IMPLEMENTATION OF FRAGMENTATION AND REPLICATION METHODS IN DISTRIBUTED SYSTEMS

Anca-Georgiana Fodor [1*]
Ion Lungu [2]

ABSTRACT

One of the most important elements of distributed systems is data distribution across the entire informational system. In order to benefit from a distributed system data architecture advantages, we need to create and implement such a data distribution design to be sure that we maximize the local data processing and minimize the cost of communication between sites of distributed system. Data distribution consists in three essential steps: data fragmentation, allocation and replication of data fragments to system's sites. The decision of fragmentation differs from one system to another and there are a lot of ways to implement the data fragmentation. The key element is to find a solution which minimize the data selection and transfer between sites and minimize the access to disk, through a proper database indexing. This paper presents activities and steps to implement fragmentation and replication methods for building distributed database systems.

KEYWORDS: *distributed systems, fragmentation, replication, allocation, implementation activities.*

1. INTRODUCTION

We are living in the Digital Age, in which the proper management capabilities, the optimization of cost base, the fast decision and the proper answer to the customer creates a great customer experience and the eligibility to stay in business.

The nowadays competition in business markets is significantly high and only those companies with a high level of optimization can survive and grow.

An essential component for business sustainability is the IT system of the company and its "time to yes". The Core IT system should contain all necessary information to support management to take documented decisions and has low operating cost.

A suitable option for an IT Core System for a company might be a distributed system which has the advantage of local processing and the diminishing of the communication costs. All those are obviously coming with a price – the increase of system complexity due to data distribution across the sites of the IT system.

[1]* corresponding author, Ph. D. at the Romanian-American University of Bucharest, anca.fodor@yahoo.com
[2] Professor., Ph. D. at the Academy of Economic Studies of Bucharest, ion.lungu@ie.ase.ro

2. DATA DISTRIBUTION METHODS

Data distribution consists in three main activities: data fragmentation, allocation of data fragments to the sites and data replication.

There are some ***reasons*** to do the fragmentation of the database in distributed systems [OV95]:

- **Usage**: user's apps are working with views rather than entire tables, then it is useful to split tables on small fragments based on utilization;
- **Efficiency**: the data storage is done in the place proximity where the specific data are used; the data which are not used by local apps are not stored. In this approach, the response time of the local database to local queries is very low due to usage of local stored data on the one hand and due to small dimension of the local database which allow it to find the data very fast;
- **Parallel processing**: by having the database split in many fragments, there is the possibility to split a transaction in many subqueries and run it in parallel on different fragments and increase the concurrency level of the system;
- **Security:** we store to a site only locally necessary fragments of database, then other fragments of the database are not available for unauthorized user.

3. DATA FRAGMENTATION METHODS

The "*data fragmentation*" concept is similar with "*data partitioning"*. Data fragmentation is a fundamental characteristic of database distributed systems.

For example, Oracle use the terminology of data fragmentation to present the data stored in discontinued area of memory. The decomposition of Global Database Schema into fragments/partitions could be realized by using three main fragmentation principle: horizontal fragmentation, vertical fragmentation, hybrid fragmentation [OV95], [CP85], [RV13].

A. Horizontal Fragmentation consist in tuples partitioning of one global data collection in smaller subsets based on some selection predicates.

The goal of horizontal fragmentation is to obtain smaller fragment to maximize the local processing of queries, then the fragments are stored to the sites where they are used.

The selection predicate is associated to each fragment and define the property on which is based the grouping of records who compose this specific data fragment. The horizontal fragments obtained have the similar structure with the global relation from which were extracted, however those fragments will contain different tuples. In general, the horizontal fragments are disjoint.

There are two subtypes of horizontal fragmentation of a global relation*: primary horizontal fragmentation* and *derived horizontal fragmentation*.

In the *primary horizontal fragmentation* the tuples selection of a relation is based on the attribute's properties of the respective relation.

At horizontal fragmentation of a global relation we have to follow the completeness rule.

This subtype of fragmentation is obtained by using the selection operator applied to the global relation and it is represented as below:

$R_i=\sigma_p(R)$, where:

σ – selection operator;
p – fragmentation predicate;
R – global relation;
R_i – the horizontal fragment as result of fragmentation.

