Mediation of Lazy Update Propagation in a Replicated Database over a Decentralized P2P Architecture

Katembo Kituta Ezechiel[1], Shri Kant[2] and Ruchi Agarwal[3]

[1] Sharda University

Received: 6 December 2018 Accepted: 4 January 2019 Published: 15 January 2019

Abstract

While replicating data over a decentralized Peer-to- Peer (P2P) network, transactions broadcasting updates arising from different peers run simultaneously so that a destination peer replica can be updated concurrently, that always causes transaction and data conflicts. Moreover, during data migration, connectivity interruption and network overload corrupt running transactions so that destination peers can experience duplicated data or improper data or missing data, hence replicas remain inconsistent. Different methodological approaches have been combined to solve these problems: the audit log technique to capture the changes made to data; the algorithmic method to design and analyse algorithms and the statistical method to analyse the performance of new algorithms and to design prediction models of the execution time based on other parameters. A Graphical User Interface software as prototype, have been designed with C , to implement these new algorithms to obtain a database synchronizer-mediator. A stream of experiments, showed that the new algorithms were effective. So, the hypothesis according to which ?The execution time of replication and reconciliation transactions totally depends on independent factors.? has been confirmed.

Index terms— peer-to-peer (P2P), database replication, data reconciliation, transaction serialization, synchronizer-mediator.

1 Introduction

n computing, a Distributed Database System (DDBS) is a database whose storage devices are not necessarily all linked to a common processing unit; but rather in this approach, the database can be stored on multiple computers, located in the same physical location or can be scattered on networked computers [1], [8]. The distribution transparency is the fundamental principle of the DDBS which consists of making a distributed system to appear similar to a centralized system to the users. The distribution transparency as well as the management of a DDBS are ensured by a program called Distributed Database Management System (DDBMS) [3]. The design of a DDBS requires that it be entirely resident on different sites of a computer network but not necessarily all. This means that at least two sites must host the database and not necessarily each site in the network, as depicted in the Fig. 1.

Thus, there are two distribution strategies: data fragmentation and data allocation on the one hand and data replication on the other hand. So, to make a good design, all these strategies are compiled [2], [3], **??**33]. The fragmentation consists in splitting a relation (a table of a database) into a number of sub-relations, called fragments; which can be horizontal, vertical or hybrid. Horizontal fragments are subsets of tuples (table records), vertical fragments are subsets of attributes (table columns), and hybrid fragmentation consists of mixing the two preceding ones. In turn the allocation is nothing more than the assignment of fragments to the sites in an optimal way [2]. When allocated fragments have to share data among them, they need the replication procedure. However, this work focuses on the data replication strategy. The replication consists of duplication and storage of multiple copies or replicas (at least two) of the same fragment or the entire relation (in the case of a fully replicated database) of a DDBS in multiple different sites. The replication is the strategy used to ensure the data exchange between fragments or relations in a fully replicated database [2], [3], [4], as illustrate in Fig. 2. In any case, the main problem of the data replication is the synchronization of replicas. Data synchronization is nothing than keeping consistent replicas in a Replicated Database System (RDBS) [5]. This means ens uring the exchange of updates between replicas. Nowadays P2P computer network is in full emergence. Comparatively to client/server model, in a P2P system, each client is itself a server. In this way replicating a Database over a P2P network require that all peers keep the same data copy. In the same way, the emergence of advanced applications of P2P systems, requiring general replication capabilities with different levels of granularity and multi-master mode [11], where each peer can transfer updates to all others and the same replica can be updated by several peers in a replicated databases environment [4], [10], the serialization of updates and the reconciliation of data turns out to be the particular P2P replication problems because those flows of updates (data) and refresh transactions conflict each other [8], **??**30], **??**33].

