Journal of Theoretical and Applied Information Technology
15th January 2019. Vol.97. No 1
© 2005 – ongoing JATIT & LLS

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A SYSTEMATIC REVIEW ON DISTRIBUTED DATABASES SYSTEMS AND THEIR TECHNIQUES

[1]KATEMBO KITUTA EZÉCHIEL, [2]SHRI KANT, [3]RUCHI AGARWAL

[1]Ph D. Scholar, Department of Computer Science and Engineering, Sharda University, Greater Noida, India

[2]Professor, Research and Technology Development Centre, Sharda University, Greater Noida, India

[3]Associate Professor, Department of Computer Applications, JIMS Engineering Management Technical Campus, Greater Noida, India

E-mail : [1]kkitutaezechiel@yahoo.com, [2]shri.kant@sharda.ac.in, [3]dr.ruchi@outlook.com

ABSTRACT

Distributed Databases Systems (DDBS) are a set of logically networked computer databases, managed by different sites and appearing to the user as a single database. This paper proposes a systematic review on distributed databases systems based on respectively three distribution strategies: Data fragmentation, Data allocation and Data replication. Some problems encountered when designing and using these strategies have been pointing out. Data fragmentation involves join optimization problem since when a query has to combine more than one fragment stored on different sites. This produces the high time response. Heuristic approaches have been examined to solve this problem as it is known as a NP-Hard problem. Data Allocation is also another particular problem which involves finding the optimal distribution of fragments to Sites. This has already been proved to be a NP-complete Problem. The review of some heuristics methods as solutions has been conducted. Finally, Data replication, with its famous synchronization algorithm, which is the unique strategy to manage exchange of data between databases in DDBS, has been studied. Thus, following problems have retained our attention: serialization of update transactions, reconciliation of updates, update of unavailable replicas in Eager or synchronous replication, sites autonomy and the independence of synchronization algorithm. Therefore, this has been our motivation to propose an effective approach for synchronization of distributed databases over a decentralized Peer-to-Peer (P2P) architecture.

Keywords: *Distributed Database, Data Fragmentation, Data Allocation, Data Replication, Data Synchronization, Peer-to-Peer (P2P) architecture.*

1. INTRODUCTION

Not long ago, various organisations were using Centralized Databases Systems (CDBS) for daily transactions in different domains such as: banking, commerce, booking etc. Even today, there are those that work under this approach. However, we can observe some issues related to the complexity, maintenance, performance, and cost of data communication in a centralized database system during query processing, depending on end-user demand from different sites. So, since a certain time, some of them are motivated to implement efficient Distributed Database Systems (DDBS) or Decentralized Database Systems in their administrative environments for scalability.

The Distributed Database (DDB) System technique derives from the combination of two diametrically opposed approaches to data processing: Databases and their Networking. This approach implicates different factors. The most common are: Data replication, Data fragmentation and Data allocation, etc. [1], [2]. According to [3], [9], a Distributed Database is a set of more than one database interconnected and propagated physically across various locations (sites) which communicate, via a computer network. Moreover [10], offer a practical and illustrative definition so that: “A Distributed Database is a collection of multiple, logically interrelated databases distributed over a Computer Network.” He adds that “sometimes Distributed Database System is used to refer jointly to the Distributed Database and the Distributed Database Management System”. In this approach, as shown in Figure 1, in a computer network, data, process and interface components of an information system are also distributed in multiple locations in the network [9].

Designing a Distributed Database, it is necessary that it be entirely resident on various sites in a computer network or its portion. In this logic, there must be at least two sites hosting the database, and not necessarily each one, in the Computer Network. The major objective of a Distributed Database System is to appear as a centralized system to end-users or user terminals or terminal emulators [9]. The administration of Distributed Database activities is conducted by the Distributed Database Management Systems (DDBMS). This last is a software that manages the Distributed Database so that each site maintains its database locally and the more it provides access mechanisms to users so to connect to the system so that their data is distributed and replicated on several sites as shown in Figure 1; disparate from the Centralized Database System (CDBS), where only one copy of the Database is stored as shown in Figure 2 [5], [10].

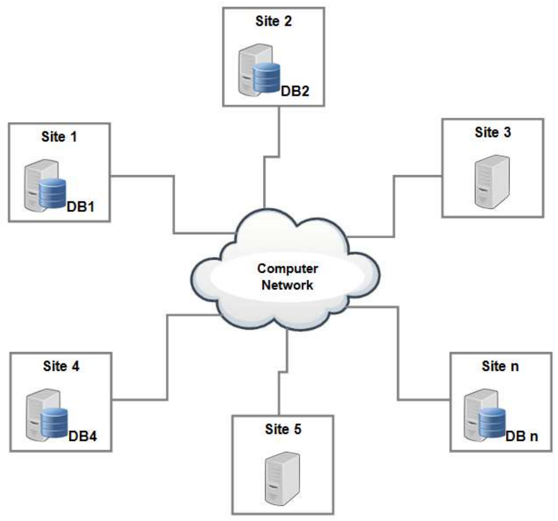


Figure 1. Architecture of a Distributed Database System.

If the architecture of a Distributed Database System was such that the database resides on a single node of the computer network (Figure 2), then despite the existence of this network the distribution problems of database management would be identical to the problems encountered in the architecture of a Centralized Database System. Because in this last logic, the Database is managed by one computer system on a central site (Site 3 in Figure 2) and all transactions or queries are oriented to that site.

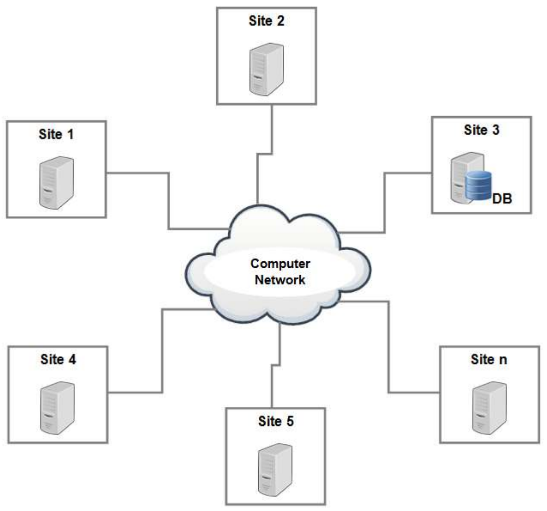


Figure 2. Architecture of Centralized Database System.

So in this way, as additional consideration, it is clear that the system has to work with a big transmission delays due to the unique node which is supposed to receive all requests. So, it is evident that it is not necessarily the existence of a computer network that suffices to justify the establishment of a Distributed Database System. That is why our interest is focused on an environment where data is distributed on multiple sites (Figure 1) [10].

1.1 Types of Distributed Databases

In general, there are two categories of Distributed Database Systems [1], [9], [24], [26], [32]:

- ✓ Homogeneous Distributed Database Systems: In these Systems, the data is dispersed over servers on which it's running the same schema of the physical Database (same schema of its portions) and same Database Management System (DBMS) software, same Operating System and Hardware.
- ✓ Heterogeneous Distributed Database Systems: In these Systems, dissimilar sites may work under the control of different schemas of the physical Databases, diverse DBMSs and so different Operating Systems and Hardware. These databases systems are always interconnected to allow access to data stored on multiple sites.

1.2 Advantages and disadvantages

Here are some advantages of Distributed Databases Systems [9], [10], [27]:

- ✓ Robustness in the functioning of the whole system: The problems experienced in one branch of the organization do not influence the other branches in the same way;

- ✓ The temporary interruption of the network that connects the branches of the company does not interrupt the operation of the local database;
- ✓ Data security: Staff access may be limited to only the part of the data that it needs to access;
- ✓ Non-overloading the network with reduced traffic also reduces the cost of bandwidth;
- ✓ Good and high performance: queries and updates are handled locally so that there is no more bottleneck on the network;
- ✓ Local errors are kept locally and do not affect the entire organization.

Here are the few drawbacks of distributed database systems [9], [10], [27]:

- ✓ Complex implementation: A distributed database presents a certain complexity in its installation as well as its maintenance; less complicated task in a centralized architecture;
- ✓ Data security: the multiplicity of access points or remote sites with access to data does not guarantee their security;
- ✓ Data integrity: given that access to data is no longer restricted to the unique users of a site, their corruption is likely;
- ✓ In order to be able to achieve data efficiency in distributed database systems, one has to carefully place the data;
- ✓ Again, to make the distributed database system efficient, we have to reduce significantly the interaction between the sites.

The purpose of this paper is to review previous work, focusing on the published literature between 2008 to 2017, by highlighting their strengths and weaknesses with the aim of achieving an effective solution to maintain consistency in homogeneous Distributed Databases architecture in full migration toward Peer-to-Peer (P2P) environment. However, the structure of the remainder of this paper is organised as follows: apart from Section 1 which is devoted to the Introduction, Section 2 review the general literature about the Distributed Database design techniques or strategies and issues encountered in the design and the usage of Distributed Databases and possible solution which already exist, while Section 3 propose the new model of synchronization through the replication process, and finally Section 4 concludes the study.

2. LITERATURE REVIEW

This paper is not the first to direct its thoughts on the problem of DDBS and their techniques. It is therefore essential for to review the literature that fits in this area in order to justify our research. This literature review is organized as follows: the first unit ‘A’ discusses the Distributed Database design techniques or strategies used, while the second unit ‘B’ discusses the issues encountered in the design and the utilization of Distributed Databases.

2.1 Distributed Databases Design Techniques

2.1.1 Design strategies

The term Distributed Database design has a very broad and un-precise meaning. The design of distributed computer systems involves deciding on the assignment of data and programs to the computer network sites, as well as the design of the network itself [1]. The aspects of the design of a Distributed Database cover, for the first time, those of the Centralized Database such us [10], [22]:

- ✓ The modelling of the conceptual diagram to describe the integrated database: the data used by the applications manipulated by the users;
- ✓ The design of the physical schema: to illustrate the conceptual schema by determining the data storage areas and the appropriate access methods.

In a Distributed Database these two problems become the design of the global schema and the design of the local physical databases at each site. The techniques which can be applied to these aspects are the same as in Centralised Databases. The distribution of the database adds to the above aspects two new ones [1], [10]:

- ✓ The fragmentation Design: this is how to define the partitions of the global relations according to horizontal, vertical or mixed partitions;
- ✓ The design of fragment allocation: the representation of diagrams on physical images that define how fragments will be allocated to sites and then replicated.

There are two major ingenuities in designing Distributed Databases: the top-down method and the bottom-up method [1], [10], [11]. These methods convey very different techniques in the design process. The top-down method is much more suitable when designing homogeneous strongly cohesive Distributed Databases, while the bottom-up method is more suitable for heterogeneous or multi-databases.

2.1.1.1 Top-down method

Mostly used when the Distributed database is implemented from beginning; as illustrated in the Figure 3. The design process starts from the analysis of requirements. This phase includes the company situation analysis, the problems definition and constraints, the objectives definition, and the scope design and boundaries [1], [12].

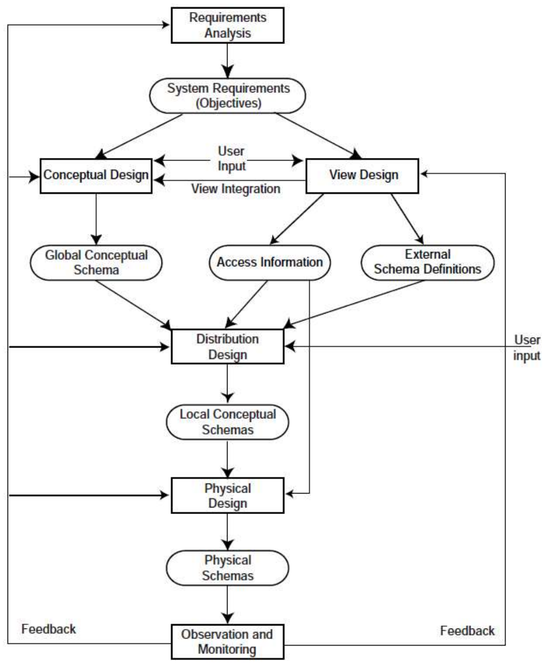


Figure 3.Top-Down Design Method [1]

The next two tasks to be undertaken are conceptual modelling and view design. Conceptual modelling deals with the formalization and standardization of entity relationships. It also focuses on data requirements while view design activity defines the end user interfaces. As the conceptual modelling process determines the types of entities and the relationships among them, then the entity analysis goes further in determining the entities and their attributes. The functional analysis determines the fundamental functions involved in the organisation modelling. View integration is an activity that defines the conceptual model that must support existing applications as well as future applications [1].

2.1.1.2 Bottom-up method

This method is used when Distributed Database already exists and other features or another Database have to be added in existing environment [11]. In this way, the problem is the integration of several existing local schemas into a global conceptual schema considered as already developing a distributed system.

When combining many existing databases to develop a distributed system, the bottom-up method is used because it is based on the integration of several existing schemas into a single global schema. It is during, the combination of more than one existing heterogeneous system to build a distributed database system using the ascending method is also possible. Thus, the Bottom-up design process requires the following steps [1], [10], [12]:

- ✓ The selection of a mutual prototype to describe the global schema of the database;
- ✓ The conversion of all local schemas into a mutual data model;
- ✓ The unification of local patterns to arrive at a mutual global schema.