The reconstruction of the global relation R can be done using the reunion of the fragments as below:

$$R = \bigcup_{i=1}^{n} R_i$$

The *derived horizontal fragmentation* is realized based on horizontal fragmentation of a different global relation and it can be done if there is a binary correspondence with the table which is going to be fragmented.

The resulted fragments are obtained by combining the R relation's tuples with horizontal fragments of the table S. After the fragmentation, we end up with R_i fragments.

$R_i = R \triangleright S_i$, where:

▷ - semi join operator;
z - maximum number of fragments to be defined on relation R;
S_i – horizontal fragments on S as following: $\sigma_{Fi}(S)$.

B. Vertical Fragmentation of a relation R consists of grouping together of attributes which are used by some apps. The resulted vertical fragments contain groups of different attributes but having the same tuple. Each fragment resulted has to contain the primary key in order to be able to reconstruct the global relation R. The vertical fragmentation specific operator is projection.

There are two ways for vertical fragmentation: by attributes partitioning or by attributes grouping.

The rule by attribute partitioning fragmentation is that we have to have attributeds to keep the link between fragments, then the resulted fragments will be distinct.

On the other hand, for attribute grouping fragmentation, the resulted fragments have to have at least one common attribute. This is impacting the update process which become more dificult due to this necesary redundancy.

Let it be the R relation having the attributes $\{A_1, A_2, ..., A_n\}$. The representation of vertical fragmentation based on attributes set $\{A_j, A_{j+1}, ..., A_k\}$ should be written as following:

$R_i = \prod_{A_j, A_{j+1}, ..., A_k.} (R)$, 1≤j≤n, 1≤k≤n, where

Π – projection operator;
A_j, A_{j+1},....., A_k – fragmentation attributes set;
R_i – resulted vertical fragment.

The reconstruction of the global relation R it is realized by joining the resulted fragments, as following:

$R=R_1 \rhd\lhd R_2 \rhd\lhd ... \rhd\lhd R_m$

An optimal vertical fragmentation is the one in which an application is accessing only one fragment locally stored. Those apps which need data from two or more vertical fragments – stored in different sites – are slower than expected because they will request many join operation between fragments across many sites.

C. Hybrid Fragmentation.

Horizontal or vertical fragmentation of a relation R do not necesary answer properly to an app need of data. In those cases it can be used a mixt fragmentation.

The mixt fragment can be obtained by vertical fragmentation of an horizontal fragment of a relation R or by horizontal fragmentation of a vertical fragment of the relation R. This fragmentation type can be achieved by using the selection and projection oprators as following:

$\sigma_p(\pi_{a1,...an}(R))$ or

$\pi_{a1,...an}(\sigma_p(R))$

where p is a predicat based on one or many attributes of the relation R.

There are some clear *fragmentation rules* that we must follow to have a proper database fragmentation [OV95]:

- **Completeness**: when a table R is decomposed in many fragments R_i, i=1,2,…,n, then each article of the table R must be present in at least one of the fragments R_i. This rule is guarantees that we are not loosing data during the fragmentation activity ;
- **Reconstruction**: it must be possible to define a relational operation to reconstruct the entire table R from the fragments. This rule assure that we are keeping the functional dependencies;
- **Disjunct nature**: if an articol of the d_i of the table R is part of the fragment R_i, ∀i=1,2,…,n, then the same article mustn't appear in other fragment. There is only one exception of this rule: in the vertical fragmentation we have to have the primary key in each fragment of the table in order to be able to reconstruct the entire table. With this rule the minimum and necessary data redundancy is assured.

4. DATA REPLICATION METHODS

From data management point of view, we can store data in one place/site or we might have many copies of data replicated in different sites. There are two data replication methods: *complete replication method* and *selective replication method*.

Complete replication method consist in complete duplication of database in any site of the distributed system. In this case, the reliability, availability and system performance are maximum, however the storage, update and communication costs are very high.

