Universitatea Babeş-Bolyai Cluj-Napoca
Faculy of Mathematics and Computer Science

DYNAMIC RELOCATION OF DATA FRAGMENTS IN A DISTRIBUTED DATABASE

PhD Thesis - summary

PhD Student:
Manuela (Horvat) Petrescu

Scientific advisor:
Prof. Univ. Dr. Leon Ţâmbulea

Cluj-Napoca
2010

Abstract

In the research activity as in the production one, the distributed technologies speedup in gaining ground in expense of the centralized databases. If centralized databases designing can be done most of the times by following a set of rules, in the case of distributed databases the designing is a little bit different. Designing a distributed database is a complex process as the fragmentation and allocation procedures (for data fragments and for their rights) should use data analysis and a cost function evaluation. These processes are in general too complex to be solved manually, and the results obtained do not reflect the amount of effort involved in elaborating them. The present paper proposes heuristic algorithms for reaching the optimal state for a dynamic distribution of the fragments in the nodes of a distributed database. These algorithms include a reallocation algorithm, a recovery algorithm, but also clearing algorithms: on each node, as a centralized one.

Data access patterns change in time, so the design of a distributed database should change to meet the patterns changes and requirements in order to obtain the optimal state. Changing and redesigning a distributed database is costly, and is as predictable as the user's access pattern changes. In this aspect the proposed algorithm brings innovation through the dynamic characteristics – it redesigns and change the data fragment allocation according to the user needs. The statistics proposed in the system help determine the data access patterns and are used in determining if the reallocation of the data fragments is optimal.

New software metrics based on the system's statistics are defined and introduced to assess the efficiency of a reallocation data fragment (paragraphs: When reallocation is optimal – case A: a balanced system and case B: a non-balanced system).

The proposed clearing algorithms keeps the system storage needs to the minimum space as it deletes not used or less used data from each node in order to obtain storage space. The clearing algorithm is adapted to each node's needs and storage capabilities – the clearing algorithms run when the available storage space is below a specific limit, or other node specific conditions are fulfilled). The main clearing algorithm runs in a centralized manner performing fragments reallocation, changing data fragments rights, so on.

The recovery algorithm takes care of the recuperation of the falling nodes and also takes care of the system partitioning cases.

Keywords: dynamic, distributed database, optimal, model

Introduction

Motivation

The business environment has an increasing need for distributed database (in expense of the centralized database) and the desire for reliable, scalable and accessible information in client/server applications is steadily rising. Distributed database systems provide an improvement as the data access is faster, and a single-point of failure is less likely to occur. However, there is the complexity problem that arise when attempting to manage and control distributed database systems, and the transparency problem – how to provide the appearance of a centralized database system. Also, when adding capacity, applications should not need to be cluster aware because the cluster-aware applications have to change as your volume of data and distribution changes. Cluster-aware applications do not just require code changes as the cluster grows: they also need to be tested, go through the Quality Assurance (Q/A) process, be deployed, certified, and so on. This can cause weeks of coordination efforts across the enterprise and inevitably drains the infrastructure from resources that can be better used elsewhere.

The distributed database seems to be the solution to these problems even if they are sometimes hard to implement due to a set of factors such as complexity and cost. Even so, the main companies announced distributed database solutions: Oracle RAC, IBM DB2, SQL Server2008 Federated DB (named FederatedDB). At a closer look, an interesting fact appears: some of these distributed databases are only partially distributed as they have multiple servers that share a common storage. The storage remains centralized in OracleRAC and in IBM DB2 PureScale. These databases offer a good solution for server failures; however, by themselves offer no protection against disasters or storage failures. Even if the FederatedDB offers a truly distributed database (with multiple servers and storages), it does not offer by itself a good solution for storage failures as the data is only fragmented and not replicated.

In this thesis is proposed a method for dynamic distribution for the fragments of a distributed database (multiple storages and servers) and a method for dynamic allocation for the access rights for these fragments. The data is replicated in at least two replicas in the proposed model- assuring in this manner the backup.

The novelty of this model is represented by its dynamic characteristics: the data distribution is changed according to the user access patters to offer the highest availability with the smallest cost. In this aspect a whole model is proposed: transaction processing procedures, clearing and reallocation procedures, recovery and deadlock detection & preventing procedures. Also, the cost of the model expressed in network cost, reallocation cost, execution cost, so on was always a concern as the model is wanted to be not only dynamic but also less costly and to make database management easier, more flexible and more cost-effective. The goals of the proposed model were always in sight when elaborating the model's algorithms and can be summarized:

- Dynamic reallocation of the data
- Improved reliability and availability
- Improved performance
- Minimal storage capacities and costs
- Minimal communication costs
- Dynamic storage administration

Characteristics of the Proposed Model

Scalability

In the old days, when a database server was running out of capacity, it was replaced with a new and larger server. As the servers grow in capacity, they get more expensive. For the applications running on the proposed model implementation there are alternatives to this solution: add a new server to the cluster and as soon as the new instance is started, the application can take advantage of the extra capacity. The distributed architecture accommodates to rapidly changing businesses requirements and to the resulting workload changes.

Storage Management

Database administrators face many challenges in the area of storage management. With the rapid growth in database size and the demand for up-time continuing to increase, taking the database off-line to perform maintenance operations such as changing disk configuration is becoming increasingly difficult to schedule. The Recovery procedures presented in the proposed model provide the solution for these storage management challenges by automating the entire function. All an administrator has to do is to allocate a set of storage devices to a database instance and the model takes care of the rest. It automates the placement and the replacement of the database files and spreads data across all available storage resources to optimize performance.

Proactive Space Management

The model provides proactive space management capabilities with its space monitoring, clearing and space trending features. If the available space usage crosses a user-defined limit, or if the processes cannot run properly in the available space, the model will obtain the needed space by taking corrective measures: it runs the clearing algorithm and deletes the rarely accessed data (with the mention that at least two copies of a data having write permission must exist in the system). Also, the amount of the data that could be deleted to gain space can be set either in terms of percentage (i.e. the $10^{\%}$ less used data fragments) or fixed (a fixed number of data fragments).

Loading Distribution

The proposed model offers loading distribution features (when the query is resend to other node, or in sql operations such as bulk insert). In resending the query, the model selects the node with the best response time - in this manner are taken into consideration the network traffic and also node loading and processing capacities.

Ensuring SQL Performance

In the each node of the network are stored the most accessed data fragments - so for the most of the queries the data is highly available and the additional network performance is minimal. This high degree of availability is reflected in sql performance. In order to optimize even furthermore, it should be applied optimization algorithms for distributed databases.

Backup

The model offers protection against disasters and storage failures using the data fragments replication and the procedures that ensure a minimum number of replicas for each data fragment. Each data fragment is replicated in at least two replicas. The model is highly configurable and allows the database administrator to set the minimum number of the replicas (a number not smaller than 2). The model procedures increase the number of replicas when one site is down in order to assure the minimum number of the requested replicas.

Intelligent Recovery

The recovery procedures for a node offer a fast and complete recovery, updating the database structure, the data fragments (their values and rights) as well as the database catalogs without requiring database administrator intervention.

Easily configurable policies

It should be easy for the database administrator to change the behavior of the model – the model is highly configurable and is parameterized so it can fit the requirements to any running application.

Conclusion

A simple sentence can conclude: the model proposed in this thesis is not only innovative but can also be efficient, configurable, easy to manage and cost efficient; the model implementation would have all the characteristics mentioned before.

Table of Contents

1. Distributed Database

A distributed database can be defined [Ta03, Öz99] as a collection of interconnected databases that ensure the transparency for a user; so, as regarding usability, a distributed database is not different from a centralized one. A distributed database has much to offer in terms of availability and fast data access, architecture independence but it also raise some problems regarding the management complexity, lack of standards, etc. [Öz99, Ro09, Ku09].

In a distributed DBMS, a relational table can be divided into two or more distinct partitions/fragments. The partitions can also be replicated, and this feature causes problems in concurrent access and in the management of the distributed databases. [Ta03, Öz99]. The fragmentation represents the partitioning of a global relation R into the fragments $R_1, R_2, \ldots R_n$, which contain enough information to recover the R relation [Ta03, Da04]. The fragmentation types are: horizontal, vertical and mixed.

2. Model presentation

2.1. Introduction – general considerations

The model is based on the following ideas and considerations: a distributed database, a collection C of fragments, each fragment is replicated in at least two replicas having a read or write right in each node. The idea this thesis is based on is the following: the replicas will dynamically change their rights and their allocation in the nodes of the database according to the user's access patterns [TaHo08].

Let's note with:

- D - a data fragment from the C collection, and with
- $C(N) \subseteq C$ – the set of fragments contained by collection C that are located on the network node N.

In the proposed model it's considered that:

- The set of fragments $C(N)$, and their distribution in the network's nodes can be changed dynamically for each node N in the database
- A fragment D stored in the node N, can have either write or read permission in the node N. The same data fragment D can have a different permission in a node different from N.

Due to the propagation cost and update cost involved in updating a replicated data, the management of a fragment D having write permission is more expensive than the management of the same fragment having read permission, so the goal is to have as few write permissions as possible, but enough to sustain a certain number of node failures. The model should not replicate all the data on all the fragments (with read permission), as this will imply having large storage capacity on each node (in this manner, is lost one of the advantages of a distributed database).

2.2. Description– model image

The model can be seen as an upper layer of the DBMS, as an extension of the DBMS that uses the current features and facilities. The innovations in this model are represented by the way the

write and read transactions are treated and by the change permissions mechanism. The proposed model implies changes in the central management algorithms, as in the processes on the each node, but does not interfere with the protocols used in data blocking, or transaction management (2PL). The following image visualizes the model:

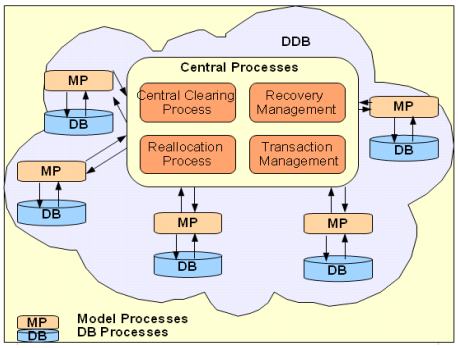

Fig.1 Model representation

The database in the proposed model can run if one of the central processes fails, all central processes (except the central clearing) are not related to a node- so there is no specific node where they are run. In case of running node failure (or other reasons), the central clearing process can be restarted on a different node from the database. In each node, the model specific processes are responsible by running the node clearing process, the recovery, the update processes.

2.3. Model specific information and Database catalog

Data access patterns are determined in order to allocate properly the data and to obtain an optimal model. Statistical parameters are used to monitor accessed data in each node; the clearing and reallocation algorithm (for example) will use them in taking the reallocation and delete decisions. The parameters are stored in each node in the database catalog. [TaHo08, TaLHo08]:

Model statistical information

R(N, D), W(N, D)

R(N, D) represents the number of read/write requests received by the node *N*, for accessing the fragment *D* for reading/writing (it makes no importance if the fragment is stored or not on the node *N*, or if the request is forwarded to other node).