2.1.2 Distribution strategies

The design of the Distributed Database System includes that of the global conceptual schema, which is added to local schemas, based on the three-level architecture of the DBMS in all sites. The establishment of a computer network across sites of a distributed system is an additional complex problem of design. The vital design problem concerns the distribution of data between the sites of the distributed system; therefore, the modelling and implementation of the Distributed Database System is a daunting task of which we can cite the following significant factors [10], [20]:

2.1.2.1 Data fragmentation

The term decentralized or distributed database is referred to distribution of the data over the computer network architecture where the copy of the entire database is distributed or stored at different sites. Likewise, it can also be a single database, split into pieces and scattered across different sites. Thus each site stores the data or relationships it often needs and gets the rest if needed from other sites [9], [10], [17]. These relations may be divided into portions of the original relation: this is what is called Data fragmentation or partitioning. Fragmentation is the process of breaking down a relation into small relations which are called fragments usually stored at different sites [1], [2], [10]. A relation can be fragmented in diverse form: horizontally, vertically, or hybrid [2], [6], [9], [17], [18], [19], [20], [21], [22]:

- ✓ Horizontal fragmentation: It divides a relation R into fragments based on its tuples or records, as depicted in Figure 4. All fragments are tuples subsets of the original relation. To constitute a fragment, the selection of certain specific record lines is done according to criteria [2], [17], [18], [19]. According to [9], every fragment comprises

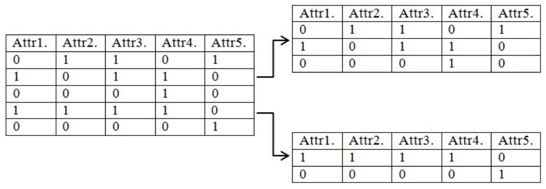
a subset of rows of the original relation. The idea behind horizontal fragmentation is that each site must store the data it currently uses so that when querying the database by a select query, the transaction executes as fast as possible [20]. Horizontal fragmentation divides the "Rows" of R where
R = R1 **U** R2 **U**…**U** RN.

R(Attr1, Attr2, Attr3, Attr4, Attr5)

➔ R1(Attr1, Attr2, Attr3, Attr4, Attr5),

➔ R2(Attr1, Attr2, Attr3, Attr4, Attr5),

➔ RN(Attr1, Attr2, Attr3, Attr4, Attr5).



Attr1.	Attr2.	Attr3.	Attr4.	Attr5.
0	1	1	0	1
1	0	1	1	0
0	0	0	1	0
1	1	1	1	0
0	0	0	0	1

Attr1.	Attr2.	Attr3.	Attr4.	Attr5.
0	1	1	0	1
1	0	1	1	0
0	0	0	1	0

Attr1.	Attr2.	Attr3.	Attr4.	Attr5.
1	1	1	1	0
0	0	0	0	1

Figure 4. Horizontal fragmentation

Horizontal fragmentation is further comprises of two types [10], [17], [22]:

Primary horizontal fragmentation: This is realized when tables in a database are not linked or have not any dependencies. So, the relationship doesn't exist among them.

Derived Horizontal Fragmentation: This type of fragmentation is suitable for relations in parent-child relationships. It ensures that fragments of tables that are linked together by means of primary keys on the parent side and foreign key on the child side are stored on the same site.

✓ Vertical fragmentation: It splits a single relation R into subsets of relations those are projections of relation R in the respect of attributes subset, as shown in Figure 5. These relations are grouped with attributes and usually accessed by queries. When these fragments joined the original relation is rebuilt [2], [17], [18]. According to [9], every portion comprises a subset of columns of the original relation. Vertical fragmentation thus consists in subdividing a relation into sub-relations by the projection principle of an original relation into a subset of attributes, apart from the primary key (s), so constitute fragments each of which must include the attribute (s) of the primary key (s) of the original relation [19], [20]. This harmonization only makes sense if various sites are predisposed to use the same entity or relation for different functions [20].

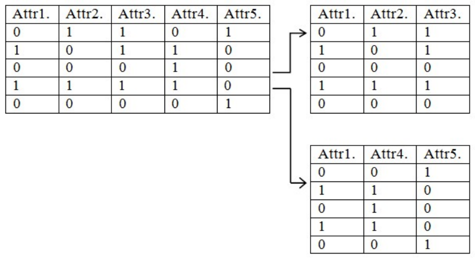
Vertical fragmentation splits "Columns" of R where
R = R1 $\bowtie$ R2 $\bowtie$ … $\bowtie$ RN. Attr1 is the primary key.

R(Attr1, Attr2, Attr3, Attr4, Attr5)

➔ R1(Attr1, Attr2, Attr3),

➔ R2(Attr1, Attr4, Attr5),

➔ RN(Attr1,…, AttrN).



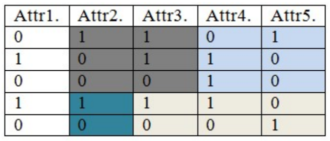
Attr1.	Attr2.	Attr3.	Attr4.	Attr5.
0	1	1	0	1
1	0	1	1	0
0	0	0	1	0
1	1	1	1	0
0	0	0	0	1

Attr1.	Attr2.	Attr3.
0	1	1
1	0	1
0	0	0
1	1	1
0	0	0

Attr1.	Attr4.	Attr5.
0	0	1
1	1	0
0	1	0
1	1	0
0	0	1

Figure 5. Vertical fragmentation

✓ Hybrid fragmentation (mixed fragmentation): it combines the two previous ones. In this type of fragmentation, the relations is broken down into random fragments, according to the expectations of the design, as represented in Figure 6 here below. Each fragment can be assigned to a defined site [20], [21]. This type is the most complex because the two preceding types (horizontal and vertical) are used to build a schema that can meet the requirements of the DB application. The original relation is obtained by joining or union operations [17], [20]. Mixed fragmentation thus consists of a veritable fragmentation followed by horizontal fragmentation, or the opposite [20], [21]. In this fragmentation we combine the selection and projection operators of relational algebra to achieve this:
Π_p(_Attr1,...,AttrN(R)) or Π _Attr1,...,AttrN(_p(R)).



Attr1.	Attr2.	Attr3.	Attr4.	Attr5.
0	1	1	0	1
1	0	1	1	0
0	0	0	1	0
1	1	1	1	0
0	0	0	0	1

Figure 6. Hybrid fragmentation

Data fragmentation major advantages in Distributed Database are [1], [9], [10], [23]:

✓ Effectiveness and Optimization: the system becomes efficient because the response time of a transaction is reduced significantly when the data

are stored closely and isolated from those used by other users and other applications;
- ✓ Easy roll-in and rollout of data.

Following are the disadvantages of data fragmentation [1], [10], [23]:

- ✓ Low performance: applications performance which use queries combining the data from the fragments stored on different sites is low because the temporal complexity of a selection or union retrieval that involves various relations of various sites is high;
- ✓ Integrity problem: relation functional dependency including the data they carry must also be fragmented to be allocated to different sites across the computer network to preserve the referential integrity.

2.1.2.2 Data replication

The replication concept connotation is understood in the sense that the storage of the same data or the same relations is done on more than one site. These technologies assure to copy and to distribute data or database objects from one database to another in order to run synchronization between them for maintaining the consistency [13], as shown in Figure 7. It stores separate copies of the database or their portions at two or more sites. It is a popular fault tolerance technique of distributed databases [14]. In a replication model the major problem turns out to be the maintenance of consistency between the data [23]. In this way, setting up a replication procedure involves deciding which fragments or tables of the database will be replicated [31].

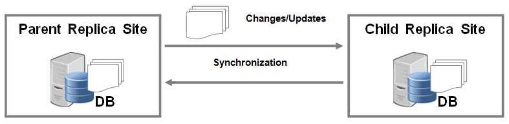


Figure 7. Data replication protocol

From this point of view, we retain two notions which we will first of all clear up: synchronization and consistency.

Data synchronization is a process that establishes consistency between data from a Source or Master or Parent to a Destiny or Slave or Child and vice versa. This technology is parameterized and runs in time according to any harmonization that can make it trigger automatically or run manually, as appropriate to copy the data changes in one or two directions [8], [49], [50], [51].

These Data Synchronization directions are as follows [40], [49]:

- ✓ One-way or Unidirectional or Asymmetrical Data Synchronization [40], [41]: This process transmits the changed data to the source or main node Database toward the target node's Database. Sometimes this form of synchronization is practiced to unload a Database to another to make an analysis, to make the Database backup in order to prevent breakdowns or for maintenance reasons of Database under usage.
- ✓ Two-ways or Bidirectional or Symmetrical Data Synchronization [7], [40]: This is used when it’s mandatory to keep the same copy of the data in each data storage node on the network. The data is replicated on all the nodes and the exchange is done dynamically between them. This configuration is used to duplicate the same data across multiple nodes to reduce server load and maintain data access very quickly.

In turn, the Data consistency is the capacity for a Replication model to reflect on the copy of a data or replicas the changes or updates made on other copies of this data. The consistency of Replication models is essential to abstract away execution particulars, and to classify the functionality of a given system [15]. At the same idea angle [23] emphasizes that mutual reliability rules need the identity of all fragments or whole relations. Therefore, the DBMS must ensure that the Database update transaction is performed on all sites where copies of data exist, to maintain the reliability of the data between the replicas.

From this emerge two methods of replication which are as follows [1], [9], [16]:

- ✓ Synchronous Replication: Each copy of modified relations (fragments) must be updated before the commitment of the transaction. It must be remembered that the end user has received the most freshly actuated data. Many tools integrating the synchronous replication algorithm allow data to be written to the Master Database and to the Slave (s) or Replica (s) simultaneously, in “real time”. This ensures that all copies remain in sync.
- ✓ Asynchronous Replication: Asynchronous replication tolerates momently the difference between values of same relation or fragment copies. Data is updated after a predefined interval of time. Tools that integrate asynchronous replication algorithms allow data to first be written to the Master database before being copied to the Slave (s) or Replica (s). Here,

therefore, the replication takes place in near-real time, i.e. that it takes place based any program or planning. For example, it can be planned that the update transactions will be transferred to the Slave (s) in batches on a periodic basis of fifteen minutes.

2.1.2.3 Data allocation

Allocation is the DB distribution approach in which DDB fragments are assigned on a distributed network sites [23]. The allocated data can be replicated (double copies) or not replicated (single copy). Fragments replication improves effectiveness and consistency of read-only queries but increase update cost [20]. Four alternatives strategies have been identified for data allocation [1], [10], [22], [23], [26], [46]:

- ✓ Centralized: in this strategy, the distributed system is just a single database and DBMS stored at one site with users distributed across the communication network. Remote users can access centralized data over the network; thus, this strategy is similar to distributed processing.
- ✓ Fragmented (or partitioned): this technique divides the entire database into disjoint fragments, where each fragment is assigned to one site. In this strategy, fragments are not replicated.
- ✓ Complete (or full) replication: each site of the system maintains a complete copy of the entire database. Since all the data are available at all sites, locality of reference, availability and reliability, and performance are maximized in this approach.
- ✓ Selective replication: this is the combination of centralized, fragmented, and complete replication strategies. In this approach, some of the data items are fragmented and allocated to the sites where they are used frequently, to achieve high localization of reference. Some of the data items or fragments of the data items that are used by many sites simultaneously but not frequently updated are replicated and stored at all these different sites. The data items that are not used frequently are centralized.

Here below, in the Table 1, it will be described these different data allocation strategies and also drew a comparison between them.

Table 1. Comparison of strategies for Data allocation [10], [46]

	Locality of reference	Reliability and availability	Workload distribution and performance	Storage costs	Communication costs
Centralized	Lowest	Lowest	Poor	Lowest	Highest
Fragmented	High	Low for data item, high for system	Satisfactory	Lowest	Low
Complete replication	Highest	Highest	Best for reading	Highest	High for updating low for reading
Selective replication	High	Low for data item, high for system	Satisfactory	Average	Low

2.2 Distributed Databases Design Issues

In the previous section, we highlighted that relations in the schema of a Database are typically divided into smaller fragments. The primary distributed database design goal is to break the relation, to allocate and to replicate the fragment in different sites of the distributed system with local optimization on each site [2]. Thus, the aim of this section is to go into detail about the problems encountered during fragmentation, allocation of fragments and replication between them.

2.2.1 Data fragmentation problem

The fundamental problem highlighted for the fragmentation techniques commonly used and presented in this work here above is minimizing distributed joins or join query optimization [2]. One database interaction issue is the join query execution across sites [37]. First and foremost, let's talk about the query processing.

2.2.1.1 Query processing

A Query is a text that gives to a DBMS an order to run on a Database. The Query consists of keywords or declarative query language commands. The Structured Query Language (SQL) is mostly used to facilitate the interaction with the database. Generally, there are two types of queries [47], [55]:

- ✓ Data retrieval query which consists of selecting data from the database. It is materialized by Select SQL command;
- ✓ Data action or update query which consists of modifying data. It is materialized by Insert, Delete and Update SQL commands.

Query processing is further categorized in to distributed query processing of multidatabase query processing:

1. Distributed query processing

As far as the "database fragmentation" approach is concerned, when we talk about the query processing, we refer to the data retrieval query, according to this section problem, already mentioned above. The query processing is an important issue in centralized and decentralized or distributed databases architecture. However, the solution to this problem is more difficult to reach in decentralized environment than centralized because of sites dispersion and relations fragmented/replicated. Theses parameters affect considerably the distributed query performance because the join of relations/fragments from different sites increase communication and processing cost [1], [57]. Once submitted, there are layers which are involved in distributed query processing. These layers are the succession of 4 steps, depicted in Figure.8, followed by the processing such that [25], [32], [33]:

- ✓ Query decomposition: The query decomposition is the first phase of the query processing which consists of breaking down the request into a series of operations of the relational algebra. DBMSs typically exploit seven set operators that are selection, join, projection, Cartesian product, union, intersection, and difference.
- ✓ Data localization: The inputs of the second layer, data localization layer, is an algebraic query on the global conceptual schema. As generally, the relations are fragmented and separated into disjoint subsets, called fragments, stored on different sites, this layer determines the fragments that are involved in the query and transforms the distributed query into a query about the fragments.
- ✓ Query optimization: The inputs of the query optimization, which is the third layer, is an algebraic query on fragments whose purpose is to find the near the optimum strategy for executing the query. Finding the optimal solution is intractable by computation. An execution strategy for a distributed query can be described with relational algebra operators and communication primitives for data transfer between sites. In short, query optimization consists of finding the "best" ranking of the operators in the query, including the communication operators that minimize a cost function.
- ✓ Query execution: The last layer is the query execution to be performed by all sites with fragments involved in the query. Each local query, i.e. running on a local site, is optimized using the local schema of the site and executed. However, the algorithms for performing the relational operators can be chosen. Local optimization uses the algorithms of centralized systems.