Selective replication method means to duplicate only a subset of data necessary to some specific sites. It is preferably to replicate only those data which are updated very rarely. The goal of this method allows us to minimize the costs and maximize the performance. This is the most used replication approach, given its flexibility and advantages.

Replication methods used by various DBMS can be, for Oracle: *with real-time access to a remote database (linked databases); a read-only copy of a table stored to the locally site; an updatable copy of a table stored to locally site; asynchronous multi-master replication; synchronous multi-master replication* [OR05].

The *real-time access to a remote database,* using linked tables is appropriate when the source table needs to be accessible to a remote site in real time.

The *read-only copy of a table stored to the locally site* using a read-only snapshot is used when a delay in snapshot update is acceptable, we can use locally the almost final version of the snapshot data and we never need to update it.

Using *an updatable copy of a table stored to locally site* (updatable snapshot) is an efficient method to horizontally partition updatable data.

The *asynchronous multi-master replication* is recommended when a table needs to be accessed from various sites. The updates are disseminated after a time slot.

In case of *synchronous multi-master replication,* the synchron replication allows the simultaneous execution of transactions to all participant sites. This method is recommended when all sites needs info from this specific table in real time and all sites is updating this table.

5. IMPLEMENTATION OF DISTRIBUTION METHODS

By analyzing the domain's publications, we consider that distributed database implementation should be done by applying the above fragmentation and replication methods and by having the following activities:

- Analysis of distribution's requirements and constraints;
- Fragmentation modelling;
- Replication modelling;
- Allocation modelling;
- Activities for query processing.

A. Analysis of distribution requirements and constraints

We consider that the process of analysis requirements should follow three steps: identify the users, locations and activities; identify the constraints; build the global database dictionary.

Identify the users, locations and activities is essential for a distributed system and is the moment when the first overview of the system complexity is outlined. Basically for an activity we will need a local server on each subsidiary of the company and for the Headquarter we will need, in addition, to centralize the info from all local servers.

Local databases for subsidiaries will store the data locally necessary and all those data will be sent to the central server to allow the access from any local server. By using the replication, the business continuity and system fast response is guaranteed for all requestors across the company system.

Identifying the constraints is very important as distributed systems are more complex than centralized ones and have many additional constraints. All referential keys and integrity constraints became more complex in a distributed environment. There are four types of relations between tables which will be part of a distributed system: objects with strong referential links; objects with medium referential links having a high data volatility; objects with medium referential links having a medium data volatility; objects with low referential links. Ideally the data objects with strong referential links should be stored together and data with high volatility, often updated, would better be stored at the same site who initiate most of the updates.

Global database dictionary catalog contains information about all objects within a distributed scheme and it should be accessible from any of system sites. In addition to a centralized catalog, this one store the info regarding distribution. The most important elements of a global database dictionary are: global scheme info (global collections names and characteristics), info regarding fragmentation, allocation info (links between fragments and physical images of global collections), data access info and statistics.

There are some key elements regarding **global database catalog**: the location of global database catalog, objects names, local data dictionary and integrity constraints control between databases.

The *location of global database catalog* should be set in such manner to make it accessible for all network sites in the same way. It might be a *centralized catalog* with the advantage of lower storage space and low complexity, however this option do not comply with distributed database rules. Another option is to have a *replicated* catalog to all sites with the advantage of fast answers, however this option is not comply with site autonomy principle and also come with the necessity to update all sites in case of any local update which can be costly. The local catalog option has main advantages database distribution, but in case of accessing a remote object, the searching time will be done “in blind” and then this is not a reliable solution. In practice, a functional solution is to store in each local catalog information about remote objects and mask the real location with synonyms. This is the ORACLE approach [OR05] and the real location of database objects are managed by DBAs.

Object names used by Oracle [OR05] are table_name@address masked by synonyms. In this manner the users are referring objects using only the synonym of the object, in this way we respect the principle of location transparency in a distributed database. The responsibility of managing tables and synonyms and the proper usage as well is left to the DBAs and apps developers.

Regarding *local data dictionary,* this should be optimized for local processing as main activity should be covered to the local site. In addition, in order to be able to access the remote objects, we have to have stored in the local dictionary the information about those objects, their address being mandatory. It would be helpful that we store some statistics regarding the volume and distribution of remote objects for an efficient query processing.