W1(N, D)

W1(N, D) represents the write requests number received by the node *N* for accessing the fragment *D* which is not stored on the node..

Model specific parameters

W_Max, W_Min

W_Max and *W_Min* are global parameters and represent the minimum/maximum number of nodes where a data fragment can be saved having write permission.

W(D)

W(D) represents the number of nodes where a data fragment D has write permission.

P(N)

P(N) is specific to each node and represents the maximum period of time that must be elapsed between two consecutive runs of the clearing algorithm.

Functia cond(N, D, Pa)

Cond (N, D, Pa) is a function/condition, defined in each node *N* and is used to decide the access permission *Pa(read, write)* for fragment *D* stored in the node *N*. This function will be determined using a set of factors such as: the available memory on the node *N*, the current loading, so on.

Replica Location- *RL(N, D)*

Every node should be able to locate all the replicas of a data fragment. In the database catalog in each node *N*, for each data fragment will be stored the fragment's replicas locations and the rights of these replicas (read/write).

Maximum update time a read-only replica- MaxUT

MaxUT is a global parameter and is used to assure that all read replica are updated in a specified amount of time. It is used only for critical data, as a its excessive use increase the network traffic and decrease the efficiency.

The number *k* of data fragments deleted in the central clearing process

It specifies the number of fragments that would be deleted in the central clearing and reallocation process.

The number of data fragments *x* deleted in the node's clearing process

The parameter *x* is used in the clearing process in a node for determining the fragments that would be deleted. All the data fragments for which the read/write number of access is below this limit *x* will be deleted.

$$R(N,d_i) < x \text{ sau } W(N,d_i) < x,$$

Proposed initial values

W_Min - *W_Min* parameter can be set to a value equal with 1, but for reliability reasons is recommended to be greater or equal with 2. *W_Min* parameter should also be related to the database characteristics: in small distributed databases (when the total number of nodes is below a certain limit) it is enough to exist only 2 replicas, in large databases (hundreds or thousands of nodes), only two replicas with write permission might be not enough [Gh03,Va07].

W_Max – The greater is the number of write permissions, the greater is the update cost. It is not recommended to have too many write permissions for a data fragment. Based on the recommendation of [We02, Gh03], is proposed a default start value of 3.

P(N) – It should be specified according to the database specifications and according to the data access patterns. The period of time should be determined according to a set of factors: the geographic areas where the nodes are located and to factors that depend on each application: the maximum or minimum loading time, for example.

Parameter k – it specifies the number of replicas deleted in the global clearing process. For example, a rate of 20-25% would provide a good clearing rate (after two or three runs of the clearing process, the number of not used replicas would be almost half of the initial number).

2.4. Model behavior

The scope of each database is to resolve the received requests: read requests and write requests. According to the request type and to take advantages of the read requests, the database performs in a different manner in order to execute it. In a centralized database, each request *q* needs a set of data fragments in order to be resolved; each set is combined with other sets and the result is sent to the user. In a distributed database not all the necessary data fragments are stored on the same node [Da09]; when solving a request *q* should be taken into consideration the localization of the data fragments *r(q)*. In the proposed model, the read replicas of a data fragment are updated asynchronously, a solution with benefits but also with drawbacks [Yo08, Ed97, CaDo09, St09].

Having the read fragments updated asynchronously raise a question: how to assure that all the data from the table is parsed when solving a request? (The problem is raised especially for inserted/deleted data). The solution would be to have one or two master table that would be always up to date (their update would be part of the write transaction), and each node should interrogate these nodes to check if updates were performed. This method involves network traffic and it also presents a potential risk: the nodes where the master tables are stored to become bottlenecks in the model. A better solution [Pu91] would be using the "dirty" messages.

When an update/insert transaction starts, it should send "dirty" message notifications to all the read replicas of the data involved in the updates. The "dirty" mark would be removed when the updated value is received by the read replica. For the insert/delete operations, the "dirty" notifications messages should be applied on the table. For tables with critical information, the *MaxUT* parameter can be used – see *MaxUT* definition.

According to the data fragments localization, two major situations can occur in solving a read request *q* that arrived on the node *N* [TaHo08]:

Case 1. All the required fragments $\{d_i \mid d_i \rightarrow q,\ i=1..\ k\}$, are stored on the node *N*

Case 2. One or more fragments are not stored on the node *N*.

In write requests, according to data fragments localization, similar cases can occur:

Case 1. All the required fragments $\{d_i \mid d_i \rightarrow q,\ i=1..\ k\}$, are stored on the node *N* and they have write permissions

Case 2. One or more fragments are not stored on the node *N,* or they don't have the necessary write permission:

- The fragments with read only permissions stored in the node *N* can change permissions from the read permission to write permission if certain conditions are fulfilled.
- The request can be sent to other node that has write permission for the data fragments that has to be updated.
- For the fragments that cannot be updated in the node *N,* the update request is sent to the node *N1* that stores the primary copy/replica of the fragment *D*, which performs the update.

2.5. Fragment's reallocation

Reallocation in the model - Case A: a balanced system

Reallocation of a write permission between two nodes should be performed only if the difference between the solving cost and the reallocation cost is greater than zero [Ho08].

In the first approach, it's considered that the system is balanced and all the nodes perform in the same manner. It's also assumed that the latency between two nodes is constant and is equal with Δ. Other notations:

- *Tu* - the time required to update a data fragment *D* on a node *N*
- *n* – total nodes number in the database network

The reallocation of a replicated data fragment d from node *N* to the node *N1* is needed when:

- The number of updating requests received by node *N1* is larger than the number of requests received directly by node *N* (so $W(N1, d) > W(N, d)$) and
- The transfer cost is smaller than the difference between the cost of the request received from the node *N1* and sent to *N* and the cost of the request that will be received from the node *N* and sent to *N1*.

These conditions can be mathematically described,and the following results are obtained (the update time of a data fragment is small, so it can be considered that $Tu \rightarrow 0$):

The **Total Cost (Tc)** for reallocation a data fragment is:

$$Tc = 2 * \Delta * n + 2 * \Delta * W(d) - 4 * \Delta$$

Cost Difference (Dc):

$$Dc = 2 * \Delta * (W(N1, d) - W(N,d))$$

The second condition for optimal reallocation can be written as:

$$Tc < Dc <=>$$
$$W(N1,d) > W(N,d) + n + W(d) - 2$$

In conclusion, the replica transfer is recommended and is optimal when the difference between the update request numbers from nodes *N* and *N1* is larger than the total number of nodes plus the number of write replicas minus 2 [Ho08].

Reallocation in the model – Case B – A no balanced system

In the second approach (a more realistic one), it's supposed that the system is not balanced and the nodes require different time for reading/writing the same set of data. Also is assumed that the latency between the two nodes depends on the network, and the node loading is not constant. A matrix *C* containing the latency and transfer cost between the nodes will be used to decide when the reallocation is optimal (*Tu* and *n* have the same meaning as in the case A):

- $c_{P,Q} \mid P.Q$ *nodes in the database*}, and $c_{P,Q}$ represents the latency cost between the nodes *P* and *Q* from the database. Notice that:

$$C_{P,Q} = C_{Q,P} \; ; \; P,Q \text{ nodes in the database}$$

The cost of notification replicas with write permission and to obtain the answer is:

WriteReplicaCost:

$RW(D) = \{Q \ ; \ Q$ *contains a write replica of data fragment* $D\}$

$Wrc = 2 \ (\sum_{Q \in RW(D)} c_{N,Q}$; Q *contains a write replica of data fragment* $D)$

The cost of notification replicas with read permission to actualize their value is:

ReadReplicaCost:

$RR(D) = \{P \ ; \ P$ *contains a read replica of data fragment* $D\}$

$Rrc = (\sum_{P \in RR(D)}^{P} c_{N,P}$; *P contains a read replica of the data fragment* $D)$

Using the optimal reallocation conditions mentioned in the case A – a balanced system, the following result is obtained (where *P / Q* are nodes that contain a read/write replica of the data fragment *d*):

$$2\left(\sum_{Q \in RW(D)} C_{N,Q} + \sum_{Q \in RW(D)} C_{N1,Q}\right) + \sum_{P \in RR(D)} C_{N,P} + \sum_{P \in RR(D)} C_{N1,P} < 2 * \left(\sum_{Q \in RW(D)}^{Q} C_{N1,Q} + \sum_{Q \in RW(D)}^{Q} C_{N,Q}\right) * (W(N1,d) - W(N,d));$$

3. Model algorithms

3.1. Requests solving algorithms

The requests solving algorithm has the role to decide how to solve it according to a set of factors such as the request type: a read or an update request, the permissions of the fragments, the fragment's location, so on.

According to the fragment's locations and permissions, the algorithms deal with two situations:

- All the data fragments required by the transaction are stored in the node where the request was made and have the corresponding permission (i.e.: for an update request, all the data must have write permissions).
- When one or more fragments required by the transaction are not stored in the node where the request was made or they don't have the corresponding permission.

All the data fragments are stored in the same node.

The first and simplest case is when the set of fragments required by the request q is stored in the same node, the *MaxUT* parameter was not set and all the data fragments have the permissions required by the request *q* (even so, the check for "dirty" data should still be performed).A request *q* received by node *N* needs to access a set of fragments:

$$r(q) = \{d_i \ , \ i \in I_q\} \subset C(N)$$

The algorithm in this case can be formalized as following:

```
If r(q)= {di|i∈ Iq}⊂C(N)
          and r(q) has the permission required by q Then

    If q is read request Then
          //check for dirty status
          Boolean dataIsDirty = true;
         While (dataIsDirty) Do
                dataIsDirty = false;
               For each di; di∈ N, i=1.. k, Do
                         dataIsDirty = dataIsDirty or (di is dirty)
                End For
          End while
    Else
          For each di; i∈Iq do
                For each Nk; Nk ∈ RL(N,di) and
                            (di has read permission în Nk ) Do
                      send "dirty" message
                End for
          End for
    End If

    Execute q;

    //update system parameters, send last values for write requests
    If q is read request Then
          For each di; i∈ Iq do
                R(N, di) = R(N,di) + 1;
          End for
    Else
          For each di; i∈ Iq do
                W(N, di) = W(N, di) + 1; // q is write request
                //send last value to read replicas
                For each Nk; Nk ∈ RL(N,di) and
                            (di has read permission în Nk ) Do
                      send di
                End for
          End for
    End If
End If
```

Fragments with invalid permission or stored in different locations

The second case is when the at least one fragment from the set of fragments required by the request *q* is stored on a different node or when at least a data fragment does not have the permissions required by the request q in the node *N* (i.e.: q is an update request, but the data fragment in the node *N* does not have the write permission). Formalizing this:

If exists at least one fragment d_j; $0 < j <= i$ and the set $r(q) = \{ d_i \mid i \in I_q \}$; with $d_j \not\subset C(N)$; or exists at least one fragment does not have the required permission (ie. the fragment has a read permission and the request needs a write permission), then the following algorithm will be used:

```
//Read Requests:
If q is read request Then

    //transfer data fragments to the node N
    For each dj; 0 < j <= i, r(q) = { di | i∈ Iq }; dj∉C(N); do
          Select N1 using RL(dj);
          Transfer dj to N

          If cond(N, dj, read) Then      //there is empty storage, ...
                store dj on N
                R(N, dj) =  R(N, dj) + 1;
                update RL(dj);
          End If
    End For

    If (MaxUT isValid) Then
          For each di; i∈ Iq} do
                If (currentTime – lastUpdateTime(di) > MaxUT ) then
                      Update di;
                      lastUpdateTime(di) = currentTime;
                End If
          End for
    End If

    //check for dirty status – data being updated
    Boolean dataIsDirty = true;
    While (dataIsDirty) Do
       dataIsDirty = false;
       For each di; di ∈ C(N), i=1.. k, Do
          dataIsDirty = dataIsDirty or (di is dirty)
       End For
    End while

    Execute q;
End If


//Write Requests
If q is write request Then
    For each dj; 0 < j <= i, r(q) = { di | i∈ Iq}; dj∉C(N);
          // dj should have write permission în the node N
                                                                  1
          Select N1 using RL(dj);

          // the write request number on N is greater than on N1
          If  W(N, dj) >  W(N1, dj) Then

                If W(dj) < W_Max and cond(N, dj, write)Then
                      //increase the number of write permission replica
                      store  dj on N;
                      W(N, dj) =  W(N, dj) +1;
                      update RL(dj);

                Else If (reallocationIsOptimal
                            and cond(N, dj, write)) Then
                      //move dj with write permission from N1 to N
                      store  dj on N;
```

```
                        remove  dj on Ni
                        W(N, dj) =  W(N, dj) +1;
                        update RL(dj);
                  End If
            End If
      End For

      For each di; i∈Iq do
            For each Nk; Nk ∈ RL(N,di)and(di has read permission în Nk} Do
                  send "dirty" message
            End for
      End for

      Execute q;  // using Regular algorithm for updating în DDB
      For each di; i∈Iq do
            // q is write request; number of write operations is updated
            W(N,di) = W(N, di) + 1;
            //send last value to read replicas
            For each Nk;Nk ∈ RL(N,di)and(di has read permission în Nk) Do
                  send di
            End for
      End for
End If
```

Observations:

- The selection of the node N_l (necessary for obtaining data fragments) from the set of nodes in the fragment's replica location (RL), should take into consideration the best response time. So the selected node: *N1* would be the node with the best response time. In this way, the node *N1* capabilities and loading as the network delays are taken into consideration.
- The "dirty" messages might depend by the type of the operation: if it is a data fragment update operation the read replicas will be informed, if it is a delete or an insert operation, then the "dirty" message should involve the whole table.
- The check for *MaxUT* parameter is performed only in read requests due to the fact that the write replicas of a data are always up to date, and should be performed only for extremely critical data.
- In determining if the reallocation is optimal, the previous mathematical results can be used, simplifying the computing.

Optimizations: Determining a better cost effective node to forward the request

In the algorithms detailed in the above paragraphs, the read requests are solved in the node *N* where they were received. This fact implies transferring over the network large amounts of data (for the data fragments that were not stored in the node *N*). The step would generate network traffic and additional cost. On the other hand, keeping copies of all the data fragments in all the nodes would need large storage devices and it would not be cost effective. A better cost effective node must be found and for this, the algorithms from section "Redistributing fragments in the model" will be used.

Optimizations: Using read requests statistics

On optimization solution is based on analyzing the read requests and determining repetitive patterns in the queries and redesigning data fragments using them. For example: If most of the queries in a node *N* for table *students* refer to students with the *avgMark* < *6*, the data fragments from table *students* should be stored in different locations according with the value from the field *avgMark (*less than 6, or greater than 6*)*.

Optimizations: Using Round Robin for bulk inserts

In a database optimization process each an I/O operation is important due to the fact that it is related with the physical characteristics of the hardware and can be improved only by changing the hard components. The number of I/O operations is fixed in updating a record, but can be decreased in case of bulk inserts. The total update time is reduced when are used bulk inserts.

Also, in distributing the loading in bulk inserts operations, the Round Robin algorithm can be applied in selecting the node which will perform the insertions [Fl03, VM07].

3.2. Clearing and reallocation algorithm

The clearing algorithm verifies the statistical parameters for each data fragment and according to them decides to change the permission, to delete or to reallocate it. The central clearing and the reallocation process is a complex one, as it composed by two distinct parts/algorithms:

- A process runs a clearing algorithm on each node *N*.
- A process that runs a clearing and reallocation algorithm for the whole distributed database.

The clearing algorithm for each node verifies the status and the statistical parameters for each data fragment and according to them makes a delete decision: delete it or not. The clearing process for a node can be initiated when a set of events are raised in the node *N,* such as the expiration of the period of time set in the *P(N)* parameter, or the completion of some conditions (the available space is below a specified minimum limit).

The central clearing algorithm performs a cleaning of the database on a specific period of time (each night/each weekend). A clear distinction must be mentioned: the clearing algorithm in a node does not perform any reallocation, and does not check data statistics (number of reads/writes for a replica) against other nodes from the distributed databases. It only tries to clean up unused data fragments until the central clearing and reallocation algorithm would clean up the whole database.

The central clearing and reallocation process for the distributed database is based on a simple algorithm and has three major steps:

- Remove unused data from each node *N* in the database
- Compute statistical data for reallocation
- If the reallocation is optimal, reallocates the fragments

The clearing algorithm for each node *N* verifies the statistical parameters for each data fragment and according to them decides to delete it. The decision to delete the data when running the clearing algorithm on a node can be:

- a very rigorous one – delete the data fragments that were never used, when
  $$R(N,d_i) = 1 \text{ or } W(N, d_i) = 1$$
- or a less strict one – delete the data if the number of reads/writes is below a specific limit (the limit is set by the database administrator and is generic for the whole database):
  $$R(N, d_i) < x \text{ or } W(N, d_i) < x; \quad x \text{ is a number}$$

The decision to delete a data fragment *D* that has write permission in a node depends also by the total number of the write permissions for that data fragment, and also by the minimum number of write permissions that should exist in the database.

Clearing algorithm in a node

The delete decision for a data fragment will be taken according with the value of the *x* parameter. The database administrator can set this parameter to value 1 if he wants a very strict delete. The value of the *x* parameter must be chosen wisely as using a too small value would generate in not deleting enough data fragments, and a too big value would generate in deleting most of the data fragments from the node.

After a data fragment was deleted, a message should be sent to the nodes in the database to update their database catalog. The message for a specific data fragment d_i should be sent only to the nodes located by the $RL(d_i,N)$: N_k ; $N_k \in RL(d_i,N)$. In this manner, network traffic is avoided, as the other nodes (that do not appear in the $RL(d_i,N)$) do not have a replica for the data fragment and are not interested by it.

The clearing algorithm for a node *N* can be formalized as follows:

```
For each d_i; i∈ I, d_i ∈ C(N) Do

      // remove unread data
      If R(N, d_i) <= x and d_i does not have write permission Then
            remove d_i;
            K = { N_k | N_k ∈ RL(d_i,N)  }
            For each N_k ∈ K
                  update RL(d_i, N_k)
            End for;
      End If

      // remove unwritten data
      If W(N, d_i) <=x and W(d_i) > W_Min Then
            remove d_i;
            K = { N_k | N_k∈ RL(d_i,N)  }
            For each N_k ∈ K
                  update RL(N_k, d_i)
            End for;
      End If
End For
```

Central clearing and reallocation algorithm

The algorithm for the database should parse each node *N* and each data fragment stored on the node, to check if reallocation or removal is appropriate. Due to the massive work and cost generate by the parsing and computing, in order to optimize, a mark is attached to each data fragment so a data fragment and its replicas will be parsed only once and treated as a single entity in the central clearing and reallocation process [Ho09].

```
mark = getMark( currentTime);
For each Nj; j =1,n;

      For each di; i∈ I, di∈ C(N) Do

            //optimize parsing and continue if the data fragment was checked
            If (di has mark; i∈ I) then
                  Continue;
            End If

            //compute statistical data for reallocation -number of reads
            //and writes for a data fragment în the whole db
            compute { Nj, R(di), W(di)| i∈ I ..j=1,n}

            //number of nodes having data di with read permission
            noNodes = |{Nj | j=1,n}|

            //remove read fragments with small number of read requests
            //k' is the number of nodes where the deleting will occur
            k' = k;
            If k not percent Then
                  k' = min(k, noNodes);
            Else
                  k' = [k * noNodes ]  // full part
            End if
            Remove first k’ fragments with the smallest Rk

            If (di has write permission în Ni) Then
                  //remove with small number of write requests
                  If (W(di) = W_Max and
                      W(Ni, di) < avg{ W(Nj, di)|
                              di has write permission în Nj}) Then
                        remove di from Ni
                  End If
            End If

            //set write permission for large number of write requests
            If (W(di) < W_Max and cond(N, di, write) and
                W(Ni, di)> avg { W(Nj, di)|
                        di has write permission în Nj })  Then
                        add write permission for di în Ni
            End If

            //reallocate if necessary
            If W(di) = W_Max and cond(N, di, write) and
               W(Ni,di)> avg{W(Nj,di)|di has write permission în Nj} Then

                  //check if reallocation between node N and node Nk
                  //with the smaller number of write request is optimal
```

```
                  minWriteReq= min {W(Nj, di)| di has write perm. în Nj};

                  Nk = Nj; W(Nj, di)= minWriteReq,

                  If (reallocationIsOptimal between Nk, Ni) Then
                        reallocate di from Nk
                  End If
            End If

            //add mark to data fragment and its replicas
            For each Ni; Ni ∈ RL(N, di)
                  Add mark to di from Ni
                  R (Ni, di) = 1
                  W (Ni, di) = 1
            End for

            Add mark to di from N
            R (N, di) = 1
            W (N, di) = 1
      End for
End For
```

Additional comments and explanations:

- *compute {N_j, R_i, W_i | j =1,n; i∈.I}* has the meaning: compute for each node N_j from the database the number of read and write requests performed for a data fragment d_i. The data fragment *di* from the nodes N_j having write permission and small number of write requests in this node will be deleted (if there are no additional constraints as the total number of write permissions should not be smaller than the *W_MIN*)
- *first d_k fragments*: *k* is a model parameter and can be set to a specific number or to a percent from the total number of fragments stored in the node.
- why use *k'*? *k'* is a parameter used to compute a fixed value for *k* if *k* was set as a percent. When *k* is a fixed value, the algorithm should assure that exist *k* nodes with d_i having write permission.
- In determining if the reallocation is optimal, the mathematical results obtained earlier can be used.
- The time mark should be corelated with the central algorithm, as is not relavant the comparison of the time in two different nodes.