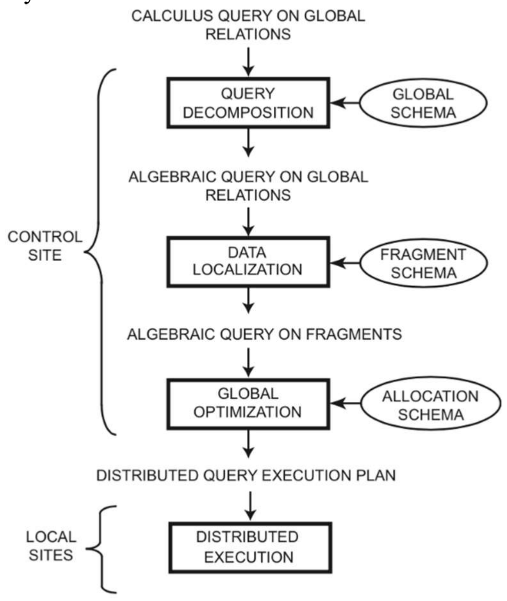


Figure 8. Query processing in Distributed Databases [1]

The query optimization, phase 3, makes it possible to improve the performance of a database query. In this way, query optimization defines the best execution by the DBMS of a given code. The database query performance is achieved when the processing of an action assigned to is done effectively and / or efficiently. In this approach it is necessary to know that there must be certain factors on the basis of which the level of performance must be calculated; e.g. the time of execution or of answer, the space of memory, etc. [25].

2. Multidatabase query processing

Multidatabase is one of heterogeneous DDSs aspects that advocates the empowerment of DBMSs in a distributed environment. This aspect is not supported by homogeneous DDBSs. However, multidatabase query processing is more complex than processing in homogeneous DDBSs. The reasons which characterize this complexity can be pointing out [1]:

- ✓ DBMSs don't have same computing capabilities, performance and behaviour to process complex queries;
- ✓ DBMSs don't have same query processing cost so that the local query optimisation depend on each local DBMS;
- ✓ DBMSs don't have same query language as well as data models so that queries translation is a challenge.

Mostly, multidatabase query processing is performed by mediator/wrapper architecture which allow the cooperation between different DBMSs. Thus, like in homogenous distributed three main layers or steps of query processing which are Query decomposition & Data localization, Query optimization & execution and Query translation & execution are involved in the multidatabase query processing. The *Mediator* performs the two first steps, Query decomposition & Data localization by rewriting queries using views and Query optimization & (some) execution by taking in consideration local fragments/relations, whereas the *Wrapper* performs the last layer, the Query translation & execution by returning, to the Mediator, the results provided by the execution on each DBMS of translated queries according to its language syntax [1], [55], [57].

2.2.1.2 Status of the problem

- ✓ Problem definition: This problem consists of finding an optimal approach to minimize the data transmission cost in DDBS, even with the one join attribute. Determining the optimal sequence of join procedures in query optimization leads to exponential complexity. In DDBS, principles of query optimization can be the cost of the query or the response time of the query. The cost of the query has primarily two things to be considered: local processing cost and communication cost. The communication cost minimization is the crucial problem to solve. The cost of the data communication between two sites is a function of the linear form B + A.X where "B" is the starting cost of the transmission, "A" is the constant cost related to the transfer of a unit of data and "X" is the volume of data transmitted from one site to another [38]. The most proposed efficient solution strategies which reduce the transmission cost are based on semi-join operation, assumed that the transmission cost is the dominant factor in distributed databases. Thus [38] proposed a modelling to this problem as follow: Given a Database D of j tables D = $\{T_1, T_2, \ldots T_j\}$, distributed over n sites $\{S_1, S_2, \ldots S_n\}$. For optimizing the processing of a query Q, query is of form T_{i1} join(Key_1) → T_{i2} join(key_2) → T_{ih} join(key_h).
- ✓ Methods: There are diverse techniques to optimize the databases queries. These techniques improve the performance of the query and decrease the cost. In this way [37] and [39] presented two of them as follow:
  - Data replicated: The first technique for the join request to transfer data from the servers to the client and insert it into the client Database so that the results are executed by join query, on the client site and take the data directly on its own Database. The basic strategy of data replicated is to send the smaller table to the site that contains the larger table, and perform the join on that site.
  - Data non-replicated: Second technique performs the join query on the client site without inserting the data into its database. Parallel processing doesn't focus on minimizing the data transmission quantity but rather maximizing the simultaneous transmissions number.

2.2.1.3 Some heuristics

The distributed joins or Join query optimization problem is a NP-Hard problem. The worst case is when the query is submitting over the global schema i.e. all sites so that data fetch from those sources through packaging [25], [27]. To deal with such a problem, there is need of heuristic approaches to solve the problem in polynomial time [28], [38].

T. Robert [58] has designed a cost model that identifies opportunities for inter-operator parallelism in query execution plans. This makes it possible to estimate more precisely the response time of a query. It has merged two existing centralized optimization algorithms DPccp (Dynamic Programming connected subset complement pair Algorithm for Query Optimization) and IDP1(Iterative Dynamic Programming Algorithm for Query Optimization) to create a much more efficient IDP1ccp algorithm. He proposed the multi-level optimization algorithm framework that combines heuristics with existing centralized optimization algorithms. The proposed Distributed Multilevel Optimization Algorithm (DistML) uses the idea of distributing the optimization phase across multiple optimization sites to make full use of available system resources.

W. Di [59], proposed the Cluster-and-Conquer algorithm for query optimization for federated database which take into account the execution conditions. He first considered the entire federation as a clustered system, grouping the data sources according to the network infrastructure or the boundaries of the enterprise; then he provided each group of data sources with their own cluster mediator. This algorithm divides the optimization of the query into two procedures: firstly, the global mediator decides on inter-cluster operations, and secondly, cluster mediators treat cluster subqueries with consideration of execution conditions.

In this section, queries were considered as positive relational algebra involving the conjunction of selection, join, projection, Cartesian product, union, intersection, and difference. The query optimization problem faced by everyday query optimizers is becoming more complex with the increasing complexity of user queries. The NP-Hard join order problem is a central problem that an optimizer must face to produce optimal plans. This has always motivated researchers in this field to find effective solutions.

However, the problem of DDBSs does not only meant finding an effective solution of join query processing. Nevertheless, as soon as a database is partitioned, it is necessary to proceed by allocating the fragments to the respective sites. Thus, the next section will deal with the issue of fragment allocation.

2.2.2 Data allocation problem

The resources allocation across computer network nodes is an ancient distribution topic that has been studied widely [1]. Let's put a set of fragments F = $\{F_1, F_2, \dots, F_n\}$ and a distributed environment containing sites S = $\{S_1, S_2, \dots, S_m\}$ on which a set of query Q = $\{q_1, q_2, \dots, q_q\}$ is running. The allocation problem involves finding the "optimal" distribution of F to S.

2.2.2.1 Status of the problem

✓ Problem: The problem of allocation implicates to find the "optimal" distribution of F to S. Problem of fragments allocation is one of the significant issues that need to be discussed in the optimality definition. The optimality can be defined with respect to two measures [1], [10]:

- *Minimal cost*. This is a function that includes the storage cost of each F_i on a site S_j, the query cost of F_i on the site S_j, the cost of updating F_i on all the sites where it is stored and the cost of data transmission. The allocation problem is to find an optimal solution by minimizing the combined cost function.
- *Performance*. The allocation strategy is put in place to maintain performance measure. Two well-known methods consist of minimizing response time and maximizing system throughput at each site.

✓ Generic model: To optimize system throughput or minimize response time at each site, several models have been designed. But, find one that improves to achieve "optimality" that takes into account performance and cost factors, in other words a model that responds to requests from users in a minimum time and also the cost of minimal processing. This remains a very complex problem. However, [1] proposed very simple modelling of the problem that is general: Let F and S, considering a single fragment, F_k. Let us set a number of hypotheses and definitions that can formalize the allocation problem.

1. Assume that Q can be modified so that it is possible to identify the update and the select queries, and define the following for a single fragment F_k:
   $T = \{t_1, t_2, \dots, t_m\}$ where t_i is the traffic of the select transaction generated at site S_i for F_k, and $U = \{u_1, u_2, \dots, u_m\}$ where u_i is the traffic the update transaction generated at site S_i for F_k.
2. Assume that the transmission cost between two sites S_i and S_j is set for a unit. Moreover, assume that there is the difference between the Update and the Select transaction so that: $C(T) = \{c_{12}, c_{13}, \dots, c_{1m}, \dots, c_{m-1,m}\}$ and $C'(U) = \{c'_{12}, c'_{13}, \dots, c'_{1m}, \dots, c'_{m-1,m}\}$ where c_{ij} is the unit transmission cost for *Select* queries between sites S_i and S_j, and c'_{ij} is the unit transmission cost for *Update* queries between sites S_i and S_j.
3. Assume that the fragment storing cost at site S_i is d_i. So we can state that $D = \{d_1, d_2, \dots, d_m\}$ to store the fragment F_k at each site.
4. Assume that for sites, no storage constraints or transmission constraint. Thus, this same problem can be formulated as that of cost minimization where it is necessary to find the set $I \underline{C} S$ which specifies the place of storage of the replicas of the fragment. Subsequently, x_j presents the decision variable for storage as

$$x_j = \begin{cases} 1 \text{ if fragment } F_k \text{ is assigned to site } S_j \\ \quad\quad 0 \text{ otherwise} \end{cases}$$

The accurate description is the following:

$$\min \left[\sum_{i=1}^{m} \left(\sum_{j|S_{j\in I}} x_j u_j c'_{ij} + t_j \min_{j|S_{j\in I}} c_{ij} \right) + \sum_{j|S_{j\in I}} x_j d_j \right]$$

Focus to

$$x_j = 0 \; or \; 1$$

The second term of the objective function above illustrates the total cost of storing all replicas of the fragments, whereas the first refers to the cost of passing updates to all sites that own the fragment replicas and to the cost of execution of selection transactions on the site. Finally, it can result in a minimum cost of data transmission [1].

But this model has been designed without taking into account information requirements or measurable data on the database. That's why it remains generic. According to [1], this model is NP-complete and several different modelling of the same problem proved to be so difficult for a long time. The complexity of the problem lies in the fact that there may be fragments and sites in large numbers. So finding optimal solutions is not surely computational. Thus, several researches have already been done to try to have good heuristics that can offer some optimal solutions

2.2.2.2 Some heuristics

Most of research have already proved that the solution of the Database Allocation Problem (DAP) formulation is NP-complete. Thus one has to examine some heuristic approaches that yield optimal solutions and taking into account requirements of information indicated here above.

U. Tosun, T. Dokeroglu and A. Cosar [34], developed a series of heuristic algorithms and adapted them to each other through experiments and defined the most efficient way to solve the DAP in distributed databases. In their experiments, the execution times and the quality of fragment allocation alternatives were studied. They managed to produce reliable and more or less satisfactory results even for a considerable number of fragments and sites. Their model is up to determine the sites where each fragment will be allocated and thus a single fragment for each site. The fragments replication on several sites and the assignment of several fragments to any site have not been considered in this work.

A. Amer and H. Abdalla [35], have implemented a heuristic for the replicated and unreplicated dynamic reassignment model that has developed an optimal solution for reassigning fragments in a Distributed Database System. This method stipulates that the allocation of fragments on the sites is executed generally based on the frequency of the requests that one executes on this site. Starting from this frequency, the model proposes a plan to reassign fragments based on transmission costs between sites and updates the cost values for each fragment. The reallocation operation is performed taking into account the maximum update cost values for each fragment and consequently the reassignment decision. Finally, the results proved that this method contributes effectively to the resolution of problems of dynamic reallocation of fragments.

Referring only to these two authors, above, let us end these lines, reminding nevertheless that several models that convey heuristic methods have already been developed to solve these problems encountered when designing the distribution of data. The scope of solutions for the DAP is based on the replicated and non-replicated static or dynamic fragments allocation.

These two approaches, diametrically opposed, focus successively on [34], [36]:

- ✓ Static algorithms: they use predefined information requirements;
- ✓ Dynamic algorithms: they take modifications of information requirements into consideration.

Since the database still only fragmented, this problem is seen from one side whereas when fragments have to share data among them through replication procedure, then it takes another look. In this way, here below, we have to review the replication problem separately because sometimes one can design distribution models based on existing fragmentation and fragments allocation patterns.

2.2.3 Data replication problem

The Data replication is another issue to consider during the design since when we advocate designing a Distributed System in which fragments or whole relations have to exchange data among them. This exchange of data is performed by the mean of synchronization procedures which are sets of transactions execution. Concretely this problem consists to keep reliability and availability among replicas [1], [57].

2.2.3.1 Transaction management

In Distributed Database Management System (DDBMS) a transaction is a sequence of Read or Write operations that takes the database from one reliable being state to another reliable state and ending with one of the following two statements [24], [31]:

- ✓ Commit: indicating the validation of all operations performed by the transaction;
- ✓ Rollback or Abort: indicating the cancellation of all operations made by the transaction.

In this way, all operations of the transaction must be validated or cancelled jointly. Essentially, DDBMSs own a query language to interact with databases, of which Structured Query Language/Programming Language (PL/SQL) is mostly used [24], [47]. As indicated beforehand, a transaction can perform two types of operations: read operation known as a query transaction, materialized by Select operator and write operation known as an update transaction, materialized by Insert, Update and Delete operators in SQL [42], [47], [55].

a) Transaction properties

DDBMSs ensure that transaction execution meets a set of good properties that lead to the consistency and reliability of a distributed database and conveniently summarized by Atomicity, Coherence, Isolation, Durability (ACID) as follow [24], [29], [31]:

- ✓ Atomicity is guaranteed by the rollback command to cancel any changes made to the database during the transaction. It also releases any locks placed on the data during the transaction by the system. In this logic, a transaction is either performed completely (so until the commit that concludes) or cancelled completely (rollback or abort in the system, in case of a possible failure or deadlock or cancelling by the user himself);
- ✓ Consistency: all commit and rollback must be set to run when the database is in a consistent state. It must always be remembered that a commit or a rollback marks the end of a transaction, and therefore defines the set of operations that must execute jointly (or "atomically");
- ✓ Isolation is the property that ensures that the execution of a transaction is completely independent of other transactions. Consequently, no other transaction can read or modify data that is being modified by another transaction;
- ✓ Durability is guaranteed by the commit command to make all updates made on the database permanent during the transaction. The system ensures that any system interruptions occurring after the commit will not affect these updates.

b) Concurrency control

In DDBMSs the simultaneous execution of multiple applications can cause concurrent access problems such that the same information being handled by multiple users at the same time. The concurrency unit is the transaction that also plays a role with respect to the control of data integrity [27]. A database must be consistent before and after a transaction. So, the problem is that consistency can be violated during the concurrent execution of transactions. In this way the concurrency control is the set of methods implemented by a database server to ensure the good behaviour of transactions, including their isolation [23].