Talking about *integrity constraints between databases*, there are two ways to practical implement it to a remote database, according to Oracle approach: either we can connect and communicate with remote database when we need to update a local database to be able to validate any update, based on our procedures and functions, or we can maintain in the local database a copy of remote table to be used for all necessary validation for keeping the data consistency.

B. Fragmentation Modelling is the phase of distributed database design and implementation in which we split the global distributed database scheme in partitions which will be allocated to network sites. Database fragments are allocation units for distributed databases and should be homogenous from apps query usage pointy of view. We will store to a specific site only the locally necessary data records to run the local apps, all other will just increase the storage usage and the storage costs.

The fragmentation solution is personalized for each distributed system and is aiming to find a solution for high performance to low costs. There are tables in the global database schema which are rarely updated and often read, for example tables with localities codes, professions catalog, country codes list, etc. which are usually international or national standards. Those tables won't be fragmented and will be fully replicated to all sites.

The key element on fragment modelling is the set of queries executed on each site and the set of tuples and attributes part of those queries and their execution frequency. In other words, first we need to identify the most accessed data fields and the type of necessary access – read only, read-write. Based on Paretto rule, 80%-20%, if we store locally 80% of accessed data we will have a good performance at an acceptable cost of communication and disk access. It would be helpful to have as well a proper indexing scheme for a faster disk access.

Usually we have mixed fragmentation, based on set of «most accessed fields », however we might have as well horizontal or vertical type of fragmentation in some cases.

C. Replication Modelling

In practice, in the phase of replication modelling, there are implemented usually read-only snapshots or updatable snapshots. For practical reason, we have choosen the Oracle approach, according to [OR05].

To create ***read-only snapshots*** we have to identify the tables stored on the "master" site which we need to replicate on locally site. Then we have to set up a database link between the site which will store the read-only snapshot and the "master" site, as following:

```
CREATE PUBLIC DATABASE LINK address
USING connection_name;
```

If we are using a connection through TCPIP protocol, then the TNSNAMES.ORA file will contain the connection description, as below:

```
connection_name = (DESCRIPTION =
                        (ADDRESS =
                                (PROTOCOL = TCP)
                                (HOST = host_name)
                                (PORT = 1521)
                                (CONNECT DATA =
                                (SID = REMOTE)))
```

This connection to the central database will be used by other operations, then, in order to create the data transparency, we need to mask the connection by using a synonym:

```
CREATE SYNONYM synonym_name
FOR table_name@connection_name
```

After we create the database link and the synonym, we can create the read-only snapshot on the locally site, taking into consideration the snapshot physically allocation, refresh intervals and some grouping clauses used in the locally queries planned to be executed on the snapshot. Those grouping clauses will be used as well on the snapshop creation selection clause.

```
CREATE SNAPSHOT snapshop_name
PCTFREE 0 PCTUSED 99
STORAGE (INITIAL 100K NEXT 100K PCTINCREASE 0)
TABLESPACE SNAPS
REFRESH [FAST | COMPLETE]
START WITH start_date
        NEXT NEXT_DAY (TRUNC (start_date,'format')) +hour/24
        WITH PRIMARY KEY
        AS
        SELECT……;
```

In addition, we need to set the automatically refresh mechanism using 2 parameters stored into INIT.ORA file: JOB_QUEUE_PROCESSES which will take at least the value 1 (default value is 0) and JOB_QUEUE_INTERVAL which will take a value lower or equal with the value of the refresh time slot (default value is 60 seconds).

In case we can afford a FAST update (only those records updated in the master table) of the snapshot, then on master site we should create a snapshot log (named MLOG$_master_table_name and updated by a trigger AFTER ROW) for the table which will own the snapshot. In case we will need a COMPLETE refresh of the snapshot, then this log file is not necessary.