Optimizations in the reallocation algorithm

The clearing and reallocation algorithm represents an important part of the model and it's quite important to be optimal and to generate as less traffic as possible. In order to reduce the cost and improve the performance, the clearing algorithm should be run in a centralized manner so that a data fragment and it's replicas to be verified only once.

The optimizations included in the algorithm are:

- Central approach (explained in the above paragraph).
- Using a mark for the data replicas stored on the other nodes so they will not be verified again when the nodes are checked.
- Using the mathematical relation obtained earlier when deciding if the reallocation is optimal.

If the mark is attached to a data fragment d_{qp} located on a node N_q, it means that all the other replicas are also checked (after the computation (N_q, R_p, W_p | q =1,n; $p \in I$) step; the replicas of the d_{qp} fragment with small number of reads/ writes requests are updated and their permissions are also updated - even if they are located on a node different from N). These replicas should not be checked again when the clearing algorithm is run on the nodes where they are stored (that's why they are marked), so the cost for updating these other replicas will be zero.

$$\text{If } cost(d_{kp}) <> 0,\ k = 1,n,\ n\text{-numãr de noduri} \rightarrow cost\ (d_{fp}) = 0\ ,\ f <> k \rightarrow$$

$$C = cost(d_{11}) + \ldots + cost(d_{1p}) + cost(d_{21}) + \ldots + cost(d_{2p}) + \ldots + cost(d_{q1}) + cost(d_{qp}) + \ldots + cost(d_{n1}) + cost(d_{np}) <=>$$

$$C = cost\ (d_{11}) + \ldots\ cost(d_{1p}) + 0 + \ldots\ 0 + \ldots\ \ 0 + 0 + \ldots 0 + 0 <=>$$

$$C = |p| * c$$

So, according to the performed calculation, the total cost is influenced by the number of data fragments and by the cost of optimizing each one of them. Even if the total number of nodes does not appear in the final formula, the premise of this calculation involves it: a larger number of nodes imply a greater cost.

Constraints in the reallocation algorithm

The central clearing and reallocation algorithm has some constraints that assure proper functioning and also the smallest cost:

- A new process (clearing and reallocation) cannot be started until the current process is over (the model does not support two central clearing processes running in parallel).
- The clearing processes on each node should run prior to the central clearing process and must be over before the central process starts.
- The mark attached to a verified data fragment should be unique for each central clearing process. To ensure the fact that the mark is unique, it will depend by a unique parameter – as the central process timestamps.

The clearing algorithm on a node differs for the central one as it does not perform any reallocation. The reallocation process is quite costly due to the fact that all the other nodes in the database are requested for information related to the number of read and writes, and these requests generate traffic. That's why the clearing algorithm for a node is quite simple and efficient and it does not interfere with the rest of the nodes in the database.

Requests and clearing algorithm relation

Solving the requests processes and running the clearing algorithm process have a special relation: they can run in parallel, but it's preferable not to. The clearing algorithm is a complex process that involves all the nodes in the distributed database and consumes resources: due to the computation and the network traffic generated, the best practice is to decrease the number of requests while the clearing process is performed. Even if the optimizations included in the algorithm help by decreasing the generated traffic – and the total cost, the best policy would be to

run it when the database is not solicited. These logical assumptions would have to be proven after the model will be implemented.

4. Transactions Management

The proposed model for the distributed database is transaction oriented; it uses transactions to ensure data integrity [Gr93, Pi00, Co02]. A transaction is a series of one or more logically related SQL statements defined to accomplish some tasks [Gr93]. A distributed transaction is a transaction that includes one or more statements using data on two or more distinct nodes of a distributed database. Due to the fact that in a distributed database a data can have multiple copies and replicas, and because they should be treated as a single logical entity from a user point of view, it's essential to have a proper management of the replicas. In the case of the operations of a given transaction executing at multiple sites, the serializability argument is more difficult to specify and enforce. The complication is due to the fact that the serialization order of the same set of transactions may be different at different sites. Therefore, the execution of a set of distributed transactions is serializable if and only if [Oz08]:

- the execution of the set of transactions at each site is serializable, and
- the serialization orders of these transactions at all these sites are identical.

If a reading operation does not imply a change in the replica, an update operation must be propagated to all the replicas in all the nodes. So, the management is simpler for reading requests, as they require only that the data should not be changed until the reading transaction has been ended. The problems related to replica management are solved using blocking protocols or time-marks protocols. In locking-based algorithms there are three alternative ways of enforcing global serializability: centralized locking, primary copy locking, and distributed locking algorithm.

One side effect of all locking-based concurrency control algorithms is that they cause deadlocks. Nevertheless, the relative simplicity and better performance of locking algorithms make them more used than alternatives such as time stamp-based algorithms or optimistic concurrency control. Blocking control algorithms (used for data consistency) for centralized database can be easily extended without major changes to the distributed databases. The databases automatically use a specific blocking level – usually the less restrictive level in order to obtain the best concurrency level but to guarantee in the same time the data integrity [Ha90, Ba91].

4.1. Blocking in the model

Updating database structure

The model does not offer any special feature that would allow updating database structures while the system is still working and responding to queries. So, in order to perform a database structure change, all users' access must be restricted until the update procedure is finalized and the database runs the update procedures. Two options are valid in case of one or more disconnected nodes:

- do not update until all the nodes are reachable

- use the UpdateStructureAlgorithm presented below

The first solution is the easiest to implement, but the hardest to use, as the user might have to wait a long time to gain access to the whole database, and restoring the connection with some disconnected nodes could last for hours if not days (for example in case of errors due to hardware fail when the hardware has to be replaced and the delivery takes a day). Meanwhile the system could not be updated even if the rest of the database is running and functioning at designed parameters. This undesirable situation can be avoided using the second option: to use the recovery algorithm for updating the database structure for a disconnected node.

In the following algorithm is assumed that the system had detected the node *N1* which will be used in updating the database structure from the node *N*. The structure of a database is composed by tables, stored procedures, views, triggers, etc. and each of them should be checked for updates.

```
For each table T from N Do
   Bring T from N1
End for
For each stored procedure St, view V Do
   Bring St,V from N1
End for
For each trigger t, cursor c from N Do
   Bring t, c from N1
End for
```

The above algorithm can be further optimized: only changed objects should be updated (mostly tables and views). Also, if a view can be created by an SQL query, then it should be updated by running the query after the data fragments were updated. A method to detect a change/update is to use a check sum for the data in the node *N* and to compare against the check sum obtained from the same data in the node *N1*. Only if the check sum is different the updated values will be transferred from the node *N1* to the node *N*.

```
For each table T from N Do
   If (checksum (T,N) <> checksum(T,N1)) Then
         Bring T from N1
   End if
End for
For each view V Do
   //Qv-query corresponding to the view V
   If (checksum (Qv,N) <> checksum(Qv,N1)) Then
         Bring Qv from N1
         Run Qv
   End if
End for
```

The algorithm can be further optimized when checking and updating tables: if the check sum corresponding to the table *T* is different in the nodes *N* and *N1*, then split the table into pages and check each page check sum against the corresponding page. Only the pages that have different check sum will be updated. This approach minimizes even more the network traffic and implicit the update cost for a table. As far as simplicity is involved, this additional step is easy to implement.

4.2. Read transactions in the model

A read transaction is a transaction which does not modify any data, it only access it to retrieve the contained value [GM82]. It can be processed with general transaction processing algorithms, but most of the time it is more efficient to process read-only transactions with special algorithms in order to take advantage of the fact that the transaction only reads [Sa93, GM82]. Because of this, the read-only transactions are less expensive than the write transactions, read-only protocols are easier to implement, and the algorithms are simpler. Also, only the read-only transactions (when write transactions do not interfere) cannot generate deadlocks – for blocking it is used shared locking.

Changing the algorithms by adding new features in the proposed model, have a small impact over the transactions. In terms of performance, the impact of the proposed algorithms is small as it mostly implies additional computation, which is performed in the server that received the request and is usually less cost expensive compared to sending or receiving messages in the network. The proposed algorithms do not to interfere with the network protocols and with the part of a transaction management performed by the DBMS. The additional steps (explained in the "Request Solving Algorithm") do not interfere with the DBMS algorithms which deal with acquiring locks, receiving messages or aggregating the partial results for a request.

The problem that arises for read transactions in the proposed model is how to assure that all the relevant data was processed when resolving a request in the case when not all the nodes from the database are interrogated. To interrogate all the nodes in a distributed database is not a solution – the cost is increasing and the performance is decreasing exponentially if the number of requests increases linearly. The chances to appear bottlenecks increased a lot, as all the nodes should in fact solve each request received in each node [Al05, Al03]. This behavior is not applicable in practice, as the system would be slower than the slowest of its nodes (due to network delays) and each node is a potential bottleneck victim. The desired situation would be that the request receiving node to know which are the nodes that have relevant information and to send the request only to those nodes. Also a "dirty mark" solution can be used: when an update transaction starts, it sends the read replicas of the modified data a message containing "dirty" warning. The updated value will be send asynchronously after the update transaction has ended, and the "dirty" mark would be removed. For inserts and deletes, the "dirty" mark would be extended to the whole table.

The generic algorithm is changed by adding some steps:

- „Dirty" mark check
- Waits if needed for the updated read-only
- Deleted the „dirty" mark.

4.3. Write transactions in the model

A write transaction is a transaction which modifies the data, but it might include a part that only reads data and accesses it to retrieve the contained value. A write transaction is defined as an atomic unit of work on the database which is either completed entirely or not at all [Gr93]. A data fragment can be stored in a database in one or more locations (is replicated). When a data

fragment is updated all the replicas must be updated, and this process of updating is called "*data propagation"*. The data propagation involves capturing the state of a replica and sending it to the other replicas through the application [Do99].

The problem that arises for write/update transactions in the proposed model is how to assure that all the write replica nodes are updates and that the last version of the read replicas of the modified data will be used by the read transactions that interrogate it. To solve the first part of the problem are used distributed commit protocols such as 2PL. For the second part, a "dirty" mechanism could be used. The model is generic and the algorithm for write transactions is changed by adding additional steps:
Old Version:

- Starts the transactions by sending the write replicas preparing messages
- ….. //other steps
- Ends transaction and releases the locks

Proposed version:

- Checks if reallocation for the updated data is optimal, and reallocates if necessary.
- Sends all the nodes having read replica "dirty" marks
- Sends the write replicas preparing messages
- ….. //other steps
- Ends transaction and releases the locks
- Sends asynchronously to the read replicas the last value – the dirty mark is removed.