For example, the operations on an account, of a customer, opened in a bank with multiple branches can be replicated by several branches of the same bank and must be able to be updated by any branch anytime, to acquire reception of a transfer, for a deposit to the account, a withdrawal from the account, etc. Concretely, changes made by refresh transactions from different peers reach a destination site at the same time and multiple updates of the same replicas by different peers break the reliability and the consistency of replicas [2]. This is why this study aims to introduce an effective approach to serialize refresh transactions and to reconcile replicas in the case of inconsistency. To overcome one of DDBS homogeneity aspects, namely the same DBMS, the result of this design needs to be implemented as a synchronizer-mediator for database replication in a Graphical User Interface (GUI) using lazy decentralized sites strategy on a P2P network. To reach this purpose, the structure of this paper is organized as follow: the first section introduced by presenting the context of this research as well as the status of the problem, the second section will review the related works, the third will present the methodology, fourth section will show the simulation environment for experimentation, the fifth section will offer the result and finally the sixth section will conclude this study.

2 II.

3 Related Works

This section will rapidly review certain research works already realized to attempt to solve these two aforementioned problems.

4 a) Data replication

Designing a RDBS pursue four majeure objectives, namely : improving data availability, improving performance, ensuring scalability and users applications requirements. These purposes can be summarized as "improving consistency and/or reliability" [2], [3]. To ensure consistency between replicas, the synchronization procedure uses the transaction running technique. A transaction is a collection of operations that transforms the database from a consistent state to another consistent state [6], as illustrated in Fig. 3. A transaction has a Begin Of Transaction (BOT) and an End Of Transaction (EOT). This End is managed by three different functions: either a "commit" to validate, a "rollback" to cancel, or an "abort" to interrupt the execution of operations inside the transaction. The consistency and/or reliability of a transaction are guaranteed by 4 properties: Atomicity, Coherence, Isolation, Durability (ACID) that make the "acidity" of a transaction [2], [7]. As we are dealing with data flow, our focus remains on the Structured Query Language (SQL) operators, especially the Data Manipulation Language operators in most of DDBMSs, which contains [9]: The write operators (Insert, Update and Delete SQL commands) and the read operator (Select SQL command). Typically, like the structuring of instructions of a procedural language, a transaction "T" can have the following structure:

Begin_Of_Transaction T Insert operator Update operator Delete operator Select operator End_Of_Transaction T However, to solve the aforementioned main problem of data replication, i.e. the synchronization of replicas, there already exits four replication strategies, resulting from the combination of two factors: "when" and "where". The "when" factor specifies when updates are broadcasted (synchronously/eagerly or asynchronously/lazily), while the "where" factor indicate where updates occur on a centralized site (primary copy/mono-master) or on decentralized sites (everywhere/multi-master) before being propagated. So when we take the factor "where" in "when", it emerges [1], [2], [3], [4], **??**30], **??**33], [34]:

A. Synchronous or Eager Replication: All replicas must be updated before the transaction commit i.e. in real-time. Here, the most up to date value of an item is guaranteed to the end user. There are two different strategies in synchronous replication:

1) Eager centralized site: This method is beneficial in case where reads are much more frequent than writes. It works under the principle "Read-One, Write-All (ROWA)". After transaction commitment, any one of replicas can be read; so the write process must update all replicas. 2) Eager decentralized sites: The principle is "update everywhere"; in this logic every site is allowed to propagate updates to all sites in the same transaction, at the same time so that on the end of the transaction updates become available on all sites.

B. Asynchronous or Lazy Replication: Allows different replicas of the same object to have different values for a short periods of time i.e. in near realtime. They are updated after a predefined interval of time. There are two different strategies in asynchronous replication:

1) Lazy centralized site: It works with the principle such that one copy of replicas is assigned as the ”primary copy or mono-master” so that changes of data or writes are possible only on it. These changes are periodically propagated to the secondary copies. The secondary copies of data can only be read. 2) Lazy decentralized sites: Here the principle is so that changes can be performed ”everywhere or multi -master”, on each site. So these changes are propagated independently to other sites sporadically.