Concurrent execution without synchronization constraints can produce a number of problems who's the most important are loss of operations and improper readings. Thus, it is necessary to fix a property determining a correct execution of transactions completion: the serializability [28]. The serializability is NP-Complete problem [10]. Transactions concurrent execution is correct (produces the same result) if it is equivalent to a serial running. Serialization is a strong property that limits parallelism to execution and improve performance [31]. We can distinguish two main techniques to ensure serialization [1], [10], [22], [29]:

1. Pessimistic or a priori approach makes sure that we cannot have an incorrect execution. We have found two algorithms: Two-Phase Locking (2PL) and Timestamp Ordering (TO), already implemented in most of commercial DDBMSs and also a hybrid technique.

   a) Two-Phase Locking (2PL) technique

   In an industrial way, the only solution implemented is the locking approach. The 2PL algorithm is the oldest, and still the most used, concurrency control method ensuring strict serializability. Unfortunately, it is reputed to induce deadlocks as well as rejections of transactions [23], [30].

   The 2PL is based on the locking of current read or update tuples. The idea is simple: each transaction wishing to read or write a tuple must first obtain a lock on this tuple. Once obtained, the lock remains held by the transaction that placed it, until this transaction decides to release the lock. A lock is a state variable associated with an object of the database and indicating its state with respect to read / write operations [29], [31].

The 2PL transactions execution concretely means that each transaction has a growth phase, where it obtains locks and accesses the data elements, and a shrink phase, during which it releases locks. The lock point is when the transaction has reached all locks but has not yet begun to release one of them. Thus, the lock point determines the end of the growth phase and the beginning of the shrinkage phase of a transaction [1]. The Figure 9 here below depicts 2PL execution protocol.

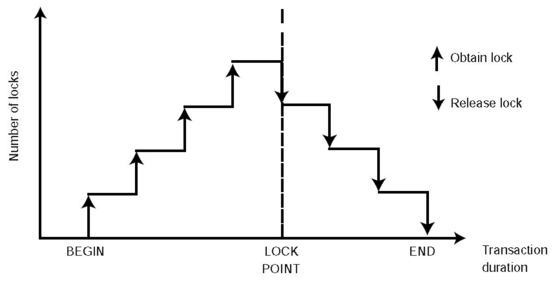


Figure 9. Growing and shrinking phases in 2PL Protocol [1]

b) Timestamp Ordering (TO) technique

In order to maintain transactions serialization in a Distributed Databases, apart from the Locking-based algorithm, there is a Timestamp-based algorithm. With this technique, to maintain the execution of transactions in serial order, on the initialization of transactions, the transaction manager assigns to each one a unique timestamp for its identification and for transactions ordering [23], [30].

The main rule is enunciated as follow: Suppose two conflicting operations, which can be read or write SQL commands in a transaction such that O_{ij} and O_{kl} respectively belonging to the transactions T_i and T_k, and *ts*, the Timestamp. O_{ij} is executed before O_{kl} if and only if $ts(T_i) < ts(T_k)$ in other words, T_i would be the oldest transaction and T_k would be the youngest. This protocol avoid completely *deadlocks* of transactions because even if $ts(T_i)<ts(T_k)$, the transaction T_k cannot *rollback* rather the operation O_{kl} will be rejected and the transaction manager would restart the whole transaction with a new Timestamp [1].

c) Hybrid technique

The visible limitation of Timestamp Ordering (TO) technique is that several restart of transactions can also influence negatively the performance of the system. There are other algorithms to attempt to improve this TO technique: Conservative TO and Multi-version TO Algorithms which aims to reduce the number of transaction restarts [1], [10], [22].

However, if one uses timestamp technique in locking-based algorithm in order to improve the concurrency level and efficiency, it should emerge the Hybrid technique. This technique should combine the advantages of 2PL algorithm and TO algorithm, in other word the notion transaction lock as well as the transaction timestamp. According to [1], this still being a challenge, since when it has never been implemented in any commercial DDBMS. But, some researches, [29], [30], [31], have already proposed Wait-Die (WD) &Wound-Wait (WW). This algorithm follows 2PL technique principals but overcome deadlock problem by applying the TO technique rule.

2. Optimistic or a posteriori approach execute the transaction without constraints and one moment of the validation one verifies that there are no conflicts with the other transactions. The optimistic approach of the concurrency control technique is appropriate in low conflict systems because the validation of each transaction for serialization, much like the pessimistic approach, can reduce performance. In these cases, the serialization test is adjourned just before validation because the conflict rate is low and the probability of aborting no serializable transactions is low as well [1].

Until today no research has yet attempting to implement the 2PL technique for the optimistic approach [10]. But according to [1], it would be possible to design a technique lock-based for this approach. Some of the proposed algorithms are timestamp-based and some have already been extended to DDBMSs [22]. One of them is Distributed Optimistic Protocol (OPT) [23], [29], [30], [31]. It is a time-based concurrency control algorithm that works by exchanging certification information during the commitment. For each data item, a read timestamp and a write timestamp are retained. OPT can provide the possibility of serialization under the restrictions imposed by two timestamps read and write basically assuming here above. If the concurrency check is not used, non-serializable execution orders can be generated by concurrent transactions.

a. Deadlock management

The 2PL protocol guarantees serializability property but does not prevent deadlock situations. Deadlock occurs when a transaction T_i is blocked waiting for resources R_i from another transaction T_j which in turn wait for resources R_j hold by T_i so that none process can complete [1], [4]. More generally, a deadlock can occur between *n* transactions [23]. The Figure 10 here below illustrate deadlock situation between two transactions T_i and T_j.

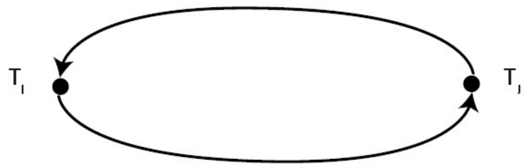


Figure 10. Wait for graph for deadlock process [1]

To manage this situation, two techniques are possible, a pessimist which prevent deadlock and an optimist for detecting deadlock [10], [28], [29], [30]:

✓ Deadlock prevention and avoidance:

- Deadlock prevention is an alternative method for resolving a blocking situation in which the system is designed so that blocking is impossible. In these schemas, the transaction manager checks a transaction during its initial launch and does not allow to perform a prior action in case of risk likely to cause a blockage.

- Deadlock avoidance is a technique which ensures that the blocking situation will not occur in a distributed system. The resource pre-command is a deadlock avoidance technique in which each data element in the database system is numbered and each transaction requests locks on those data elements in numerical order. This technique requires that each transaction gets all its locks before execution. The numbering of the data elements can be done globally or locally.

✓ Deadlock detection and resolution: we maintain a graph of dependencies between the transactions which makes it possible to detect the situations of deadlock (presence of a cycle in the graph). It is then necessary to kill one of the transactions participating in the deadlock. The choice is made heuristically (transaction with the least updates to undo, most recent transaction, ...). It is this technique that is implemented in commercial systems.

No solution is ideal and a choice must be made between the risk of occasional and unpredictable anomalies, and blockages and rejections that are just as punctual and unpredictable but which ensure the correction of concurrent executions [1].

b. Recovery management

Recovery is the process of ensuring that a database can achieve a reliable state in the event of failure. The failure recovery is, as the name implies, to ensure that the system is able, to recover the state of the database at the time the failure occurred. The basic unit of recovery in a database system is the transaction, i.e. on recovering time one should make sure that transactions display the properties of atomicity and durability [10]. The term fault here refers to any event that affects the operation of the processor or main memory. This could be, for example, an electrical interruption interrupting the data server, a software failure, or hardware failure. We will distinguish four types of failure (whatever the cause) [1], [22]:

✓ Transaction failures (aborts): usually occasioned by incorrect input and detection of deadlock. This conduce the transaction to abort and re-establish the database to the state before the initiation of the transaction;

✓ Site (system) failures: is caused mostly by hardware or software failure which leads to the loss of the main memory content while the secondary memory (disk) is safe and correct. The Site failure, the more usually make the concerning site unreachable in a distributed system;

✓ Media (disk) failures: is the failure of the secondary memory or disk which store the database. This can be due to disk crash or operating system errors. But since when the backup technique exists, the system can avoid such kind of catastrophe by recovering the database from the backup disk;

✓ Communication line failures: it results to network problems or destination site problems which can make impossible the communication and the transaction outcome. This issue is particularly identifying to distributed systems, but centralized one aren't very implicated. However, the strength of distributed systems is so that even if the communication line fail, it cannot affect all sites but some of them.

2.2.3.2 Status of the problem

Problem definition: Even though data replication presents perfect profits, it faces the challenge of keeping data copies synchronized. However, this problem, in simple way, consists of maintaining consistency among data copies or replicas as well as data availability [1], [46]. Data or Replica reliability is the domain presented by a set of data, in this case, Databases that contain the same information and placed on different nodes of the computer network when they are editable, they must all be updated (or synchronized) to maintain reliability [41]. Thus, it emerges two approaches [4]:

One is mutual reliability, which deals with the convergence of values of physical data elements that correspond to logical data. Mutual reliability of replicate databases can be strong or weak:

- ✓ Strong mutual reliability: Need that all data item copies contain the same data at the commitment of an update transaction.
- ✓ Weak mutual reliability: don't necessitate the data item replicas values to be same when an update transaction ends. In this way, it is necessary that when the update ends at a given moment, the data ends up becoming identical. This is normally called final reliability and this means that the replica data may not be the same over time, but eventually converge.

The second is transaction reliability is one of the transaction management proprieties which refer to activities of concurrent transactions. It is desired that the database keep a reliable state even if there are many read or update requests from users that are simultaneously submitted to it. On the other hand, to ensure the reliability of transactions, as it has been indicated here, this is the very objective of the concurrency control.

Updates propagation methods: Replica reliability is obviously a part of the data replication, which is in turn a method to overcome the problem of slow data access, low availability, fault tolerance, etc., in DDBSs. Previously it has been presented two general approaches to manage updates propagation that allowed the categorization of replication models. These strategies depend on "when" parameter i.e. we need to know when updates are propagated. Thus, it emerges two update strategies as follow [1], [4], [41], [46], [52]:

- ✓ Eager or synchronous or active or pessimistic replication: This method recommends that all data replicas be updated in the same transaction as the write transaction. This transaction typically presents itself as a basic Two-Phase-Commit (2PC) an atomic broadcast protocol. But after the operation all the replicas are coherent and bear the same physical state. Consequently, it is clear that disconnected sites can still block an update procedure because the 2PC protocol works by the principle according to which if a transaction is executed on multiple sites it must commit on all sites or abort on all these sites [1], [6]. This strategy provides strong mutual consistency or reliability.
- ✓ Lazy or asynchronous or passive or optimistic replication: The second method, in turn, introduces a new approach to overcome the difficulty of distributed locks. Its technicality prone to update a subset of replicas during the execution of an update transaction and then transmit the modification to the other replicas a little later. Only a part of replicas is updated. Other replicas are fetched up-to-date lazily after the commitments of the transaction. This process can be triggered by the commitment of the transaction or another periodically executing transaction. This approach provides weak mutual consistency or reliability.

These replication dimensions are orthogonal with "where" parameter i.e. we need to know where updates are going to take place. From this we have [1], [4], [41], [46], [52]:

- ✓ Single Master or Primary Copy or Mono Master with Limited or Full Replication Transparency (centralized): only one copy of the data is updated (master copy) and all others (secondary copies) are subsequently updated to reflect the changes in the master copy.
- ✓ Update everywhere or Multi Master (decentralized or distributed): updates are done on any data copy i.e. all sites that have a copy of the data can perform the data update and the changes are replicated to all other copies.

2.2.3.3 Types of replication protocols

Assume a fully replicated database and each site work under a Two-Phase-Locking (2PL) concurrency control technique. Therefore, we have four possible combinations [1], [4], [41], [46], [52]:

1. Eager centralized

In this approach, on a master site, operations (mostly write transactions) on a data element are conducted. These procedures are combined with strong consistency methods, so that the single update transaction which is committed using the Two-Phase-Commit (2PC) protocol performs logical data item updates.

First case: Single Master with Limited Replication Transparency

Let W(x) be a write transaction and R(x) be a read transaction where x is the replicated data item. The Figure 11, here below depicts the Eager Single Master Replication Protocol in the logic of Read-Any, Write-All (RAWA) or Read-One, Write-All (ROWA) using Time-Stamping Algorithm. The user submits the Write Transaction on the Master Site only and the Master Transaction Manager System forward synchronously these updates/changes to Slaves. A Read-only transaction can be submitted to anyone of Slaves Sites or the Master Site itself.

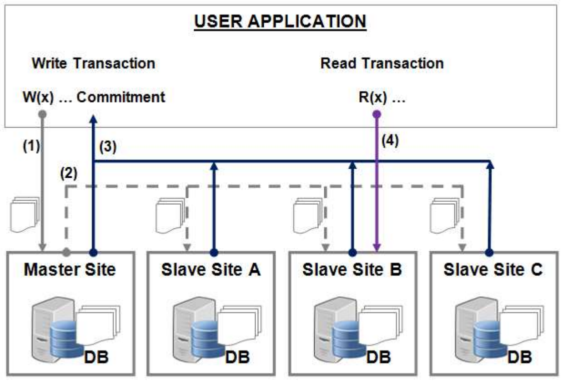


Figure 11. Eager Single Master Replication Protocol

(1) A Write transaction is performed on the DB on the Master Site;

(2) Write is then despatched to other replicas;

(3) At the commitment time updates become permanent;

(4) Read transactions are routed to any slave copy.