4.4. Consistency in the model

In the proposed model, each node sees the update operations in the same order (sequential consistency), even if this order might be different from the real order. Even if the update is performed asynchronously, the user/database administrator has the opportunity to get always the latest value for a data fragment by setting the *MaxUT* parameter. This parameter *MaxUT* (maximum update time) is used in the systems where the consistency of the data is critical and the user must be assured that is has the latest value.

4.4.1. Model correctness demonstration

Correctness in the write requests transactions

In the first part I want to focus on the correctness in the write request transactions. The model completes the database layer and that's why the update protocols in a distributed database (2PC, 3PC, and quorum, so on) are not analyzed in this paper. The write request algorithm in the proposed model enlarge the general update protocols by adding additional steps – looking for a node that has a write replica, resending the update request to that node, …, so on, but does not interfere with it. So the update protocol used in the model would be the database protocol – with all its advantages and disadvantages, and the model offers the same degree of consistency and correctness as the database update protocol does.

In case of a write request failure, the correctness is provided by the transaction's rollback process and the recovery procedures. According to the model adds, there are two main segments where a write transaction can fail – in the distributed database update protocol and in the model's additional steps. In the first case, if the transaction fails in a node involved in the distributed transaction (that performs the update) the rollback is performed by the database, and in the second case, if the node where the request was first received and then forwarded – the rollback and the recovery should be performed by a model central process. In case of a node failure – the node that initiated the write request or the node where it was resend, the procedure is detailed in the contingency paragraph.

If other types of failure appear – the transactions are rolled back and the system states become a consistent one. The Recovery paragraphs explain in detail the steps in case of failure and how the correctness and the consistency of the system are preserved.

Correctness in the read requests transactions

Due to asynchronous update of the read replicas, the value found in a specific moment of time for a data fragment might not be the last one. Due to this reason, the proposed model addresses in the first place to all the applications for which the last value is not critical, but even for those, the "dirty" mark procedures would help the user to get the latest version or at least would be notified that the data he is using is stale and soon would be changed by a write transaction. For the applications having one or two critical tables, the parameter *MaxUT* (maximum update time) was introduced. Using it, the user is certain that the data fragment was updated in the last *MaxUT* time, and it contains one if not the latest value.

4.5. Deadlock

The deadlock problem is complex even in a single system, but in a distributed system, the algorithms to detect the deadlock have a big complexity. Special algorithms as "edge-chasing", or algorithms based on a local "wait-for" graphs were developed to solve the deadlock problem in distributed databases.

4.5.1. Deadlock in the model

Not all the factors that appear in a deadlock can be eliminated, but as prove in practice, some of them (as escalation usage) affect the deadlock occurrence, and these factors should be suppressed when is possible. The best way to minimize the number of occurrences of deadlocks in the system is to take into consideration the factors that influence the appearance: for example when a page or table blocking mode is used, deadlock will appear more often than in a row blocking mode. Using escalation in the blocking process might also have an impact on the number of the deadlock occurrences. The proposed system proves again to be quite flexible as the data fragment's size can depend on the storage capacities, user's needs and might vary from row size to page size.

In finding a deadlock, the proposed model can use the algorithms already implemented. The only changes that should be implemented in the lock monitor algorithm are referring to the policy of choosing which process to be killed for releasing the necessary locks. The process that would be killed was usually chosen using priority and cost criteria. Due to the fact that in the proposed

model there are two types of locking- for reading and for writing, a new criteria must be added: killing a process depends also on the type of lock achieved (the process with read lock will be killed).

So in the proposed model there are three criteria in deciding what process should be killed:

- Process blocking type – according to the process lock
- Process priority
- Process cost.

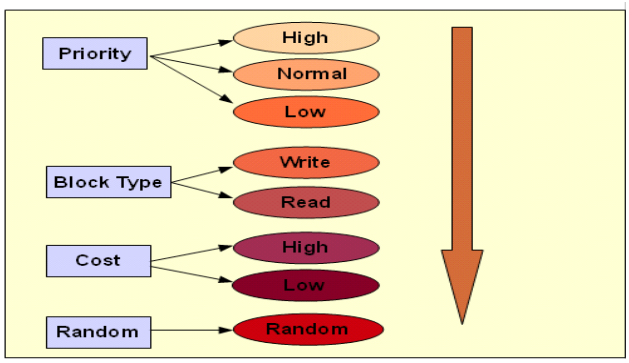

Fig. 15 Criteria visualization

5. Contingency

In this chapter are analyzed different types of errors that might appear after the model will be implemented. These are general errors, but the solution and the recovery procedures for each of them are changed to fit the requirements of the proposed model. The error causes can be divided into two main categories according to the triggering factor: human errors and system errors. The recovery is based on the error type. The errors can be grouped as follows:

- Statement failures and user process failures (An abnormal disconnection was performed by the user in the session; the user rebooted the client while connected to a database in a client-server configuration; The user's program raised an exception that terminated the session).
- User failures in updating the information from database (a user accidentally drops or truncates a table; drops an index or a stored procedure; …;a user accidentally drops one or more constraints)
- Failures caused when an instance terminates abnormally (a power outage causes the server to crash, O/S errors, etc.)
- Errors caused by media failure (A data file (on-line or archived log), or a control file is accidentally deleted, overwritten, or corrupted…)
- Failures caused by the database splitting due to connection errors (data recovery and data update should use the restored connection). In the special case when a whole subsystem is disconnected then:
    - When searching for the closest node the restored connection must be used. This fact will minimize the probability to choose a node from the same subsystem.
    - When performing an update over a data fragment, the update algorithm should consult first the database catalog and check the number of nodes with write permissions – noted with *Wp*. *AWp* can be used to note the number of available

nodes in the subsystem having write permission for the data fragment *D*. The update can be performed only if:

$$AWp >= [Wp/2] + 1;$$

- The recovery algorithm is modified as follows:

```
Select the subsystem S1 with the smallest number of nodes.
For each node N în S1 Do
      Resynchronize node N using the restored connection
End for
```

The **Recovery Algorithm for a Node *N*** can be detailed as follows, with the mention that "the closest node" is the node with the best response time. Using the best response time guarantees that in that moment the best choice in the best response time: the loading, the machine capacities, the network traffic, all are taken into considerations. The recovery algorithm for a node has the following steps (it uses the checksum to make the process more efficient):

- *Rollback started and not finalized transactions and*
- *Remove the locking on the data fragments in the node N.*
- *Select the "closest" node N1 from the distributed database*
- *Use optimized updateStructureAlgorithm to update N using N1*
- *If the number of write permission for a data fragment is greater than the maximum number of write permission allowed in the system, remove the write permission*
- *Update the other database nodes catalogs*
- *Update database catalog in the node N*

In the special case when the user's statement affects the whole database and not just a node, and an error appears- the transaction must be rolled back. In this case the structure is restored using the log file/ the last version of the database (according to each database rollback methods).

6. The efficiency of the proposed model

The number of transaction vary depending on the number and the size of the data fragments that are updated, so the number of transactions performed in a specific period of time cannot be used as a metric in determining the efficiency of the system. A better metric can be considered the number of modified bytes [Ha05]. But this last metric does not take into consideration aspects related to the machine's characteristics and performance, or to the network's features. In a system where the data is replicated, the number of replicas and their localization can also have a relevant impact in the efficiency. According to [Va07] the best response in a distributed database is obtained when there are only two copies for each data. In the proposed model, the default value for minimum write permission *W_Min* is 2. Moreover, the number of read replicas and the way they are updated is also important: their update is performed synchronously or asynchronously? To be more efficient, in the model is proposed to update the read replicas asynchronously.

In terms of performance, the most important hardware component is the storage device. A solution that uses parallel elements (disks, controllers) or advanced caching solutions as non-volatile memory has a significant effect and increases the efficiency of the system. Even if all

these factors are taken into consideration, the efficiency in the model can be deducted and not proved. The efficiency proof would be available only after the model would be implemented.

6.1. Efficiency and queries

A good system is above all, efficient. In the proposed model, I've tried to solve one of the current problems: having an efficient dynamic distributed database, having a model in which the best response time is obtained with minimum cost (in terms of transfer and storage). This goal can be achieved by paying attention to a few aspects: the load distribution, the replication of the data fragments in the nodes where are mostly used, using read request statistics or using different algorithms for read/write requests, so on. Comparing the model idea against the idea of Yohiro M. (and colleagues) described in Patent 5379424 – "*Distributed database management system for retrieving data files from databases selected based upon retrieval time"*, the model idea has the following advantages:

- there is no need that all the data from a table to be stored in a node
- if a node is too busy to satisfy a request – only the request is sent to another node and not large chunks of data. Each node in the model is capable of looking for necessary data fragments and the data fragments are sent in a node after a careful analysis (including the number of data fragments, and their size in subchapter 2.6.1) in order to minimize the network traffic.

Query optimizations

Query optimizations refer to the process by which the "best" execution strategy for a given query is found from among a set of alternatives [Va00]. An issue in this model is to determine how the optimization algorithms can be applied in a dynamic distributed database. In distributed DBMSs, two more steps are involved between query decomposition and query optimization: data localization and global query optimization. The input to data localization is the initial algebraic query generated by the query decomposition step and can be performed in the proposed model using the Replica Location information from the database catalog. The goal of query optimization is to find an execution strategy for the query which is close to optimal. The query optimizer is usually seen as three components: a search space, a cost model, and a search strategy. It should be studied how the optimization algorithms can be applied in a distributed database running the proposed model.

6.2. Comparison with BDD: Oracle RAC, Federated DB and IBM DB2 PureScale

Architecture

Oracle RAC and DB2 PureScale are distributed database, having multiple servers that solve the requests even if there is only one database storage. So, in the perspective of distributed storages, neither Oracle RAC nor DB2 are a distributed database. Only Federated DB has a distributed architecture: distributed servers and distributed storages. Oracle RAC and DB2 offer availability and scalability, as servers can be added or removed from the network and the user requests are shared between them;

The same degree of availability and scalability is offered by the model as it proposes a distributed, dynamic architecture.

Hot nodes

In Oracle RAC as in DB2, data is not partitioned on a per-node basis and the connections are routed to the least loaded node. So the hot node syndrome does not exist there. Working with the Federated Database, a DBA needs to be very careful how to partition a data to avoid creating a "hot" node. The data cannot simply be partitioned on a percentage of the database, as this would not take into consideration the distribution of queries and might cause a hot node – who would became a bottleneck.

In the proposed model, the request received by a server is forwarded only if a node that can solve it with a smaller cost can be found. Due to this suggested approach, some of the requests will be forwarded, but the reason would not be load balancing as the optimal cost. The optimizations (such as using read request statistics) help storing only the necessary data and solving faster the requests, thus minimizing the chances to appear hot nodes.

Data Backup

None of the databases does not offer by itself data backup; Oracle RAC and DB2 require backup for only storage, while FederatedDB requires backup for each storage.