These replication strategies, have already been implemented in most of modern DDBMSs [9]. It is largely the centralized strategy that is much more wrapped in the replication models offered by almost all DBMSs. But, although these modelling are done, there remains a problem to emphasize in eager centralized site approach such that if there is a site unavailable during updates propagation by the master site, the transaction cannot commit. So, some researches are already attempting to design an optimal algorithm that can allow the update transaction commitment on the available sites and to update unavailable sites as soon as they become available again; hence the approach ”Read-One, Write-All Available (ROWA-A)” [2], [30], **??**33]. In addition, one could expect the problem related to the momentary interpolation of the line of communication between the master site and the slave sites, because it is enough for example that the master site overlord or be inaccessible so that the slaves no more access to updates [8]. Well, there is only the decentralized strategy that can clear this concern.

Nevertheless, eager decentralized sites experience the same problem as eager centralized site, whereby update transactions that arise from all sites, if they find at least one site unavailable they abort. But to overcome this problem, such kind of systems should be able first of all to commit transactions on only available sites and so update unavailable sites as soon as they become available again; hence the approach ”Update Everywhere Available” [17]. So nowadays, some researches attempt to improve these algorithms by distributed voting algorithm [4]. Thus, if the sites number quorum is reached the transaction commit on them; so afterwards, when writing, update all fraction of the replicas and when reading, read enough replicas to ensure you get at least one copy of the most recent value.

In view of the above, it seems that the lazy strategy is appropriate for P2P topology, especially since it allows replicas of various sites to diverge for a given moment. So as in a P2P network, the participants (Peers) are present or absent momentarily, updates propagation can be applicable on the present Peers while the absent Peers will remain with non -updated replicas in order to receive their updates when they become available again [10], **??**33]. Thus, lazy centralized sites approach is appropriate for the centralized P2P topology because updates are performed only on the central site and then forwarded to slave sites in near real-time while lazy decentralized sites approach is the most appropriate for the materialization of replication on a decentralized P2P topology because in near real-time, like centralized approach, updates can be performed everywhere, i.e. on each peer and then be broadcasted to all others.

Referring on our problem concerning replication over a decentralized P2P architecture, the observation has been that only a few of DDBMSs have already tried to implement the lazy decentralized strategy in order to formalize the P2P replication; let us quote for instance SQL Server [13] and Oracle [14]. Unfortunately, the particular problems of P2P replication still exist and will be developed in following lines:

? Transaction conflicts: Several updates carried by refreshing transactions, from different sites reach a destination site at the same time but they cannot be performed on the same time, then reliability and consistency will be lost and there will be the risk of transaction conflicts [2], [30],

[33], [35]. DDBMSs must ensure that transaction execution meets a set of properties that lead to the consistency of distributed databases and conveniently summarized by the ACID, since when the execution is always concurrent [6], [7]. Thus, several researches have already been undertaken to solv e the transaction concurrency control problem. Concurrent execution without harmonization constraints poses a number of problems, the most important of which is the loss of operations and incorrect readings. Therefore, it is necessary to set the serializability, a property determining a correct execution of the completion of transactions [3]. ? Data conflicts: P2P replication allows to perform changes on each peer in the topology and then forward them to other peers. However, as changes are performed at different peers, probable data conflicts are to be pointing out when modifications are being broadcasted [2],

[30], **??**33]. Thus, in all DDBMSs which have already succeed to implement the lazy decentralized sites approach to make it P2P replication, one can distinguish three types of data conflicts [13], [14],

[20], [21]: a) Primary key or uniqueness conflict: Occurs when a record with the same primary key has been created and inserted at more than one peer in the topology. So when those peers need to exchange updates, it is then impossible to violate the criterion of entity integrity; b) Foreign key conflict: Can occurs if in any case the refresh transaction forward updates which contains a record with a foreign key column but whose primary key is not yet forwarded to the destination peer. So it is then impossible to violate the criterion of referential integrity;

5 Data modifications conflicts:

? Update conflict: occurs when the same record has been updated on more than one peer; ? Insertion/Update conflict: occurs when a record has been updated on a peer and the same record has been deleted and re-inserted on another peer; ? Insert/Delete conflict: occurs when a record has been deleted on a peer and the same record has been deleted and re-inserted on another peer; ? Update/Delete conflict occurs when a record has been updated on one peer and the same record has been deleted on another peer;

? Deletion conflict: occurs when a record has been deleted on more than one peer. Thus it is necessary to think about a certain number of rules to warranty the conflict policy avoidance in the decentralized P2P replicated environment. Apart from the inconsistency of data caused by transaction conflicts and data conflicts, there are other phenomena which make the replicated data inconsistent. Thus, although the transaction that propagates the updates is successfully committed, the data remains inconsistent. Hence, there is the need of an automatic data reconciler.