CREATE SNAPSHOT LOG ON *table*
WITH PRIMARY KEY
TABLESPACE SANP_LOGS
STORAGE (INITIAL 40K NEXT 40K PCTINCREASE 0);

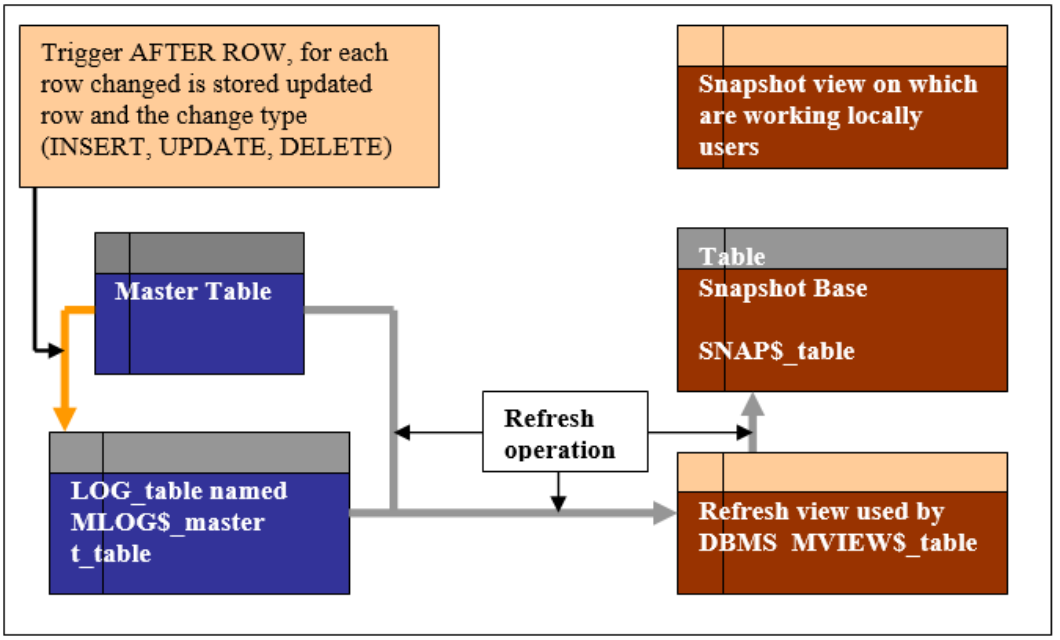

Figure 1. Internal mechanism for read-only snapshots refresh (source [OR05])

By using *Updatable snapshots* we can reflect on master table the updates done in the locally site, because those snapshots allow us to perform DML commands which will be done immadiatelly or later to the master table as well. It is recommended that updated from snapshot to the master table to be done in the same moment with the snapshot refresh with the master table changes.

In comparison with read-only snapshots, the updatable ones, in addition of MLOG$ table and the TLOG$ trigger located on the central site, there is a table locally created as result of snapshot creation, named USLOG$_master_table and a trigger named USTRG$_ master_table as well, responsible with all updates done in the updatable snapshot which needs to be executed on the master table later on.

The *multi-master* replication allow us to maintain the same data set on many sites. Oracle technology allow the automatically synchronizing of all changes done in any site and this type of replication is used for those tables with a dynamic update.

D. Allocation Modelling is the activity which allows us to decide which fragment will be stored on which site based on the set of query initiated and processed at each site. In principle we might have a non-redundant allocation (partitioned) or a redundant allocation (partially or totally replicated).

We can start initially by a non-redundant data allocation and store in each site the fragment with the highest locally usage and the lowest storage cost. Then it might be done a progressivelly allocation of fragments replicas to some sites.

Also, there are some tables/fragments which are used very often read-only which can be fully replicated to all sites and increase the local processing.

E. Distributed query processing **s**hould find a strategy of query execution across distributed system to return expected results in short time in a context of minimizing the communication costs and disk accesses costs.

A distributed query processing starts with the global query addressed to global relations and formed by using the relational calculus operators. In this stage the data distribution is not visible for the user who initiate the query execution.

Using the fragmentation schema and fragments statistics, the initial query is decomposed in sub-queries addressed to local databases. Global query optimization means to find the best operations executions order of sub-queries operations on database fragments, including the communication operations by minimizing a communication cost function. Then, in a second stage, based on local database schema, there is another locally optimization of each sub-query to each local site.