This case present one major drawback of overloading the master site by write transactions. As presented in the Figure 11, here above, every write transaction from each user application need to be deferred to the master copy before being dispatching to Slaves. Moreover, one important issue that persist is to make the difference between an “update” transaction and a “read-only”.

Second case: Single Master with Full Replication Transparency

This case overcomes the issue of Master overloading by Write Transactions from users. Thus apart from the Eager replica control algorithms coupled with Time-Stamping algorithm, the concurrency control which uses the coordinating Transaction Manager has been introduced.

However, the logic of RAWA or ROWA is still keeping but using Transaction Management algorithm. The user application is alleviated to know the Master Site. Even if the implementation of this case is more complicated than the first alternative discussed but it is responsible to provide Full Replication Transparency, keeping the same schema depicted in Figure 11, but using the Transaction Management algorithm.

Third case: Primary Copy with Full Replication Transparency

Let W(x) and W(y) be a write transactions and R(x) be a read transaction where x and y are replicated data items, successively first routed to Master(x) and Master(y). The Figure12, here below depicts the Primary Copy with Full Replication Transparency with the supposition of fully replication. A is the Master Site storing the data x and B and C are Slave Sites containing replicas; in the same way, C is the Master Site that holds the data y with B and D its Slave Sites.

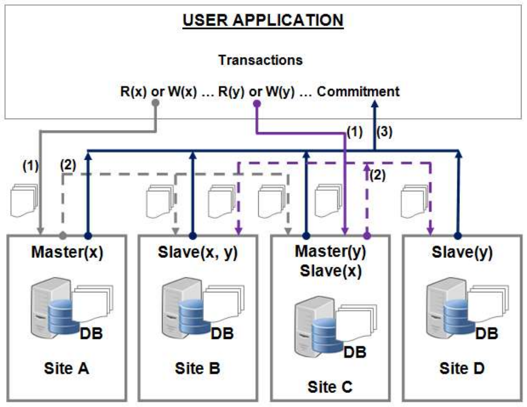


Figure 12. Eager Primary Copy Replication Protocol

(1) Read or Write transactions for all element of data are directed to that master of data items. A Write transaction is first performed at the Master;

(2) Updates are then despatched to the other replicas;

(3) The commitment of Updates is performed.

Primary copy method requires a sophisticated repertory at all sites, as well as a joint replica concurrency control technique, but it also overcomes some issues discussed in the previous approaches such that to reduce the load of the Master Site without producing a great volume of transmission among the transaction managers and the lock managers.

2. Eager distributed

Changes or updates can come from anywhere to first update the local replica and then send to other replicas. If the updates result from a site where there is no data element, it is sent to one of the replica sites, which in turn harmonizes the execution. The update transaction is responsible for fulfilling all these possibilities. The user is notified and the updates become permanent when the transaction commit.

Let W(x) be a write transaction where x is a data item duplicated at sites A, B, C and D. The Figure 13, here below depicts how two operations modify different copies (at two sites A and D). This procedure turns with the logic of Read-Any, Write-All or Read-One, Write-All constructed on the concurrency control techniques.

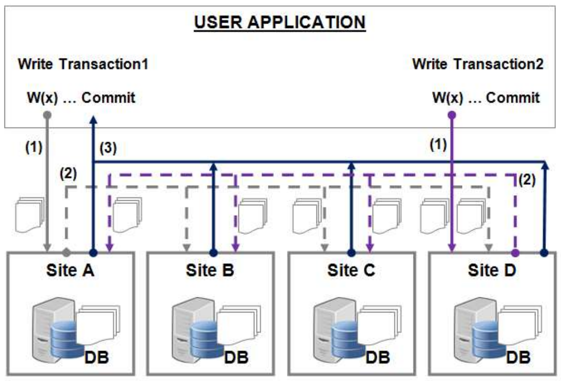


Figure 13. Eager Distributed Replication Protocol

(1) Two different Write transactions are simultaneously performed on two different local replicas of the same item of data;

(2) Write transactions are transmitted to the other replicas independently;

(3) At the time of commitment Updates become available.

3. Lazy centralized

This protocol provides algorithms which are alike eager centralized replication. But in this procedure updates or modifications are first performed to a Master replica and then transmitted to the slaves. The greatest dissimilarity is that the modifications or updates can't be dispatched through the update transaction, but forwarded by a separate refresh transaction to Slaves, asynchronously after the transaction commits. Thus, it is possible that a read transaction from any Slave, anytime, reads an out-of-date copy as the updates are pushed to Slaves one lapse long after the Master's update.

First case: Single Master with Limited Transparency

Let W(x) be a write transaction and R(x) be a read transaction where x is the replicated data item. The Figure 14, below, illustrates the sequence of execution steps for Single Master with partial sharpness. Here the write transactions are executed and deferred precisely on the main site (exactly as for Single Master). The second transaction, which we qualify as a refresh transaction, shares the updates to the slaves after validation of the first transaction. As soon as there is one master copy for all the data elements, the execution command is done according to the timestamp attached to each refreshing transaction at the site, according to the order of the commitment of the transaction. modification or actual update. Thus, in the timestamp order, Slaves would smear refresh transactions.

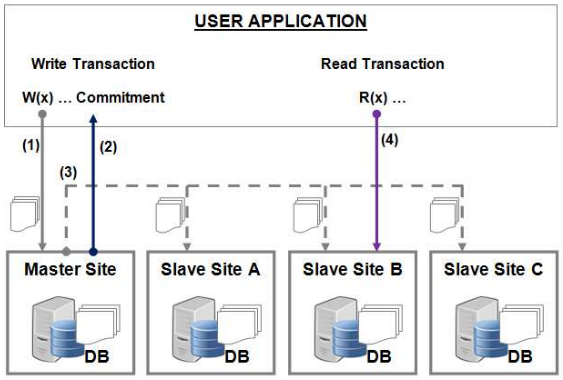


Figure 14. Lazy Single Master Replication Protocol

(1) The modification is performed on the local replica.;

(2) Updates become available as soon as transaction validation is successful;

(3) A refresh transaction propagates updates to other replicas;

(4) A read transaction is routed to a local copy of the slave.

When databases are partially replicated, a desired primary copy with a limited replication transparency approach makes sense if update transactions access only data items whose master sites are identical, since update are fully executed by a master. The same problem exists in the case of lazy primary copy, limited replication approach. The problem in both cases is how to design the distributed database so that meaningful transactions that can recognize the difference between an "update" and a "read only" transaction can be executed.

Second case: Single Master or Primary Copy with Full Replication Transparency

This protocol is an alternative that provides complete transparency by allowing the submission of read and write transactions to some site and then despatching them to the appropriate Master. This is delicate and involves two problems: the first is that, unless cautious, the global history in a serial order may not be definite; the second problem is that a transaction may not see its own updates.

So far, these two difficulties have found a partial solution: [1], proposed an algorithm respecting a chronological sequence of transactions executions. This algorithm is presented in the same way as that of centralized Eager, a primary copy with complete replication transparency case, but with the difference which is such that it makes it possible to retrace sequentially the history of serial transactions. Thus, a transaction does not start until the commitment of another, so lazily.

Although this algorithm manages the first problem, but the second according to which a transaction does not see its own scripts remains unresolved. To solve this problem, it has been advocated to keep a list of all the modifications made by a transaction and to consult this list when executing a reading. Nonetheless, as soon as only the master knows the updates, he deviated more to keep the list and all transactions (reading as well as writing) must be executed on the master.

4. Lazy distributed

In the control of lazy distributed replicas, updates come from wherever, they are first run on the local replica, and then propagated to other replicas later. In this way, the read and write transactions are executed on the local copy and the commitment of the transaction is locally before the refresh transaction propagate updates to other sites.

Let W(x) be a write transaction where x is a duplicated data item at sites A, B, C and D. The Figure 15, here below shows how two transactions modify or update two different copies (at sites A and D) and after commit the refresh forward updates to all sites.

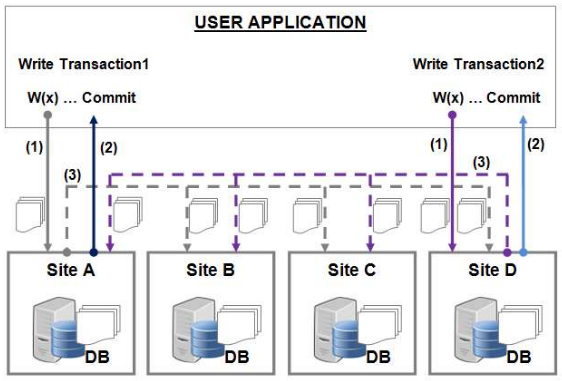


Figure 15. Lazy Distributed Replication Protocol Actions

(1) Two modifications or updates are performed on two local copies;

(2) Commitment of the transaction makes the modifications available;

(3) The modifications or updates are self-reliantly transmitted to the other replicas.

Once implemented, these protocol guarantees availability of updates although propagated instantly on all sites. Nevertheless, they leave the door open to investigation because they may present certain limitations that can be analysed in order to improve them. Although, these protocols establish the external appearance of replication; but one may need to know what's going on internally. On this preoccupation the answer is that internally there is synchronization procedures which are responsible to coordinate the exchange of data by running transactions in order to maintain replicas consistent. So the next section will present briefly the synchronization algorithm.

2.2.3.4 Synchronization algorithm

Data synchronization is part of the replication procedure that ensures that each storage object in the database contains the same data [8]. Mostly the synchronization is a technique used for working "online" and "offline", in the absence of the network or during server failures so that Data are stored in the user's site Database (Local Server), in order to be automatically synced with the server (Central Server) in serial order when the system recover from the failure [45], [56].

The synchronization procedure is mostly appropriate for lazy or asynchronous replication because it is this approach which can allow momently replicas discrepancies. Thus, it is a set of transactions which broadcast updates made on a Master toward a Slave (s) in near real-time.

Similarly, the synchronization procedure in Eager or synchronous replication use real-time transactions to broadcast updates. So it is from these approaches replication procedures which are used in most of DBMSs are working. They forward updates audit-log based, the combination of timestamp [1] and triggers [7] technique, stored procedure based [42], Message digest based [43], XML Based [45], etc.

1. Timestamp and current datetime

The timestamp is a data type in most of DBMSs so that if a table contain a timestamp column, each time a row is modified by an Insert, Update, or Delete statement, that column capture the change and the timestamp value of the row is set or the current value is incremented by 1. Basically, it is a combination of date and time plus a minimum of six positions for decimal fractions of seconds and an optional with time zone qualifier [55], [57]. The simple timestamp technique consists to concatenate the number of site with the local clock i.e. the combination of date and time so that when a row is modified the current date and time in the timestamp value also change. But one can also prefer to record the current dates and times of data changes in a table, using the datetime or smalldatetime datatype in a particular column. Thus, replication refreshing transactions can be scheduled to take place according to the timestamps and datetime comparison [1]. Certainly, the problem remains with the synchronization of the clock because there is no global agreement in time between sites in a distributed system.

2. Triggers

Typically, triggers are used to form the Audit Log of the database. This supports: Data Definition Language, to keep the history of activities that modify the physical schema of the database, Data Manipulation Language, to keep the history of the data modification and Data Control Language, to save the access history to the database by users [7]. In terms of database synchronization, the trigger is actually a stored procedure that execute a custom action before or after a certain event on database table records, such as Inserts, Updates and Deletions. Today almost all DBMSs support triggers, to keep records history. There are two types of triggers [42], [55], [57]:

- ✓ Before triggers are used to update or validate record values before saving them to the database;
- ✓ After triggers are used to access system-defined field values, and to apply changes to other records. Records that activate the after trigger are read-only.

Most synchronization procedures use this method to capture data changes in order to broadcast data updates. To achieve this, the table identifier or the primary key, the audit action type and the last timestamp of the audit table are required for a record. This is why the use of timestamp is very important for the comparison between records. The timestamp is used to check if there is inserted data, updated data, or deleted data from the synchronized database based on the identifier, the audit action and the timestamp.

3. Stored procedure

Stored procedures are code scripts that automate actions, which can be very complex. A stored procedure is actually a series of SQL statements designated by a name [55], [57]. They are stored in the database and used, just like all objects in the database. Once the procedure is created, it is possible to call it, by its name. The instructions of the procedure are then executed. Stored procedures can be initiated by a user or automatically by a triggering event [42].

4. Message digest

Only used for mobile database synchronization, the Synchronization Algorithms based on Message Digests (SAMD), is an algorithm which run data synchronization between a server side database and a mobile database. It allows data exchange between two message digest tables, one on the server-side and another on the mobile side. The message digest is a function which detects changes made on rows and facilitate the exchange of data in order to maintain consistency between the mobile database and the server database [43]. Other varieties of message digest algorithms are:

- ✓ ASWAMD: Advanced Synchronization Wireless Algorithm based on Message Digest, a synchronisation technique to assure data synchronization under the image format between the mobile device database and the server-side database. Based on an image stored in a message digest table, this algorithm compares two images and identifies the lines to be synchronized [71].
- ✓ ISAMD: Improved Synchronization Algorithms based on Secured Message Digest, like ASWAMD, this algorithm compares images and run their synchronization between the mobile device database and the server-side database. This algorithm does not use techniques that depend on specific database providers; it also does not use triggers, stored procedures, or timestamps. It uses only standard SQL functions to synchronize [72].

5. XML (Extensible Mark-up Language)

Many mobile synchronization techniques based on XML have already been developed. Same of them are presented here below [45]:

- ✓ Synchronization Mark-up Language (SyncML) is a mobile database synchronization technique based on XML, to carry messages over a network.
- ✓ DefferedSync is another synchronization technique to transform relational database into an XML tree structure and then makes use of deferred views in order to minimize bandwidth and storage space on mobile client.

Subsequently we will proceed to review also software structures which generally implement these replication protocols and all these techniques aforementioned here above. Thus, we will study the replication under some most used DBMSs.