As the data is replicated in the proposed model (in at least two replicas), there is no need for data backup. The number of replicas would be verified and managed by the central clearing and reallocation algorithm and by the update requests solving algorithm;

Node failure

In Oracle RAC and in DB2 PureScale, no single node is responsible for a portion of a data, so the connections to the node that failed can be automatically reconnected to the surviving nodes. On a node failure, in FederatedDB section of the data becomes unavailable to the application, and there are few real-world applications that can tolerate a segment of their data to be taken offline. So it's crucial that the failing node to be recovered as fast as possible.

The model does not propose an algorithm what would resend the queries and the connections from a falling node.

Efficiency

Oracle RAC has only a storage device, but the queries are solved in different nodes, so the request processing is split so the storage would have to take care only by the I/O operations and not the request processing. Consequently, this implies the fact that the database storage can introduce delays in request solving. In order to reduce communication between nodes in the cluster for lock management and global caching services, DB2 PureScale uses the powerHA PureScale cluster acceleration facility (here-in referred to simply as the *CF*) along with RDMA (Remote Direct Memory Access) technology to deliver transparent application scalability.

The load balancing is not a priority for the FederatedDB, if a FD server receives a request, it cannot forward it to other nodes – the requests have a special format for each server as the user must specify exactly the location of each fragment. So the same query differs if it is sent to different nodes.

The model proposes a different approach: the queries can be resend to other nodes. Also, in the proposed model the data is replicated in at least two sites and if one is heavily loaded, the necessary data can be provided by the other site – assuring a fast and efficient response. A comparison between the efficiency of the systems is hard to predict as in the proposed model the

query plans (optimization algorithms) need to be analyzed to see what changes that must be done in a dynamic database context.

Space management

Oracle server will issue an alert to forewarn the appropriate person — DBA — that the database is running out of space so that corrective measures can be taken. By default, Oracle will issue a warning alert when a table-space is 85% full and a critical alert when it is 97% full. SQL Server has the capability to monitor space usage: space usage monitoring is done using the SQL Server agent, whose primary function is job scheduling and alert handling. Thus, the monitoring and alerting mechanism is based on polling of database meta data [Ko08]. This process might be intrinsically inefficient and expensive.

In the model is proposed to start running the clearing algorithm in a node in order to obtain the necessary space when the database in that node is running out of it. Thus the human intervention is not needed as the model should be fully automated. This would ensure better performance and lower administration cost. Also the proposed central clearing algorithm would perform the clearing of the less used data, so the database fragments would be reallocated and deleted if they are not used.

Usability

Oracle RAC, DB2 offer a high degree of usability by providing a transparent environment, where the user works as in a centralized database. The proposed model should offer after implementation the same degree of usability. FederatedDB is hard to use, as the user must adapt his queries to the node/server where the query is sent.

6.3. Comparison with document BDD and with the lastest BDD releases: MckoiDDB

Comparison with document BDD

The document distributed database are special due to the fact that the number of updates is relatively small and also the accuracy of the data is not as important as in the systems that deal with money or products where the resources (stocks) are limited. Due to these reasons, these document distributed databases protocols are easier to implement and are less complex than the proposed model, as it must solve the requests in real time. Their drawbacks are the drawbacks of the static systems: the queries are not sent to the node that stores the data (and is performed additional network traffic). In the proposed model, the queries would be sent most of the time to the nodes where needed data is stored. The advantages of a document database system, (for example StrokeDB [Au08]) are specific documents facilities: documents revision control with diff/merge facilities built-in, etc., and these facilities were not taken into consideration when proposing the model in this thesis. The disadvantages of these databases are the static characteristics; the proposed algorithms in the model should provide (after running a period of time necessary to determine data access patterns and to move data fragments accordingly) the best response time and the smallest cost.

Comparison with the latest BDD

There are a set of distributed database systems and new ideas in the field [Go08], but MckoiDDB was chosen due to the fact that the version 1.0 was released in 6 October 2009. Using the data from [Mc09], the following comparison can be made:

- Dynamic scalability (the servers can be dynamically added/removed from the database)
- High availability
- Data is replicated in both models
- Protocols in write transactions: 2PL was proposed model, optimistic concurrency control is used in MckoiDDB
- MckoiDDB offers support for add-on libraries.
- Optimal dynamic distribution of the data fragments is proposed in the model.

Even if MckoiDDB is one step ahead (the first version was released), both models are still in work, and conceptually, the proposed model would offer more facilities at small cost.

7. Conclusion

This thesis proposes a method of dynamic distribution for the fragments of a distributed database and a method for dynamic allocation of the access rights for these fragments. The novelty of this model is represented by its dynamic characteristics: the data distribution is changed according to the user access patters to offer the highest availability with the smallest cost. In this aspect, a whole model is proposed: transaction processing procedures, clearing and reallocation procedures, recovery and deadlock detection & preventing procedures. Also, the cost of the model expressed in network cost, reallocation cost, execution cost, storage cost, was always a concern as the implemented model is wanted to be not only dynamic and highly configurable, but also cheap and easy to use, so it would make the database management easier and more cost-effective. The following goals were always in sight when elaborating the model's algorithms and they can be summarized:

- Dynamic reallocation of the data
- Improved reliability and availability
- Improved performance
- Minimal storage capacities and costs
- Minimal communication costs
- Dynamic storage administration

The model would offer a high degree of availability due the proposed distributed architecture and due to the variable number of servers. It should be highly scalable as it would offer alternatives when the database server runs out of capacity. Instead of replacing the server, a new server could be added to the cluster and as soon as the new instance is started, the application running on the proposed model implementation could take advantage of the extra capacity. This distributed architecture accommodates to rapidly changing businesses requirements and the resulting workload changes. In the each node of the network should be stored the most accessed data fragments- so for the most of the queries the data would be highly available and the additional network performance would be minimal. This high degree of availability would provide sql performance. In order to optimize even furthermore, optimization algorithms for

distributed databases would be applied. The load balancing in the proposed model would be performed during the request solving process (resending the query to other node or in sql operations such as bulk insert).

Database administrators face many challenges in the area of storage management. With the rapid growth in database size and the demand for up-time continuing to increase, taking the database offline to perform maintenance operations such as changing disk configuration is becoming increasingly difficult to schedule. The proposed Recovery procedures presented in the model would provide the solution for these storage management challenges by automating the entire function. All an administrator would have to do is to allocate a set of storage devices to a database instance and the model's implementation would take care of the rest. The proposed algorithms would automate the placement and the replacement of the database files and would spread data across all available storage resources to optimize performance and utilization. Also, the model would offer proactive space management by monitoring the free space on each database. If the available space would run out, the model would take the corrective measures: it would run the clearing algorithm and it would delete the rarely accessed data (with the mention that at least two copies of a data having write permission must exist in the model). Also, the amount of the data that could be deleted to gain space can be set either in terms of percentage (i.e. the $10^{\%}$ less used data fragments) or fixed (a fixed number of data fragments).

It should be easy for the database administrator to change the behavior of the model – the model would be highly configurable and would be parameterized so it would fit to any running application.

The model would offer protection against disasters and storage failures using the replication and the proposed recovery procedures: each data fragment would be replicated in at least two replicas.

The recovery procedures for a node should offer fast and complete recovery, updating the database structure, the data fragments (their values and rights), as well as the database catalogs without requiring database administrator intervention.

A simple sentence can conclude: The proposed model is not only innovative but it should also be efficient, configurable, easy to manage and cost efficient. All data would be automatically replicated over multiple servers, thus providing load balancing and fault tolerance in the event of a server failure.

The future work is to create a basic implementation of the model, and to performed simulation tests for determining the best value of the default parameters according to different access patterns.

8. References

[Ab05] Abbasi A.,*Oracle Database Administration Concepts & Implementation Made Simple,* ISBN: 0-9770739-0-4, 2005, pp.78-80

[Ab88] Abbot R., Garcia-Molina H., *Scheduling Real Time Transactions: A Performance Evaluation,*Proceedings of the 14th International Conference on Very Large Data Bases, ISBN:0-934613-75-3,1988, pp.1-12

[Al03] Alkhatib G., Labban R.S., *Transaction Management in Distributed Database Systems: the Case of Oracle's Two-Phase Commit*, Journal of Information Systems Education, Vol. 13(2), http://www.jise.appstate.edu/Issues/13/095.pdf, 2003 pp. 95-103

[Al05] Alapati S.R., *chapter Oracle Transaction Management*, *Expert Oracle Database 10g Administration*, ISBN: 978-1-59059-451-3 (Print) 978-1-4302-0066-6 (Online), 2005, pp. 225-275

[Ala07] Alapati S.R., Kim C., *Oracle Database 11g NewFeatures for DBAs and Developers,* ISBN-13: 978-1-59059-910-5, ISBN-10: 1-59059-910-1, 2007, pp. 128-130

[Au04] Mike Ault, Madhu Tumma, *Oracle 10g Grid & Real Application Clusters Oracle10g Grid Computing with RAC*, ISBN 0-9744355-4-6, ISBN13 978-0974435541, 2004, pp. 581-584

[Au08] Auvray S., *StrokeDB, Just Another Distributed Database? Not Really.,* http://www.infoq.com/news/2008/04/distributed-db-strokedb , Apr. 2008

[Ba01] Baker M., *Cluster Computing White Paper* , http://arxiv.org/ftp/cs/papers/0004/0004014.pdf, Jan 2001

[Ba91] Barghouti N.S., Kaiser E., *Concurrency control in advanced database applications,* ACM Computing Surveys, Vol, 23, No 3, ISSN 0360-0300, Sep 1991, pp 269 – 317

[Ba92] Badrinath B.R., Ramamritham K.,*Semantics Based Concurrency Control: Beyond Commutativity,* ACM Transactions on Database Systems, Vol.17, Issue 1, ISSN:0362-5915, 1992, pp.163-199

[Ba95] Balanescu T., *Corectitudinea algoritmilor,* Eds. Tehnica, ISBN 911-T-T, 1995

[Be81] Bernstein P.A., Goodman N., *Concurency Control in Distribuited Database Systems,* ACM Computing Surveys (CSUR), Vol 13. , No2., ACM 0010-4892/81/0600-0185, 1981 pp. 185 – 221

[Bi09] Birman K., *A History of the Virtual Synchrony Replication Model,* http://www.cs.cornell.edu/ken/History.pdf, 2009

[Bi87] Birman K.P., Joseph T., *Exploiting virtual synchrony in distributed systems,* Proceedings of the 11th ACM Symposium on Operating systems principles, Austin Texas, Vol. 21, Issue 5, ISSN:0163-5980, Nov. 1987, pp.123 – 138

[CaDo09] Castro P., Dockter B., Divringi L., Sundaresan S., *US Patent 7503052 - Asynchronous database API,* Mar 2009

[CaJ09] Callison J., *Database Locking: What it is, Why it Matters and What to do About it,* http://www.peakperformancetechnologies.com/; Issue of Methods & Tools, 2009

[Ca09] Callagham M., *The futures of replication in MySQL,* http://www.facebook.com/note.php?note_id=126049465932, Aug, 2009

[Ch02] Cheng C.H., Lee W.K., "*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.