In practice are known 2 main strategies used for sub-query execution when we still need data from many sites: *query site* and *move small*.

In the Query-site strategy, all necesary data fragments to execute a query are transfered to the site where the query was initiated.

By using the *move-small strategy*, the small fragment is moved to the site where the bigest fragment is located, the query is executed there and the results and then send to the site which initiated the query.

6. CONCLUSIONS

To implement the data distribution in a proper manner across a distributed system is a complex activity.

We always need to bear in minds that by doing the fragmentation, the allocation of fragments to sites and the data replication should lead us to a usable distributed system in real-life which should support the company in day-by-day activities and top-management with proper information to be able to take good decisions.

Regarding performance of the distributed system, this should bring us to a solution which minimize the system time-to-info, minimize the communication cost between sites and minimize the storage accessing costs.

Talking about distribution, we might have some tables which would be better to not fragment it at all and even replicate it to all sites. There are so called codes tables which are rarely updated but very ofter read. By keeping to all sites, with insignificant storage costs we can increase the local processing to sites and diminish the communication cost.

Regarding fragmentation, in practice we will use a mixed fragmentation, rather than simple horizontal or vertical fragmentation, based on 80% of volumes of data locally accessed by apps queries.

For replication practical solution, we should understand which fragments are used mainly for read operations. Those can be easier replicated using read-only snapshots. On the other hand, for dynamic fragments of data, in order to increase the locally processing we might use either updatable snapshots or even muti-master snapshots.

Despite the complexity of data distribution, companies could benefit from locally independency of the distributed system in case of glabal system downtime moments. From business perspective, the business continuity is essential.

Regarding costs, a distributed system can come with small servers on locally sites which will be less expensive than a mainframe located in a central node and one or two mirrored machine as backup and disaster recovery solution.

In a nutshell, a distributed system come with a set of advantages in terms of reliability and costs and with an additional complexity which needs to be properly managed in the real life.

7. REFERENCES

[1] [AF08] A.G. Fodor – "Solutii de realizare a sistemelor distribuite", PhD Thesis, ASE, 2008.

[2] [AF16] A.G. Fodor – "Sisteme Informatice Distribuite", ProUniversitaria Publisher, 2016

[3] [CC89] W. Chen, P. Chen, "Centralized and Distributed Data Base Systems", IEEE Computer Society, New York, 1989

[4] [CD99] Charles Dye, "Oracle Distributed Systems", O Reilly&Associates Inc., 1999

[5] [CP85] S. Ceri, G. Pelagatti, "Distributed Databases, Principles and Systems", International Student Edition, McGraw Hill, 1985

[6] [IL11] Ion Lungu, Adela Bara, Constanta Bodea, Iuliana Botha, Vlad Diaconita, Alexandra Florea, Anda Velicanu – "Tratat de baze de date. Organizarea, Proiectarea si Implementarea", ASE Publisher, 2011, ISBN 978-606-505-472-1 / 978-606-505-481-3

[7] [IL15] Ion Lungu (coordonator), Anca Andreescu, Adela Bara, Anda Belciu, Constanta Bodea, Iuliana Botha, Vlad Diaconita, Alexandra Florea, Cornelia Gyorodi – "Tratat de baze de date. Sisteme de gestiune a bazelor de date", ASE Publisher, 2015, ISBN 978-606-505-472-1, 978-606-505-862-0

[8] [OR05] *** Oracle® Database - Administrator's Guide - 10g Release 2 (10.2), B14231-01, Oracle Corporation, June 2005 (https://docs.oracle.com/cd/B19306_01/index.htm)

[9] [OV91] M. Tamer Ozsu, P. Valduriez, Principles of distributed database systems, Prentice-Hall, 1991.

[10] [RV13] Ramesh Dharavath, Vikas Kumar, Chiranjeev Kumar, Amit Kumar - An Apriori-Based Vertical Fragmentation Technique for Heterogeneous Distributed Database Transactions, "Intelligent Computing, Networking, and Informatics", ISBN 978-81-322-1664-3 (http://link.springer.com/chapter/10.1007/978-81-322-1665-0_69)