2.2.3.5 DBMSs and types of replications

It has been necessary to select some most used DBMSs under which replications procedures ran, to properly investigate Data replication. This survey will examine the key feature and efficiency provided by Data replication in following Distributed Database Management Systems: Oracle DB, MySQL, SQL Server, and PostgreSQL. First of all, let us talk a bit about them and the types of replications they provide.

1. Replication in Oracle DB

Oracle DB is a Relational Database Management System (RDBMS) that since the introduction of object model support in version 8 can also be called Object Relational Database Management System (ORDBMS). It is provided by Oracle Corporation and incorporates the SQL database query language. It has the necessary tools for Database replication, which makes it a Distributed Object Relational Database Management System (DORDBMS). It is Multiplatform, it can allows following environments: Windows, Linux, Mac OS, and others [63]. Two forms of replication are supported by Oracle: basic and advanced replication [64], [65].

- Basic replication

Basic replication consists of a Master-Slave environment where updated data from the Master are propagated to Slaves for read-only. However, Slave-based applications may have access to query local replica data for read-only purposes, but in case the changes are needed, these applications must access data on the Master. Basic replication can be used for several types of applications: information distribution, information off-loading, and information transport. The main technique used here is read-only table snapshots that consist of a local copy of table data from one or more remote primary tables. To make this technique more efficient, Snapshot Refreshes is used to make this capture reflect a real state of its Master.

- Advanced replication

Advanced replication is the expansion of basic replication capabilities in that it allows applications to update table replicas in a replicate database system. Replicas of data from any site in the replication environment can be accessed for read and write. So, Oracle database servers which make up the replication system must automatically converge the data replicas and ensure the overall consistency of the transactions and the integrity of the data.

Advanced replication can be used for several types of applications with special requirements: Disconnected Environments, Failover Site, Distributing Application Loads and Information Transport. Advanced replication allows basic components replication such as: objects, groups, sites, and catalogues. To make easy the replication, Oracle DB provides an administrative tool, “Oracle Replication Manager”, to run Advanced replication. Oracle require a proper replication administrator, a user whose responsibility consists of setting up the replicated environment and for this it require a specific user account.

Despite all these assets, advanced replication process leaves a challenge of possible conflict as it allows update anywhere. One can distinguish 3 types of conflicts [65]:

- ✓ Uniqueness conflicts due to entity integrity violation (primary key, foreign key, unique constraint);
- ✓ Update conflicts due to attempting to update a row which is being updated by another process;
- ✓ Delete conflicts due to attempting to delete or update a row which is being deleted by another process.

However, Oracle integrate some technique for detecting and resolving conflicts. To detect conflict, it runs the comparison of amount of row from the Master site and the Slaves site; if there is the difference, the conflict is detected. The second way consists of the recommending the usage of the primary key in order to identify records. If a table don’t have a primary the user must designate an alternative key. Advanced replication is essentially asynchronous. This technique uses mechanisms of high concurrency control to resolve conflicts between transactions at the Slave site. Nevertheless, this replication can be synchronous if and only if an application updates all replicas in the same transactions directed to the local replica. Advanced replication in turn supports the following replication configuration types [65]:

a) Peer-to-Peer replication

Peer-to-Peer replication, also known as Multi-Master replication, allows multiple equal peers from different sites to manage replicate database objects. Thus, applications access and update any replicate data from tables stored on any peer in the replication environment. Peer-to-Peer replication uses the asynchronous method to propagate peer updates.

b) Materialized View replication

The Master sites of a replication system can consolidate the information updated by the applications on remote snapshot sites. In this logic, applications can make insertions, updates and deletions of table rows via snapshots. These kinds of remote snapshots are called Updatable Snapshots or Materialized Views, which gives this type the name of Materialized View Replication.

c) Hybrid replication

Peer-to-Peer replication combined with Materialized View Replication gives rise to hybrid or "mixed" configurations to meet different application requirements. In these kinds of configurations, one can have any number of master sites and more than one view site materialized for each master.

2. Replication in MySQL

MySQL is a Relational Database Management System (RDBMS), one of the most widely used database management software in the world, both by the general public (mainly web applications) and by professionals. SQL refers to the Structured Query Language, the query language that it uses. It was purchased from Sun Microsystems by Oracle Corporation. As a result, it holds two competing products, Oracle Database and MySQL. It is Multiplatform, it can allows following environments: Windows, Linux, Mac OS, and others [69]. It owns also the Data replication strategy which make it to be a Relational Distributed Database Management System (RDDBMS).

In MySQL, the replication is a mechanism for copying data from a source or master MySQL database server to one or more other destinations, MySQL databases servers or slaves. So this replication is Mono-Master/Multi-Slaves. By default, replication in MySQL is asymmetrical and asynchronous, that is to say it only allows updating to be done in near real-time in one direction. The permanent connection between slaves and master is not recommended to receive updates from the master, so if an update transaction finds an unconnected slave, the system has the option to update it time as soon as he connects i.e. lazily. With MySQL replication can be customized in two ways but keeping the idea of the principle Mono-Master/Multi-Slave: replication of all databases and replicate selected databases or even select tables in a database [70].

3. Replication in SQL Server

Microsoft SQL Server is a Relational Database Management System (RDBMS) which incorporate SQL language developed and marketed by Microsoft. It runs on Windows and Linux since March 2016, but it is possible to launch it on Mac OS after downloading some necessary components [60]. It assures the distribution of Databases by the replication strategy warrantied by synchronization procedures to maintain consistency among Databases objects. This is why it is a Relational Distributed Database Management System (RDDBMS). Types of replication provided by SQL Server for use in distributed applications are following:

- Snapshot Replication

This replication consists in taking a snapshot of the data published in the database (the publisher) and move them to another, which may or may not be stored in the same machine (the distributor), in order that the distribution agent transmits in turn these data to other databases (subscribers) periodically based on the specified schedules. This type of replication requires little work overload for the publisher server because the operation is punctual. Subscribers are updated by copying all published data rather than just making the changes (Insert, Update, and Delete) that occurred [61], [62].

This replication is well suited for small volume publications; otherwise subscriber updates may require significant network resources. Snapshot replication is often used when subscribers need access to read only information and they do not need to know the information in real time. The Snapshot Agent is responsible for performing the job to prepare the files containing the schemas and data from the published tables. These files are stored on the distributor [13].

- Transactional replication

It is used to replicate objects in a database. Transactional replication uses the log to recognize changes occurred to published data. These changes are stored first on the distributor before being propagated to subscribers as they occur (i.e. in real time) in the publication, to ensure transactional consistency. This is a dynamic replication mostly used in Server-to-Server replication [61], [62].

This type of replication requires low latency between sites when changes are made to the publisher and the changes arrive at the subscriber. So in an environment where connections are optimal, the latency between the publisher and subscribers can be very low. The publisher has a very large volume of Insert, Update, and Delete activities because as database users insert, update, or delete records on the publisher, transactions are transmitted to subscribers. The publisher or subscriber can have different types of DMBS [13].

- Merge replication

Merge replication allows two or more databases to be synchronized. All changes applied to a database are automatically transferred to other databases and vice versa. It allows data modifications on the publisher and the subscriber, but also allows offline scenarios i.e., it can allow synchronization to take place automatically between the subscriber and the publisher after a subscriber has been disconnected from an editor for a given period. And here, the Merge Agent is responsible for synchronizing changes between the publisher and its subscribers [61], [62].

The logic of operation remains the same with that of the snapshot replication, with the difference that is such that it uses a set of triggers to identify the items (records) that have changed and save these changes finally that the merge agent uses this history to update subscribers [13].

- Peer-to-Peer Transactional replication

SQL Server peer-to-peer replication ensures high availability and a scaling solution that maintains multiple copies of data across multiple peers (servers). It is based on transactional replication and broadcasts updates by consistent transactions in near-real time. It advocates redundancy of peer data to increase availability. In a peer-to-peer replication system, read performance on a peer is similar to that of the entire topology because changes from insert, update, and delete operations are propagated to all peers [61].

The main problem with Peer-to-Peer replication in SQL Server is that modifying a record on more than one peer causes a conflict or loss of update when propagating updates. To ensure consistency, it is recommended that a record be updated by one and only one peer. In addition, when dealing with an application that requires immediate visibility of the last changes, there should be a problem of dynamic load balancing between multiple peers. Also, as far as conflict management is nowadays concerned in almost all research about peer-to-peer replication, this also has an option to detect and to avoid conflicts and loss of updates. Unfortunately, this feature is not yet very effective, especially since the resolution consists of treating the problem as a critical error that causes the distribution agent to fail; and finally the data remains inconsistent until a manual resolution is made throughout the topology [13].

4. Replication in PostgreSQL

PostgreSQL is a Relational and Object Database Management System (RODBMS). It owns features for Database replication, which make it a Distributed Object Relational Database Management System (DORDBMS). It is a free tool and available according to the terms of the license used. This system competes with other free or commercial DBMSs due to its availability. Like the Linux operating system free project, PostgreSQL is not controlled by a single company, but rather based on a global community of developers and companies. It is mostly used as an open source relational database chosen by many organisations and people for their experimentations. It runs on almost all operating system. The origins of PostgreSQL are based on the Ingres database, which was developed by Michael Stonebraker in Berkeley. He decided in 1985 to restart the development of Ingres to finally arrive at a new product that he named Postgres, shortened post-Ingres. In 1995, the SQL features had been added and the product was renamed Postgres95. This name was finally changed in PostgreSQL [66]. In PostgreSQL, replication can be classified in 3 generic ways [67], [68]:

- Synchronous and asynchronous replication

PostgreSQL asynchronous replication take place after the transaction has been committed on the master. In other word, the slave is never ahead of the master; and in the case of writing, it is usually a little behind the master and this delay it takes to forward data from the master to a slave is called lag. The Synchronous replication is considered in

PostgreSQL as a method to enforce rules of high consistency by running a write transaction from one server (the master) and the same transaction before commit, it updates at least two servers at the commitment time. This implies that the slave does not lag behind the master and the data seen by the end users will be identical on both the servers. In some cases, the quorum server decision is used to commit the transaction; in this way, more than a half of servers must agree so that the transaction commits.

According to S. Hans-Jürgen [67], synchronous replication still produces overhead since when if a transaction is replicated synchronously, PostgreSQL has to reassure that the data reaches the destination node. This lead to latency issues whose a lot of works have been done to reduce this overhead as much as possible, but efficient solution is not yet reached. Another more problem he pointed out about asynchronous replication, is the case when the Master dies before forwarding updates carried to it by local committed transactions. The Slave will never get these updates. The small lag is required to reduce data loss, but in any case, lag cannot be equal to zero, it is always more than zero and lag larger than zero is susceptible to data loss and sacrifice the consistency. This problem also need an effective solution.

- Single-Master and Multi-Master replication

PostgreSQL provide Single-Master replication which consists to direct all updates on the Single-Master so that this last forwards these updates synchronously or asynchronously to the Slave(s). In turn the Multi-Master replication allows updates to be done anywhere. So as writes can go toward many nodes at the same time, possible conflicts can be known between replicas; which is the main challenge of this configuration because the conflicts resolution increases network traffic which finally turn in scalability issue causes by latency.

- Logical and physical replication

The Logical replication is which is based on the flow of logical data like for instance data which are provided by function while physical replication is based on the flow of data as it is to the remote node.

5. DBMSs replication critical analysis

The features to which this investigation has been fixed are the following: the approaches of updates propagation, the direction of updates propagation and the configuration (number of Masters and different DBMSs supported).

Table 2. Replication Approaches and Directions

Approaches / DBMSs	Synchronous	Asynchronous	Symmetric	Asymmetric
Oracle DB	Yes	Yes	Yes	Yes
MySQL	No	Yes	No	Yes
SQL Server	Yes	Yes	Yes	Yes
PostgreSQL	Yes	Yes	No	Yes

This table 2 presents the directions and replication approaches based on the "when" parameter in the four aforementioned DBMSs. SQL Server and Oracle DB answer all questions positively.

Table 3. Replication Confifurations

Configurations / DBMSs	Mono-Master	Multi- Master	Mono-DBMS	Multi-DBMS
Oracle DB	Supported	Supported	Perfect	Not Perfect
MySQL	Supported	Supported	Perfect	Not Perfect
SQL Server	Supported	Supported	Perfect	Not Perfect
PostgreSQL	Supported	Supported	Perfect	Not Perfect

This table 3 shows the replication configurations based on the "where" parameter in the four above-mentioned DBMSs. However, almost all four DBMSs can support Multi-Master replication. But as for the interoperability between DBMSs, when it is done, it is after a long journey of settings, which remains a work of insiders alone. At this level, it is necessary to consider a platform that can guarantee cooperation between DBMSs from the point of view of data replication.

The replication systems offered by these four DBMS investigated above are based on generic replication protocols. Of course each corporation providing a DBMS possessing replication tools, as Microsoft and Oracle have already programmed these tools to address various problems, in different ways and under different circumstances. But the bedrock of reflection remains the same, namely the two main approaches to replication based on the parameters "when" and "where". During this investigation, the attention remained more focused on the parameter where and more specifically the Multi-Master configuration or Update everywhere.

Microsoft and Oracle have already managed to wrap the Multi-Master configuration to make it a Peer-to-Peer replication. For PostgreSQL, being a free DBMS open to all, its replication remains generic and this work is not done yet, so it still offers Multi-Master replication from the generic protocol of decentralized replication. Nevertheless, everywhere, two major challenges were highlighted, namely: the possible conflicts between the updates from several Masters at the same time and the blocking due to the simultaneous update of the same replica by several Masters at the same time. Thus, these two problems deserve special attention and will be clear in the lines that follow.

2.2.3.6 Deficiencies collected

We are not pretending clear all the collection of issues about the replication procedure. But here we present some problems we have been able to collect during the literature conducing through this Distribution strategy and for which we didn't found effective solution.

Disaster management differs between Eager replication and Lazy replication methods. This last case is reasonably easy because these procedures allow for data discrepancy between master copies and replicas, because when communication failures make one or more sites inaccessible, accessible sites only can simply continue processing.

