read on the site Sr
P ← P \bigcup S_r
End If
End For
Q ← Ø
For each Sr ∈ S, r ← 1 to n Do
If getRight(R,Sr) = ‘write’ Then // replica R has the right write on the site S_r
Q ← Q \bigcup S_r
End If
End For
frag ← Ø // the set of fragments of F; nf ← 0 // the number of fragments of F
For each N1 ∈ S\N, N1 ← 1 to n Do
If getRight(R,N1) = ‘write’ Then
```

```
UtilEM = \(2 \sum_Q C_{N1,Q} * (FSB * nwr(N1) - nwl(N)) + 2 \sum_Q C_{N,Q} * (FSB * nwr(N1) - nwl(N)) - (FSC \times C_E + 2 \sum_Q C_{N,Q} + 2 \sum_Q C_{N1,Q} + \sum_P C_{N1,P} + \sum_P C_{S})\)

If (UtilEM > 0) and \((max-min+1 > \text{card(frag)})\) Then
\(nf \leftarrow nf + 1; \text{frag} \leftarrow \text{frag} \cup (N1, \min, \max, \text{UtilEM}); \) End If; End If; End For

ordering \((s, c, nf)\) // the nf sites s are ordered by value c; sorting\((s, d, l, nf)\) // the nf sites s are ordered by the response time. This, will take into account the distance \((d)\) to the site N and loading sites \((l)\) removeIncompatible\((\text{frag})\)
\(nm \leftarrow 0; \text{nmr} \leftarrow 0\)
For all \((N1, \min, \max, \text{UtilEM}) \in \text{frag}\) Do // with the biggest benefit and smallest response time
If \(\text{load}(N1) > \text{limsp} T\) Then
\(F1 \cup F_{\text{new}} \cup F2 = F; \text{Migrate}(F_{\text{new}}, N1)\)
\(/\text{notify(SETSITESREPLICAWRITE(R))}/\)
For each \(S_m \in S, \text{rm} \leftarrow 1\) to n Do
If \(\text{getRight}(R, S_m) = \text{‘write’} \) Then
If \(\text{checksum}(TH, S_m) \neq \text{checksum}(TH, N)\) Then
\(\text{UPDATETABLE}(TH, S_m)\)
// warning messages are sent to sites that contain replicas with read right of the data that need to be updated
For each \(S_r \in S, \text{rc} \leftarrow 1\) to n Do
If \(\text{getRight}(R, S_r) = \text{‘read’} \) and \(\text{getRight}(R, S_r) \neq \text{‘write’} \) Then
Send “warning” message; End If; End For
End If; End If; End For
End If; End If; End For

compute \((\text{load}(N1))\); compute \((\text{load}(N))\) // is recalculated loading the site after the transaction
// are updated only data from the replicas with read right marked with “warning” of which checksum differs by that of site N
Boolean \(\text{dataIsWarning} = \text{true}\)
While \((\text{dataIsWarning})\) Do
\(\text{dataIsWarning} = \text{false}\)
For each \(S_r \in S, \text{rc} \leftarrow 1\) to n Do
If \(\text{getRight}(R, S_r) = \text{‘read’} \) and \(\text{getRight}(R, S_r) \neq \text{‘write’} \) Then
\(\text{dataIsWarning} = \text{dataIsWarning} \) or \((R \text{ is marked with warning message)}/\)
If \(\text{checksum}(TH, S_r) \neq \text{checksum}(TH, N)\) Then
\(\text{UPDATETABLE}(TH, S_r)\) End If; End If; End For
End If; End For

\(F \leftarrow F1 \cup F2\) // fragments are joined
Histogram is reorganized
END
To avoid different fragmentation decisions which take place simultaneously at sites with replicas of the same fragment, this algorithm is applied only to master replicas.
The result of every execution of algorithm may be:
- don’t do anything (the fragment is as supposed to be)
- migrate the entire replica with write right, or
• extract a new fragment $F_{\text{new}}$ and migrate new replica with write access to the site $N_1$ (if on $N_1$ is enough storage space available).

3. Conclusion
The proposed method for an unbalanced distributed database system has two major components: 1) the detection of replicas access models and 2) given those statistics, decisions on refragmentation and reallocation will be made. The dynamic characteristic of the model consist in the fact that the change of access models (read, write) must lead to the refragmentation and reallocation of fragments and creation or deletion of fragments replicas (the fragments replicas can change their rights).

In this paper we focused on data allocation problem with the aim to assure an optimal distribution of data in the process of the distributed database design in correlation with data fragmentation. The decision to use a fragmented database is very important because it determines the execution performance of a distributed query.

Future plans consist of implementing the algorithm for execution of simulation tests and evaluation of results.

Acknowledgment
This work was partially supported by the strategic grant POSDRU/88/1.5/S/52826, Project ID52826 (2009), co-financed by the European Social Fund – Investing in People, with in the Sectoral Operational Programme Human Resources Development 2007-2013.

4. References