[Chxx] Chapple M., *Sql Server Replication,* http://databases.about.com/cs/sqlserver/a/aa041303a.htm

[Co02] Connolly T., Begg C., *Database Systems*. ISBN-10: 0321210255, ISBN-13: 978-0321210258, 2002, pp. 572-629, 780-800

[Co07] Cox K., Marinucci T., Bevilacqua S., *Distributed Partitioned Views / Federated Databases: Lessons Learned,* http://sqlcat.com/technicalnotes/archive/2007/09/11/distributed-partitioned-views-federated-databases-lessons-learned.aspx, 2007

[Co71] Coffman E.G., Elpnick M., Shoshani A., *System Deadlocks***,** ACM Computing Surveys (CSUR)Vol. 3 ,Issue 2, ISSN:0360-0300, June 1971, pp. 67-78

[Da04] Darabant A, *Specificare şi modelare obiectuală în baze de date distribuite*, PhD Thesis, Library of Babes Bolyai University, Cluj Napoca, 2004

[Da09] Darabant A., *Proiectarea bazelor de date distribuite,* ISBN 9789731336057, Eds. Casa cartii de Stiinta, Cluj-Napoca, 2009

[De09] Delaney K., *Lock modes: Microsoft SQL Server 2008 Internals,* **Print ISBN-13:** 978-0-735-62624-9, Mar. 2009, pp. 598 - 618

[Di00] Diestel R., *"Graph Theory"*, *Springer-Verlag,* Heidelberg 2000, Electronic Edition.

[Di77] Dijkstra E.W., *Selected Writings on Computing: A Personal Perspective,* Springer-Verlag, ISBN 0-387-90652-5, 1982, pp.308-312

[Do99] Donovan R., Martin J., *High Performance Legacy Data Propagation*, DM Review Magazine, March 1999

[Dy99] Dye C,. *Oracle Distributed Systems*, ISBN 10:1-56592-432-0, ISBN 13: 9781565924321, Apr 1999

[Ed97] Edwards W.M., Hidalgo D. S., Williamson, L. A., *US Patent 5689697 - System and method for asynchronous database command processing,* 1997

[En03] Engler D., Ashcraft K. *RacerX: Effective, static detection of race conditions and deadlocks,* 19th ACM Symposiom on Operating Systems Principles, 2003.

[Es07] Esakkirajan S., Sumathi S., *Fundamentals of Relational Database Management Systems*, ISSN print edition:1860-949X, ISSN electronic edition: 1860-9503, ISBN-10: 3-540-48397-7, ISBN-13: 978-3-540-48397-7, 2007, pp.568-572, 578-586

[Fl03] Flickenger R., *Linux Server Hacks,* Print ISBN: 978-0-596-00461-3, ISBN10: 0-596-00461-3, Jan. 2003, hack 79

[Fr09] Fritcey G., Dam S., *chapter. Deadlock Analysis, SQL Server 2008 Query Performance Tuning Distilled,* ISBN-13: 978-1-4302-1902-6, ISBN-13(online): 978-1-4302-1903-3, 2009, pp. 401- 414

[Ga03] Garmany J., Freeman R.G., Burleson D.,*Oracle Replication Snapshot, Multi-Master and Materialized Views Scripts*, ISBN 0972751335, 9780972751339, 2003, pp. 16-20

[Gh03] Ghemawat S., Gobioff H., Leung ST., *The Google File System,* labs.google.com/papers/gfs-sosp2003.pdf, 2003

[GM82] Garcia-Molina H., Wiederhold G., *Read-Only Transactions in a Distributed Database,* ACM Transactions on Database Systems (TODS) Vol. 7 , Issue 2,ISSN:0362-5915, 1982, pp. 209 – 234

[Gr03] Graham J., "*Efficient Allocation in Distributed Object Oriented Databases"*, Proceedings of the ISCA 16th International Conference on parallel and Distributed Computing Systems , Reno Nevada, August 2003, pp150-154.

[Gr05] Gray J., van Ingen C., *Empirical Measurements of Disk Failure Rates and Error Rates,* Microsoft Research Technical Report MSR-TR-2005-166, http://research.microsoft.com/pubs/64599/tr-2005-166.pdf, Dec. 2005

[Gr93] Gray J., Reuter A., *Transaction Processing:Concepts and Techniques*, Morgan Kaufmann Publishers,Inc. ISSN 1046-1698, 1993, pp. 426-428

[Gr96] Gray J., Helland P., O'Neil P., Shasha D., *The Dangers of Replication and a Solution,* Proceedings of the 1996 ACM SIGMOD international conference on Management of data, ISBN:0-89791-794-4, 1996, pp.173-182

[Go08] Gold B.T., Ailamaki A., Huston L., Falsafi B., *Accelerating Database Operators Using a Network Processor*, White Papers, http://www.silicon.com/white-

papers/components/2008/12/04/accelerating-database-operators-using-a-network-processor-60502340/ , Dec. 2008

[Ha05] Harris, T., Marlow, S., Jones S. P., and Herlihy, M., *Composable Memory Transactions*, ACM Conference on Principles and Practice of Parallel Programming 2005 (PpoPP'05). 2005, pp. 48-60.

[Ha06] Hall T., *Oracle PL/SQL Tuning. Expert Secrets for High Performance Programming,* ISBN: 0-9761573-9-X, ISBN 13: 978-0976157397 , 2006, ch. 3

[Ha90] Haritsa J.R., Carey M.J., Livny M., *Dynamic Real Time Optimistic Concurrency control,* Proceedings of ACM PODS 1990,

[He03] Herlihy M., Luchangco V, Moir M, and Scherer III W., *Software Transactional Memory for Dynamic-Sized Data Structures*. Proceedings of the Twenty-Second Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC) 2003, pp. 92–101.

[He07] Hernandez G., *Locking*, http://www.georgehernandez.com/h/xComputers/Databases/SQLServer/Locking.asp, 2007

[Ho08] Horvat M., *Reallocation in Dynamic Distributed Database Model*, Proceedings of the National Conference ZAC 2008, (Zilele Academice Clujene 2008), ISBN: 978 973 610 730 6, 2008, pag 171-180,

[Ho09] Horvat-Petrescu M., *Optimal dynamic distribution in a distributed database,* Information Technologies' 2009 -15th International Conference on Information and Software Technologies. Research Communications/ Kaunas University of Technology. Kaunas: Technologija, ISSN 2029-0039, 2009, pp. 159 - 165

[HoSm05] Ho A., Smith S., Hand S., *Technical Report Number 633, On deadlock, livelock, and forward progress*, ISSN 1476-2986, May 2005

[Hoxx] Holliday, J.A., Abbadi A., *Distributed Deadlock Detection,* Encyclopedia of Distributed Computing, Kluwer Academic Publishers, accepted for publication.

[Hp03] *TPC Benchmark C Full Disclosure Report for hp Integrity rx5670 Cluster 64P Using Oracle Database 10g Enterprise Edition with Real Application Cluster and Partitioning; and Red Hat Enterprise Linux AS 3,* http://www.tpc.org/results/FDR/TPCC/HP%20Integrity%20rx5670%20Cluster %2064P_FDR.pdf, 2003

[Hu01] Huang Y., Chen J., *"Fragment Allocation in Distributed Database Design"*, Fragment Allocation in Distributed Database Design, Journal Of Information Science And Engineering, 17, 2001, pp. 491-506

[IB09] IBM White Paper, *Transparent Application Scaling with IBM DB2 pureScale*, ftp://ftp.software.ibm.com/software/data/sw-library/db2/papers/db2-pure-scale-wp.pdf, Oct. 2009, pp.4 - 10

[Ja09] Jacobs K., *InnoDB.Innovative Technologiesfor Performance andData Protection,* MySQL Conference, April 2009

[Ju08] Jula H., Tralamazza D., Zamfir C., Candea G., *Deadlock Immunity: Enabling Systems To Defend Against Deadlocks,* Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Dec 2008

[Ju98] Jung I., Lee J., Moon S., *Concurrency control in multidatabase systems: A performance study,*Journal of Systems Architecture: the EUROMICRO Journal, ISSN:1383-7621, Oct. 1998, pp. 97 - 114

[Ka01] Kaveh N., Emmerich W., *Deadlock Detection in Distributed Object Systems,* Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, ISBN: 1-58113-390-1, 2001, pp.44-51

[Ka07] Karanja E., Badamas M.A., *An organizational data replication model for mobile databases.,* Journal of Academy of Business and Economics, ISSN: 1542-8710, Mar 1, 2007, 122-129

[KaSi07] Kaur N., Singh R., Misra M., Sarje A. K., *Secure Transaction Management Protocols for MLS/DDBMS,* Springer Berlin / Heidelberg, ISSN: 0302-9743 (Print) 1611-3349 (Online), ISBN: 978-3-540-77085-5, 2007, pp.219-233

[Ke01] Keating B., *Challenges Involved in Multimaster Replication,* Database Specialists, Inc., http://www.dbspecialists.com, 2001

[Kn87] Knapp Edgar, *Deadlock detection in distributed databases,* ACM Computing Surveys (CSUR) Vol. 19 , Issue 4, ISSN:0360-0300, Dec. 1987, pp.303-328

[Ko08] Sergey Koltakov, Mughees Minhas, *Technical Comparison of Oracle Database 11g and SQL Server 2008: Focus on Manageability,* http://www.oracle.com/technology/products/manageability/database/pdf/competitive/ss_2008_vs_oracle_11g_tech_comparison.pdf, Dec. 2008

[KoHe08] Koskinen E., Herlihy M. *Deadlocks: Efficient deadlock detection.,* Proceedings of the 20th ACM Symposiom on Parallelism in Algorithms and Architectures, 2008.

[KoMi08] Koltakov S., Minhas M., *Technical Comparison of Oracle Database 11g and Sql Server 2008: Focus on Manageability. An Oracle White Paper.*, http://www.oracle.com/technology/products/manageability/database/pdf/competitive/ss_2008_vs_oracle_11g_tech_comparison.pdf, Dec. 2008

[Kr07] Kroenke D. M., David J. A. *Database Concepts. 3rd ed*. New York: Prentice, ISBN-10: 0131986252, ISBN-13: 978-0131986251, 2007

[Ku09] Kumar A., *Pros and Cons of Distributed Databases,* http://www.articlealley.com/article_911011_11.html, Jun. 2009

[La98] Lamport L., *The part-time parliament.* ACM Transactions on Computing Systems Vol. 16, Issue 2, ISSN:0734-2071, May 1998, pp. 133 – 169

[Le85] Leung C. H. C., Wolfenden K., *Analysis and Optimisation of Data Currency and Consistency in Replicated Distributed Databases,* The Computer Journal, Vol 28, No. 5, 1985, pp. 518-523

[Lu06] Lungu I., Fodor A.G., *"Optimizing Queries in Distributed Systems"*, Revista Informatica Economica nr. 1 (37), 67-72, 2006.