6 b) Data reconciliation

Database reconciliation is a process of verifying data when there has been a migration or transfer of data from a source database to a destination. The purpose of this process is to ensure that the migration has been done accurately ??22]. In this logic, in a global manner, the data is the set of tables of a given database and in a basic way, the set of records of definite tables which can be accessed by a certain selection criterion. In a replicated Databases environment, updates broadcasting as well consists to migrate or to transfer data changes from a Prima ry site toward Secondary sites [23].

However, during data migration, errors may have occurred [12]. Most are like execution failures due to network interruptions as well as network overload those end up corrupting transactions and causing data to be lost or remain in an invalid state at the destination [8], [34]. These phenomena lead to a series of problems such as: missing records, duplicate records, incorrect values, missing values, incorrectly formatted values, broken relationships between tables in case of forced redundancy, etc.

[22]. But, some researches have already been undertaken to find solutions in several ways and some algorithms are already implemented in DDBMs and particular software to reconcile data after migration process.

Oracle Corporation [24], possesses some databases reconciliation tools for their DDBMSs: Upgrade Reconciliation Toolkit is used to compare the data on the Oracle DB source and Oracle DB destinations after data migration and after running the parallel End Of Day (EOD) activities mostly for different branches of a bank. This tool generates also the reconciliation report at the end of the process. Another tool is mysqldbcompare especially for MySQL, this tool compares two databases by identifying differences between databases objects; changed or missing rows of tables are shown in standard formats like grid, table, etc. It is going beyond the data comparison; this utility compare also objects data definition of two databases [25]. Nevertheless, all these tools run reconciliation between one source and one destination. The only one which can reconcile one source and multiple destinations is Upgrade Reconciliation Toolkit for Oracle. Unfortunately, it is only limitated to Oracle DB. The tools mysqldbcompare and MySQL_Diff are also limitated to MySQL and they are not taking in to account multiple destinations. The Tool LegiTest's should be more interesting because it is able to reconcile multi-DBM S databases, but it is also one source, one destination; and all others which have been listed in this review present such kind of limitation.

Moreover, these data reconciliation tools rely on simple counting of records to keep track if the expected number of records has been migrated. It can be esteemed that this was mainly due to the importance of the processing of essential data to carry out field validation of a given data. Nowadays, for more accuracy, the data migration algorithm should provide data reconciliation capabilities that allow the reconciliation of each data or each field, i.e. at the intersection of each row an d each column (attributes by record) of each database table [12].

To preserve data inconsistency and to maintain acidity, all instructions of the replication procedure must be wrapped in transactions [2], [7]. The instructions of a transaction are the commands or operators of the data manipulation language. But, when an operator of the data modification language is executed on a site, some time passes while waiting for the response. While a transaction may have more than one operator and the factors are likely to be varied in a P2P environment, this phenomenon should greatly influence the temporal complexity in the event of variation of different factors. So it is necessary to design a prediction model of replication and reconciliation execution time.

The assumption of this study is formulated as follows: "it seems that P2P replication systems experience the weak performance, especially since the time to replicate and to reconcile data from a Master Peer to Slave Peers dependent, if not totally, partially of certain factors, such as: the number of records in each table, the number of tables whose data has changed, the number of peers connected during the propagation of updates and other factors (number of columns per table, data types columns, etc.)".

However, these problems deserve a special attention; that is why there is a reason to wonder about setting up "a synchronizer-mediator for lazy replicated databases over a decentralized P2P architecture". This system should be able to serialize updates performed simultaneously on different replicas of the same database and to reconcile this replicas, effectively, over a decentralized P2P network.