However, it is also clear that more than one update, carried by refreshing transactions, from different sites can reach a destination site at the same time. This needs an efficient serialization algorithm. Moreover, these updates can be performed on different sites, simultaneously on different copies of the same data item. This calls for an efficient algorithm to reconcile updates. According to [1], this algorithm can be based on heuristics and in this logic he gave the example of the importance of the transmitter site in the hypothesis where there are sites whose updates are more urgent than those of others.

But, the problem remains with Eager protocol since when it implements the ROWA procedure, which ensures that all of the replicas have the same value when the update transaction commits. An alternative to Read-One / Write-All (ROWA) that should attempt to solve the problem of low availability is the Read-One / Write-All Available procedure (ROWA-A). The general idea is that write transactions are performed on all available copies and the operation ends. The copies those were unavailable at the time when the transaction ran, will have to "catch up" when they become available. This also needs an effective approach which will remove this limitation.

Centralized update propagation techniques, Eager and Lazy, as shown in Figure 11, 12, and 14, present a major problem that is such that they only offer a single gateway, their Masters (Central Servers), which are the bottlenecks of everything over the network; because updates or modifications are first performed at a Master copy and then propagated to Slaves, (Clients). In this way the main disadvantage is that, as in any centralized procedure, if there is one central site that swarms all Masters, this site can be encumbered and can become a hold-up. Distributed update propagation techniques, Eager and Lazy have overcome this limitation in the sense that updates can originate and be forwarded from any site.

But, in order to overcome the limitations of Homogeneous Distributed Database Systems (HDDSs), early work on DDBMSs had primarily concentrated on Peer-to-Peer (P2P) architectures. In this approach there isn't the difference between nodes or sites in the system. The modern Peer-to-Peer systems go beyond the simple description and diverge from the old one by important ways like: the massive distribution (thousand sites), sites heterogeneity and autonomy, sites are rather often i.e. people's individual machines and they join and leave the P2P system at will, etc. [1], [44].

First of all, let us notice that obviously the Centralized update propagation techniques, Eager and Lazy problems are same with what knew Centralized Peer-to-Peer Architecture [44], which work based a Central Server and Clients (Peers), as shown in the Figure 16. The request is submitted by a Peer "A" to the Central Server so that it provide a list of nodes that satisfy to its demand. As soon as the

Peer "A" obtains the list which repays the Peer "B" then he can communicate with "B". So it would be enough that this Server knew a breakdown to block or to disconnect all Clients and to stop the operation of the whole system because no Peer will have the data no longer updated.

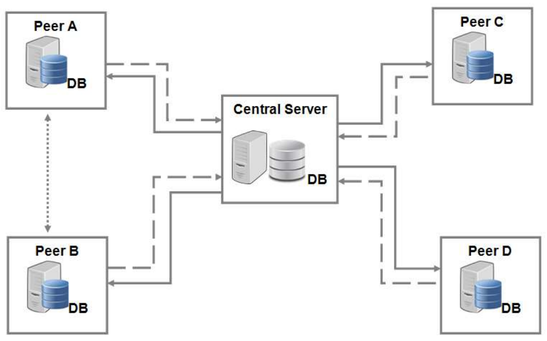


Figure 16. Centralized Peer-to-Peer Architecture

Therefore, it is necessary to design a synchronization algorithm that can update several copies of Databases at the same time. And each machine in its roles will be identical to another, and then we should call this type of system Decentralized Pure Peer-to-Peer Synchronizer.

The replication protocols we have discussed here above are appropriate for closely integrated Distributed Database Systems where the protocols are inserted into each DBMS components. So since we advocate designing modern P2P system, we have to remove the limitation of particular DBMS and thing in the sense of multi-DBMS Synchronizer. In multi-DBMS, the process of replication has to be supported outside the DBMSs by "Mediators". In this way this synchronisation algorithm should be implemented independently of DBMSs as a Mediator or may be used by Distributed Databases and applications designers since they will need interaction between different DBMSs.

3. PROPOSED MODEL

After the course of the literature above offered by our predecessors in this field, we realized however that Distribution issues can be categorised in two based on distribution strategies: Fragmentation and Allocation from one side and Replication from another side. In previous section we have presented more than one approach, from previous works, to resolve efficiently each of every one of these problems.

So, in general, analyses from previous work concluded that in the field of distributed database systems the problems of data distribution, which are grouped in two: fragmentation and allocation of fragments on one hand and data replication on another hand, have effective solutions when the number of sites is still very limited and the sites are still static and configured in a homogeneous way. But with the arrival of Peer-to-Peer (P2P) network, the efficiency still far from being found. Hence, the new research in the field must take into account this new technology that is in full emergence, where peers can be business servers, personal computers or even pocket computers and other electronic devices.

But particularly, the replication problem has retained our attention; mostly while synchronization, which is the process of propagating modifications or updates to sites that hold the replicas of the fragment or replicas of the whole relation, in the case of full replication, is running.

3.1 Status of the Problems and Proposed Solutions

Assuming that the Database is full replicated, the proposed models would resolve following problems:

- ✓ Several updates carried by refreshing transactions, from different sites can reach a destination site at the same time but they cannot be performed on the same time. Proposed solution: an effective serialization algorithm [10], [53].
- ✓ These updates can be performed on different sites, simultaneously on different copies of the same data item, if they reach the destination like that then reliability or consistency will be lost and there will be the risk of conflicts [10], [48]. Proposed solution: an effective algorithm to reconcile updates.
- ✓ During Eager or synchronous replication which is essentially Read-One/Write-All based, if some copies were unavailable at transaction running time, the update transaction can't commit. But normally the transaction should commit and unavailable sites should get updates when they become available [10], [54]. Proposed solution: an effective approach taking in account Read-One/Write-All-Available.
- ✓ These protocols suffer from many problems which the introduction of the modern P2P systems should overcome. Some innovations of the modern P2P are: sites autonomy in the sense that each machine in its roles is identical to another, play same roles, and can leave and join the Network anytime. Proposed solution: an effective approach for synchronization over a decentralized P2P architecture, as shown in

Figure 17, that can update several copies of the databases in real-time or near real-time.

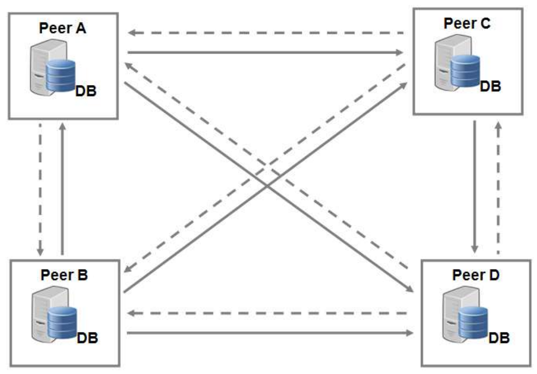


Figure 17. Decentralized Peer-to-Peer Architecture

✓ Moreover, this approach should overcome one aspect of Distributed Databases homogeneity in the sense that it should allow replication between different DBMSs. So this synchronisation algorithm should be implemented independently of DBMSs as a Mediator, illustrated in Figure 18, or used by designers when they need interaction between Databases managed by different DBMSs.

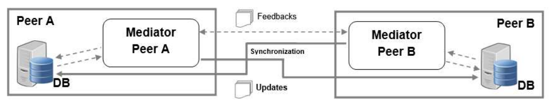


Figure 18. Peer-to-Peer mediator-synchronizer protocol

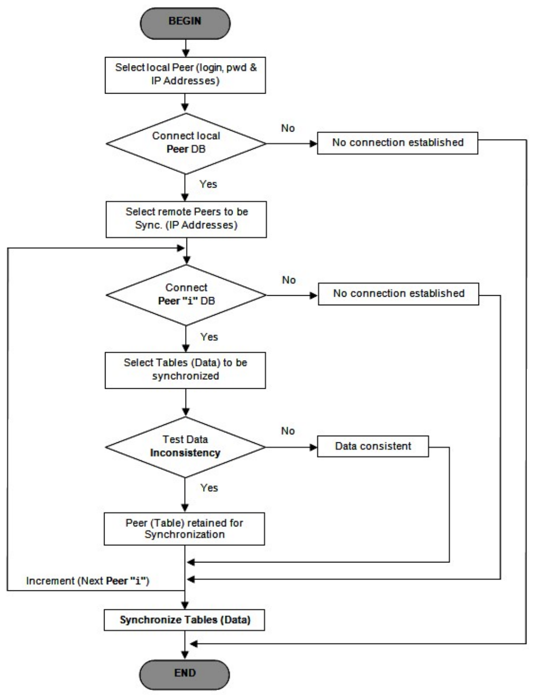

BEGIN
Select local Peer (login, pwd & IP Addresses)
Connect local Peer DB
No
No connection established
Yes
Select remote Peers to be Sync. (IP Addresses)
Connect Peer "i" DB
No
No connection established
Yes
Select Tables (Data) to be synchronized
Test Data Inconsistency
No
Data consistent
Yes
Peer (Table) retained for Synchronization
Increment (Next Peer "i")
Synchronize Tables (Data)
END

Figure 19. Peer-to-Peer synchronization Algorithm

The preliminary operations of this algorithm are depicted in the Figure 19 as follows:

Step1. Select the local Peer and connect on it by providing the login, password and the IP address (facultative): if these provided parameters are incorrect then no connection established else next step;

Step2. Select remote Peers, by indicating theirs IP addresses, to be sync and test connection with them one by one: if Peer non-jointed then no connection established, next Peer else next step;

Step3. Select Tables (Data) to be synchronised and test Data inconsistency: if Data consistent then next Peer else Peer (Table) retained for Synchronization, next step;

Step4. ***Synchronize Tables (Data)*** of all retained Peers at the same time.

3.2 Proposed Protocols

Assuming that the Database is full replicated, the proposed models of the Decentralized Peer-to-Peer Architecture are presented based on Eager replication and Lazy replication as follows:

✓ Eager replication: Let W(x) be a write transaction where x is a replicated data item at Peers A, B, C and D. The Figure 20, here below depicts how transactions update different copies at all Peers and before commit the refresh forward updates to all Peers.

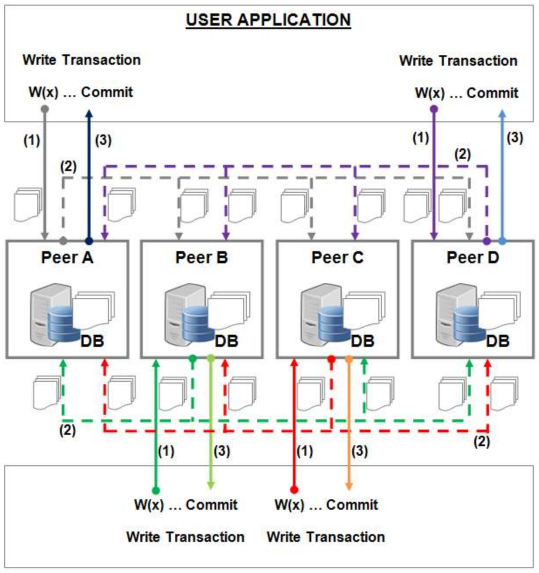


Figure 20. Eager Decentralized Peer-to-Peer Replication Protocol

(1) Updates are applied on all replicas;

(2) The updates are dependently propagated to the other available replicas;

(3) Transaction commit makes the updates permanent.

✓ Lazy replication: Let W(x) be a write transaction where x is a replicated data item at Peers A, B, C and D. The Figure 21, here below depicts how transactions update different copies at all Peers and after commit the refresh forward updates to all Peers.

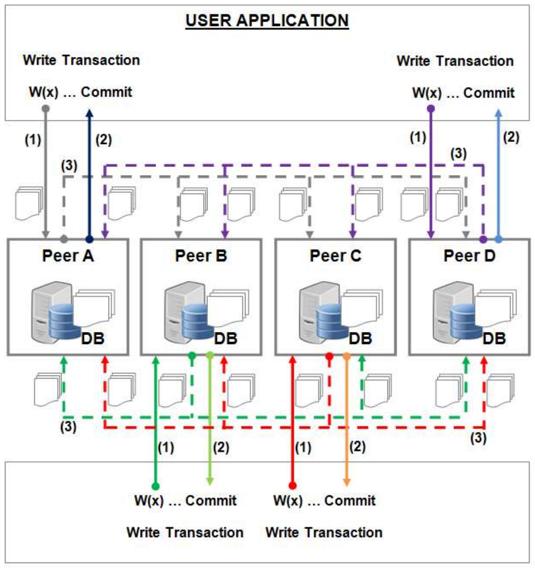


Figure 21. Lazy Decentralized Peer-to-Peer Replication Protocol Actions

(1) Modifications are reflected to all replicas;

(2) The commitment of a transaction makes the modifications stable;

(3) The modifications are independently transmitted to the other data copies or replicas.

This work differs from others in the sense that it set itself as a goal to completely review the literature on distributed database systems in order to find out if existing distribution approaches remain effective when in full migration to P2P networks. Focusing first on replication, it has been found that existing replication approaches are not appropriate for supporting replication on a P2P network. Thus, it has been proposed new replication approaches adapted to the P2P network.

4. CONCLUSION

In this paper, literature survey has been conducted on Distributed Databases and their techniques. Nonetheless in this relevant literature, distribution strategies and some problems encountered when designing and using distributed Databases have been pointed out. These problems have been collected based on respectively three distribution strategies: Data fragmentation, Data allocation and Data replication.

First of all, Data fragmentation has been analysed and our attention has been retained by the join optimization problem since when this problem occurs when executing a query combining more than one fragment stored on different sites. In this way time response become high when the query has to concatenate fragments by join. This problem is known to be a NP-Hard one; so the exploration of some existing heuristic approaches, as solution, has been necessary.

Since Data or relations become fragmented the next step is to allocate these fragments to sites. Thus Data Allocation is also another particular problem which involves finding the "optimal" distribution of fragments to sites. This has already been proved to be a NP-complete Problem. Its solution consists to heuristic methods. So during this study the review of some heuristic that yield suboptimal solutions have been done.