[Ma01] Markus T., Moroşanu C., Varga V.: *Stochastic Query Optimization in Distributed Databases using Semijoins*, Annales Universitatis Scientiarum Budapestinensis de Rolando Eotvos Nominatae Sectio Computatorica 20, 2001, pp. 107-131.

[McPr07] McElroy P., Pratt M., *Oracle Database 11g: Oracle Streams Replication,* http://www.oracle.com/technology/products/dataint/pdf/twp_streams_replication_11gr1.pdf , July 2007

[Mc07] McGehee B., *SQL Server Federated Database Performance Tuning Tips,* http://www.sql-server-performance.com/tips/federated_databases_p1.aspx, Mar. 2007

[McSt07] McRobb S., Stahl B.C., *Privacy as a shared feature of the e-phenomenon: a comparison of privacy policies in e-government, e-commerce and e-teaching,* http://www.ccsr.cse.dmu.ac.uk/jpapers/papers/Mcrobb_2007_ijitm.pdf, International Journal of Information Technology and Management, Vol. 7, 2007, pp. 232-249

[Mc09] Mckoy Distributed Database, *Introduction,* http://www.mckoi.com/index.html, http://www.mckoi.com/Introduction.html, 2009

[Mi08] Michalewicz M., Newlan P., Lundhild B., *Oracle Real Application Clusters 11g, Technical Comparison with Microsoft SQL Server 2008, An Oracle Competitive white Paper*,
http://www.oracle.com/technology/products/database/clustering/pdf/twp_racsqlserver_2008.pdf, 2008

[Mo08] Moldovan G., Valeanu M., *The performance optimization for date redistributing system in computernetwork*, International Journal of Computer, Communications & Control, ISSN 1841-9836, E-ISSN1841-9844, Vol III, Supl. issue, 2008, pp. 116-118.

[Mo93] Mosberger D., *Memory Consistecy Models,* ACM Operating Systems Reviews, Vol. 27, Issue 1, 1993, pp.18-26

[MS5.1] *MySQL 5.1 Reference Manual:: 16 Replication,* http://dev.mysql.com/doc/refman/5.1/en/replication.html

[Mu05] Muntean C., *Using the database replication technology for the jet engine,* Revista Informatica Economică, nr. 1(33)/2005 , pp. 108-111

[Na02] Nagarkar N., *Autonomous and Distributed Transactions in Oracle 8i/9i*, Database Journal, http://www.databasejournal.com/features/oracle/article.php/1550951/Autonomous-and-Distributed-Transactions-in-Oracle-8i9i.htm, Dec, 2002

[Ne09] Newman H., *RAID's Days May Be Numbered,* http://www.enterprisestorageforum.com/technology/features/article.php/3839636,Sep. 2009

[OB08] O'Brien J., Marakas G.M., *Management Information Systems,* ISBN: 0073511544, 2008, pp.185-189

[Oi07] Oiaga M., *Microsoft Takes Peer-to-Peer to the Next Level with a New Synchronization Platform,* http://news.softpedia.com/news/Microsoft-Takes-Peer-to-Peer-to-the-Next-Level-with-a-New-Synchronization-Platform-70009.shtml, Nov. 2007

[Or8i01] *Replication Overview,* Oracle8*i* Replication Release 2 (8.1.6)**,** Part Number A76959-01,
http://www.mscd.edu/~ittsdba/oradoc817/server.817/a76959/repover.htm#14080, 2001

[Öz08] Özsu M.T., Valduriez P., *Distributed and Parallel Database Systems,* http://www.silicon.com/white-papers/server-hardware/2008/04/10/distributed-and-parallel-database-systems-60308215/, Apr. 2008

[Öz94] Özsu M.T., Valduriez P., Dayal U., Distributed Object Management, ISBN-13: 9781558602564, ISBN: 1558602569, 1994, pp.212 – 231

[Öz99] Özsu M.T., Valduriez P., *Principles of Distributed Database Systems,* Prentice Hall, ISBN 0-13-659707-6, 1999

[Pa06] Pavlin I. *Transaction and Concurrency in Oracle Server,* http://silicondetector.org/display/ds/Transaction+and+Concurrency+in+Oracle+Server, 2006

[Pa92]Pang H., Livny M., Carey M.J.,*Transaction Scheduling in Multiclass Real-Time Database Systems*, Sept 1992

[Pi00] Piattini M., Diaz O., *Advanced Database Technology and Design*, ArtechHouse Publishing, ISBN-10: 0890063958, ISBN-13: 978-0890063958, ISBN:0890063958, 2000

[Pu91] Pu C., Leff A., *Replica Control in Distributed Systems: An Asynchronous Approach* Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data, ISBN ISBN:0-89791-425-2, 1991, pp 377-386

[Ra04] Ramamritham K.,Sang H.S., DiPippo L.C., *Real-Time Databases and Data Services,* International Journal of ISSN:0926-8782 ,Vol. 28, Issue 2-3 , pp. 179-215

[Ro03] Rothschild M., *US Patent 6567823 - Change propagation method using DBMS log files,* http://www.patentstorm.us/patents/6567823-fulltext.html, May 2003,

[Ro09] Rob P., Coronel C., *Database systems: design, implementation and management*, ISBN-13: 978-1-4239-0202-0, 2009

[Sa93] Satyanarayanan O.T., Agrawal, *Efficient execution of read-only transactions in replicated multiversion databases,* IEEE Transactions on Knowledge and Data Engineering,Vol. 5, Issue 5, ISSN:1041-4347, Oct 1993, pp: 859-871

[ScGi07] Schroeder B., Gibson G.A., *Disk Failures in the Real World: what does an MTTF of 1,000,000 hours Mean to You?* Proceedings of the 5th USENIX Conference on File and Storage Technologies, 2007

[Sc07] Schlichting D.,*What's New in SQL Server 2008*, Database Journal, http://www.databasejournal.com/features/mssql/article.php/3702381/Whats-New-in-SQL-Server-2008-Part-3.htm, Oct.2007

[Sc08] Schwartz B., Zaitsev P., Tkachenko V., Zawodny J.D., Lentz A., Balling D, *High performance MySQL (second edition)*, ISBN: 980-0-596-10171-8, June 2008, pp. 262 – 264

[Sh07] Shi J.Y., *Why Synchronous Parallel Transaction Replication is Hard, But Inevitable?A White Paper,* CTO Parallel Computers Technology Inc. (PCTI), 2007

[Si05] Sitar-Taut D., *Baze de date distribuite*, ISBN 9736510380, 2005, pp210-215

[Sl07] Sleit A., AlMobaideen W., Al-Areqi S., Yahya A., "*A Dynamic Object Fragmentation and Replication Algorithm In Distributed Database Systems",* American Journal of Applied Sciences 4 (8)*,* 2007, pp. 613-618

[St07] Stahl B.C., *Reflective responsibility for risk: a critical view of software and information systems development risk management,* http://www.ccsr.cse.dmu.ac.uk/jpapers/papers/2007_reflective_resp_risk_IJRAM.pdf, International Journal of Risk Assessment and Management, Vol. 7, No. 3, 2007, pp. 312-325

[St09] Steinberg D.H, *Asynchronous and Synchronous,* http://weblogs.java.net/blog/2003/10/24/asynchronous-and-synchronous*,* 2003

[Ta03] Tambulea L., *Baze de Date*, 2003.

[TaHo08] Tambulea L., Horvat M., *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

[TaLHo08] Tambulea L., Horvat M., *Redistributing Fragments into a Distributed Database*, Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844, Vol. III (2008), No. 4, pp. 384-394

[Th05] Thomas B., *Solutions for Highly Scalable Database Applications. An analysis of architectures and technologies,* http://download.microsoft.com/download/a/4/7/a47b7b0e-976d-4f49-b15d-f02ade638ebe/OracleRAC.pdf , Performance Tuning Corporation, 2005

[Th09] Thalman L., Kindahl M., *New Replication Features*, http://assets.en.oreilly.com/1/event/21/New%20Replication%20Features %20Presentation.pdf, Apr. 2009

[Tu09] Tuuri H., Sun C., *InnoDB Internals: InnoDB File Formats and Source Code Structure,* MySQL Conference, April 2009

[Ul03] Ulus T., Uysal M., *"Heuristic Approach to Dynamic Data Allocation in Distributed Database Systems",* Pakistan Journal of Information and Technology 2 (3), 2003, pp. 231-239.

[Up08] Upadhyaya S., Lata S., *"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.

[Va00] Varga V., *"Optimal Strategies for Query Processing in Distributed Databases*", Teza de Doctorat, Biblioteca Universitatii Babes Bolyai, Cluj Napoca, 2000

[Va06] Varga V., *Interogarea bazelor de date distribuite*, Casa Cartii de Stiinta, Cluj-Napoca, ISBN 973-686-842-7, 2006.

[Va07] Vasileva S., Milev P, Stoyanov B., *Some Models of a Distributed Database Management System with Data Replication,* International Conference on Computer Systems and Technologies – CompSysTech'07, http://ecet.ecs.ru.acad.bg/cst07/Docs/cp/SII/II.12.pdf, ISBN:978-954-9641-50-9 , 2007

[VM07] VMware technical note, *Round-Robin Load Balancing,* http://www.vmware.com/pdf/vi3_35_25_roundrobin.pdf, Dec. 2007

[Wa08] Waldron T., *OS/2's Symmetrical Multiprocessing Demystified*, http://www.edm2.com/index.php/OS/2%27s_Symmetrical_Multiprocessing_Demystified, July 2008

[We02] Weikum G., Vossen G., *Transaction Information Systems Theory, Algorithms, and Practice of Concurrency Control and Recovery*, ISBN 1558605088, 2002

[We10] Werner V., *Choosing Consistency,* http://www.allthingsdistributed.com/, 2010

[Wi05] Williams A., Thies W., Ernst M.D. *Static deadlock detection for Java libraries*. 19th European Conference on Object-Oriented Programming, 2005.

[Wo92] Wolfson O., Jajodia S., *"An Algorithm for Dynamic Data Distribution",* Proceedings of the 2nd Workshop on the Management of Replicated Data (WMRD-II), Monterey, CA, Nov. 1992.

[Yo08] Young S.T., Givens M., Gianninas D., *Adobe® AIR™ Programming Unleashed, chapter Synchronous Versus Asynchronous Database Operations,* ISBN-10: 0-672-32971-9, Print ISBN-13**:** 978-0-672-32971-5, Nov.2008, pp. 145 – 151

[Yo95] Yojiro M., Sekiguchi K., Muranaga M., Kato N., *US Patent 5379424 - Distributed database management system for retrieving data files from databases selected based upon retrieval time,* http://www.patentstorm.us/patents/5379424.html*,* 1995

[Xi04] Xiong M.,Agarwal S.,*SQL Server 2000 Incremental Bulk Load Case Study,* http://technet.microsoft.com/en-us/library/cc917716.aspx*,* Sept 2004