On finish as fragments or whole relations have in certain cases to exchange data among them, the Data replication, which is the unique strategy to manage this procedure, has been studied and its famous synchronization algorithm. The main problem here is to maintain consistency among fragments or whole relations on different sites. However, in this literature, we have indicated some issues which can violet consistency, such that: no serialization of update transactions, no reconciliation of updates, no update of unavailable replicas in Eager or synchronous replication, no sites autonomy, no independent effective synchronization algorithm which can play the role of Mediator between different DBMSs.

Despite the correctness of all protocols studied earlier, since these problems just indicated here above are not solved then consistency can be broken anytime in replicated database systems. Thus this has been our motivation to propose an effective approach for synchronization of distributed databases over a decentralized peer-to-peer architecture.

As future work we will develop a complete and DBMS independent algorithm in which it will be presented step by step scenarios to synchronize database tables. It will be implemented as a prototype in a Graphical Interface User, as a Mediator of DBMSs, to attempt to reach the aims of modern Peer-to-Peer in Distributed Database Systems.

REFERENCES

[1] Özsu MT, Valduriez P. Principles of distributed database systems. *Springer Science & Business Media*; 2011 Feb 24.

[2] Shareef MI, Rawi AW. The Customized Database Fragmentation Technique in Distributed Database Systems: A case Study.2011.

[3] Kaur K., Singh H., “Distributed database system on web server: A Review”. *International Journal of Computer Techniques*, Vol. 3, pp. 12-16, 2016.

[4] Souri A, Pashazadeh S, Navin AH., “Consistency of data replication protocols in database systems: a review”, *International Journal on Information Theory (IJIT)*, October 2014, 3, 19-32.

[5] Idowu SA, Maitanmi SO. Transactions-Distributed Database Systems: Issues and Challenges. *International Journal of Advances in Computer Science and Communication Engineering (IJACSCE)* Mar 2014; 2:24-6.

[6] Jain G. Distributed Data Management: Challenges and Solution on Distributed Storage. *IJCER*. 2016 May 14;5(2):24-7.

[7] Gudakesa R, Sukarsa I, Sasmita A, Made Ig. Two-Ways Database Synchronization In Homogeneous Dbms Using Audit Log Approach. *Journal of Theoretical & Applied Information Technology*. 2014 Jul 31;65(3).

[8] Malhotra N, Chaudhary A. Implementation of Database Synchronization Technique between Client and Server. *International Journal Of Engineering And Computer Science*. 2014 Jul 28;3(07).7070-7073.

[9] Tomar P. An overview of distributed databases. *International Journal of Information and Computation Technology*. 2014 Feb;4(2):207-14.

[10] I. Singh and S. Singh, Distributed Database Systems: Principles, Algorithms and Systems, New-Delhi, India: *Khanna Book Publishing, Co. (P) Ltd*, 2015.

[11] Hiremath DS, Kishor SB, Distributed Database Problem areas and Approaches, *Journal of Computer Engineering: National Conference on Recent Trends in Computer Science and Information Technology*, 2016, 2278-8727.

[12] Gadicha AB, Alvi AS, Gadicha VB, Zaki SM. Top-Down Approach Process Built on Conceptual Design to Physical Design Using LIS, GCS Schema. *International Journal of Engineering Sciences & Emerging Technologies*. 2012;3:90-6.

[13] Microsoft Corporation, SQL Server Replication, from Microsoft Documentation: https://docs.microsoft.com/en-us/sql/relational-databases/replication/sql-server-replication, Retrieved September 2017.

[14] Truica CO, Boicea A, Radulescu F. Asynchronous replication in Microsoft SQL Server, PostgreSQL and MySQL. *InInternational Conference on Cyber Science and Engineering* (CyberSE’13) 2013 (pp. 50-55).

[15] Souri A, Pashazadeh S, Navin AH. Consistency of Data Replication protocols in database Systems: A review. *International Journal on Information Theory (IJIT)*. 2014 Oct;3(4):19-32.

[16] Rouse M., Backup and recovery, from TechTarget: http://whatis.techtarget.com/glossary/Backup-and-Recovery, Retrieved September 2017.

[17] Verma SK. Fragmentation Techniques for Distribution Database: A Review. *International Journal of Innovative Computer Science & Engineering*. 2016 Apr;3(2):47-50.

[18] Kaundal G, Kaur S, Vashisht S. Review on Fragmentation in Distributed Database

Environment. *IOSR Journal of Engineering*, ISSN (e). 2014 Mar:2250-3021.

[19] Salunke IT, Potdar P. A Survey Paper on Database Partitioning.A Survey Paper on Database Partitioning", *International Journal of Advanced Research in Computer Science & Technology*, 2014 2(3), 210-212.

[20] Bhuyar PR, Gawande AD, Deshmukh AB. Horizontal fragmentation technique in distributed database. *International Journal of Scientific and Research Publications*. 2012 May;2(5):1-7.

[21] Kelemu Y, Patil S. Hybridized Fragmentation of Distributed Databases using Clustering.Hybridized Fragmentation of Distributed Databases using Clustering. *International Journal of Engineering Trends and Technology*, July 2016 Vol. 37(1), 1-5.

[22] Ceri S. Distributed databases. Tata McGraw-Hill Education; 2017.

[23] Ashish MK. Security and Concurrency Control in Distributed Database System. *International Journal of Scientific Research and Management*. 2014 Dec 5;2(12).

[24] Srivastava A, Shankar U, Tiwari SK. Transaction management in homogenous distributed real-time replicated database systems. *International Journal of Advanced Research in Computer Science and Software Engineering*. 2012 Jun.

[25] Boicea A, Radulescu F, Truica CO, Urse L. Improving Query Performance in Distributed Database. *Journal of Control Engineering and Applied Informatics*. 2016 Jun 23;18(2):57-64.

[26] Johnsirani B, Natarajan M. An Overview of Distributed Database Management System. 2015; 2(5):118-121

[27] Beynon-Davies P. Distributed Database Systems. InDatabase Systems 1996 (pp. 107-115). Palgrave, London.

[28] Bamnote GR, Joshi H. Distributed Database: A Survey. *International Journal Of Computer Science And Applications*. 2013;6(2):0974-1011.

[29] Bernstein PA, Goodman N. Concurrency control in distributed database systems. *ACM Computing Surveys (CSUR)*. 1981 Jun 1;13(2):185-221.

[30] Gupta AM, Gore YR. Concurrency Control and Security Issue in Distributed Database System. *International Journal of Engineering Development and Research*. 2016;4:177-81.

[31] Kaur M, Kaur H. Concurrency control in distributed database system. *International Journal of Advanced Research in Computer Science and Software Engineering* ISSN. 2013 Jul;2277.

[32] Ip A, Rabayu W, Singh S. Query optimisation in a non-uniform bandwidth distributed database system. InHigh Performance Computing in the Asia-Pacific Region, 2000. Proceedings. *The Fourth International Conference/Exhibition* on 2000 May 14 (Vol. 2, pp. 818-823). *IEEE*.

[33] Umar YR, Welekar AR. Query Optimization in Distributed Database: A Review. Query Optimization in Distributed Database: A. 2014.

[34] Tosun U, Dokeroglu T, Cosar A. Heuristic algorithms for fragment allocation in a distributed database system. InComputer and Information Sciences III 2013 (pp. 401-408). Springer, London.

[35] Amer AA, Abdalla HI. A heuristic approach to re-allocate data fragments in DDBSs. InInformation Technology and e-Services (ICITeS), *2012 International Conference* on 2012 Mar 24 (pp. 1-6). *IEEE*.

[36] Kamali S, Ghodsnia P, Daudjee K. Dynamic data allocation with replication in distributed systems.Performance Computing and Communications Conference (IPCCC), *IEEE 30th International*, Orlando, FL, USA, November 2011.

[37] Jiang S, Ferner C, Simmonds D, Reinicke B, Clark U. Optimizing Join Query in Distributed Database. Annals of the Master of Science in Computer Science and Information Systems at UNC Wilmington. 2011 Apr 26;5(1).

[38] Mahajan SM, Jadhav VP. Tri-variate Optimization Strategies of Semi-Join Technique on Distributed Databases. *International Journal of Computer Applications*. 2013 Jan 1;66(6).

[39] Kaur P, Sahiwal JK. Join Query Optimization in Distributed Databases. *International Journal of Scientific and Research Publications*. 2013;3(5):1-3.

[40] DBConvert Help Center, Bidirectional Database Synchronization, from DMSoft Technologies Articles: https://support.dbconvert.com/hc/en-us/articles/201210922-Bidirectional-Database-synchronization, Retrieved October 2017.

[41] Pucciani G, Donno F, Domenici A, Stockinger H. Consistency of Replicated Datasets in Grid Computing. InHandbook of Research on Grid Technologies and Utility Computing: Concepts for Managing Large-Scale Applications 2009 (pp. 49-58). IGI Global.

[42] M. Mali, Database Management System, Computer Science and Engineering-Information

Technology, Mumbai University, India: Tech-Max Publications, Pune, September 2015.

[43] Kalyanakumar P, Sangeetha A. A Synchronization Algorithm For Mobile Databases Using Samd.*International Research Journal of Engineering and Technology (IRJET)*, May 2015. 2, pp. 2395 -0056.

[44] Vu QH, Lupu M, Ooi BC. Peer-to-peer computing: Principles and applications. *Springer Science & Business Media*; 2009 Oct 20.

[45] Kekgathetse MB, Letsholo KJ. A survey on database synchronization algorithms for mobile device. *Journal of Theoretical & Applied Information Technology*. 2016 Apr 10;86(1).

[46] Nicoleta–Magdalena IC. The replication technology in e-learning systems. *Procedia-Social and Behavioral Sciences*. 2011 Jan 1;28:231-5.

[47] Bayross I. SQL, PL/SQL: The Programming Language of Oracle. *Tech Publications Pte Limited*; 2000.

[48] Kumar D, Sharma R. Data synchronization and offloading techniques for energy optimization in mobile cloud computing. *InInfocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS), 2017 International Conference* on 2017 Dec 18 (pp. 633-638). *IEEE*.

[49] Shodiq M, Wongso R, Pratama RS, Rhenardo E. Implementation of Data Synchronization with Data Marker Using Web Service Data. *Procedia Computer Science*. 2015 Jan 1;59:366-72.

[50] Choi M et al., A Database Synchronization Algorithm for Mobile Devices, *IEEE Transactions on Consumer Electronics*, May 2010,56(2).

[51] Balakumar V, Sakthidevi I. An efficient database synchronization algorithm for mobile devices based on secured message digest. InComputing, Electronics and Electrical Technologies (ICCEET), *2012 International Conference* on 2012 Mar 21 (pp. 937-942). *IEEE*.

[52] Wiesmann M, Pedone F, Schiper A, Kemme B, Alonso G. Understanding replication in databases and distributed systems. InDistributed Computing Systems, 2000. Proceedings. *20th International Conference* on 2000 (pp. 464-474). *IEEE*.

[53] Pedone F, Schiper N. Byzantine fault-tolerant deferred update replication. *Journal of the Brazilian Computer Society*. 2012 Mar 1;18(1):3-18.

[54] Khayat G, Maalouf H. Trust in real-time distributed database systems. InInformation Technology (ICIT), *2017 8th International Conference* on 2017 May 17 (pp. 572-579). *IEEE*.

[55] Silberschatz A, Korth HF, Sudarshan S. Database system concepts. New York: McGraw-Hill; 1997 Apr.

[56] Tang C, Donner A, Chaves JM, Muhammad M. Performance of database synchronization algorithms via satellite. *InAdvanced satellite multimedia systems conference (asma) and the 11th signal processing for space communications workshop (spsc)*, 2010 5th 2010 Sep 13 (pp. 455-461). *IEEE*.

[57] Elmasri R, Navathe S. Fundamentals of database systems. London: Pearson; 2016 Sep 2.

[58] Feng ZH, Qiao SU, JIAO YB, SUN JS. SQL Query Optimization on Cross Nodes for Distributed System. *DEStech Transactions on Environment, Energy and Earth Sciences*. 2016(peem).

[59] Wang D, Mani M, Rundensteiner EA, Gennert MA. Efficient Query Optimization for Distributed Join in Database Federation (*Doctoral dissertation, Worcester Polytechnic Institute*)2009.

[60] Atkinson P, Vieira R. Beginning Microsoft SQL Server 2012 Programming. John Wiley & Sons; 2012 Apr 16.

[61] Meine S. Fundamentals of SQL Server 2012 Replication. Red gate books; 2013 Aug 27.

[62] Hussain A, Khan MN. Discovering Database Replication Techniques in RDBMS. *International Journal of Database Theory and Application*. 2014;7(1):93-102.

[63] Loney K. Oracle Database 11g The Complete Reference. McGraw-Hill, Inc.; 2008 Dec 17.

[64] Deshpande K. Oracle Streams 11g data replication.2011.

[65] Oracle Corporation, Database Advanced Replication: https://docs.oracle.com/cd/B19306_01/server.102/b14226/toc.htm, Retrieved June 2018.

[66] Matthew N, Stones R. Beginning Databases with PostgreSQL. Apress; 2005.

[67] Schönig HJ. PostgreSQL Administration Essentials. Packt Publishing Ltd; 2014 Oct 15.

[68] PostgreSQL Global Development Group, Chapter 25. High Availability, Load Balancing, and Replication: https://www.postgresql.org/docs/9.2/static/high-availability.html, Retrieved June 2018.

[69] McLaughlin M. MySQL Workbench: Data Modeling & Development. McGraw Hill Professional; 2013 Apr 30.

[70] B. Ronald and S. Chris "Effective MySQL Replication Techniques in Depth", New York, United States of America: McGraw-Hill, 2013.

[71] U.R. Ramesh, G. Yogeswari and N. Tamil, "Database Synchronization for Mobile Devices by Using ASWAMD", *National Conference on Computing and Communication-International Journal of Innovative Research in Computer and Communication Engineering*, Vol.3, No.1, pp. 2320-9801, 2015.

[72] V. Balakumar and I. Sakthidevi, "An Efficient Database Synchronization Algorithm for Mobile Devices Based on Secured Message Digest", *IEEE International Conference on Computing, Electronics and Electrical Technologies [ICCEET]*, 21-22 March, Kumaracoil, India, 2012.