7 III.

8 Methodology

To ensure strong replica consistency in a distributed database, traditionally the implementation of a syn chronous or eager refresh algorithm which is specially Two -Phase-Commit (2PC) based technique is the unique gateway to avoid discrepancies between replicas [2]. However, this solution is inapplicable in a P2P architecture because does not guarantee the updates delivery to all peers as they are not all always available at the same time [15]. Thus, asynchronous or lazy replication is more appropriate for P2P systems because it allows replicas to be updated independently and to remain divergent until a refresh transaction takes place [16]. Modifications which have been done to the local replica, by local transactions are captured and the refresh transaction propagates them to remote replicas asynchronously i.e. in near real-time. The technique used in this work to capture modifications is audit-log.

9 a) Audit-log technique

Almost all DDBMSs support this technique by running triggers belonging to a specific table in order to capture data modifications. A trigger is attached to an event produced by an Insert or Update or Delete operator so that it captures changes before or after the event has taken place in the database [5], **??**33]. So, in this work the interest is carried on after trigger. To achieve this, for each data-table the creation of one ( ) C audit-table is necessary. The audit-table is composed by the data-table primary key column, other data-table columns (apart from the primary key), the updated column name, the audit action, the timestamp and the synchronization ID. These elements are required for a record to do the comparison between data. Each table in the database would need three triggers to run after Insert, after Update and after Delete. The flow chart, Fig. 4 here below illustrates the audit-log creation. Suppose that the database is homogenous and full replicated, as soon as the audit log creation of each data table completed, on each peer, for each SQL data modification operation, the DDBMS performs following action accordingly:

? After each Insert operation in the data table, the "insert trigger" captures the newly added record and inserts it in the audit table, as shown in Fig. The column synchronisation ID (Sync_ID) in Audit -tables don't have same value; for a Master Peer Audit-table its content is "Local-Transaction", value automatically provided by the trigger procedure when the transaction is initiated locally by the user application whereas for a remote transaction the synchronization procedure update automatically this column by the sync. ID provided by the Sync. Mediator-System. So, the synchronization procedure select only data whose Sync_ID is equal to "Local-Transaction" and whose Audit_Timestamp is in the interval of begging date and time to ending date and time and apply them to Slave Peers according to the Audit_Action value. This technique permits us to resolve the problem of the endless loop in the sync. procedure used two -ways or symmetrical replication which was knowing old synchronizers [5].

10 b) Algorithmic method

The Algorithmic method will be used to design and to analyse instructions of algorithms and steps of a Peer-to-Peer Synchronizer. This method will take in account the Circulatin g Token Ring Algorithm, the Decentralized Peer-to-Peer Replication Algorithm and the Decentralized Peer-to-Peer Data Reconciliation Algorithm.

11 i. Network Topology and Algorithm

When a peer needs to broadcast its captured updates toward other peers, it needs a token which gives it the state of a Master i.e. the permission to forward its updates and other peers become automatically Slaves. A fully replicated P2P database system includes p peers and each peer has a complete copy of the database. Peers communicate with each other by exchanging messages and forwarding updates or accessing peer data by performing transactions [17]. In this way, updates will be applied according to a circulating token, as depicted in Fig. 6, which determine transactions serialization order or one can give the privilege to updates from certain sites considered to be mo re important or privileged.

Suppose a network consisting of four peers A, B, C and D all networked. The Fig. 6 below presents the decentralized topology of peer-to-peer token ring network. A predefined order of releasing or getting the token, since we are in a P2P network where a peer p may or may not be available, is not needed. The optimization policy here is to give the token directly to a peer which needs it instead of going through a list of peers that we are not sure of their availability at the time of the token release. So, transaction serialization is managed by the new circulating token algorithms 1 and 2, successively for getting the token and releasing the token. Since when a peer (p), which can be "A" or "B" or "C" or "D" gets the token, it executes the transactions according to the algorithm 3, 4 and 5 for data replication and 6 for data reconciliation. Consequently, all transactions performed are accepted and none rejection because only a peer which possess the token can perform a transaction of its updates broadcasting and reconcile other peers' data with its updates. As soon as peer "A" finishes to perform updates and reconciliations with peers "B, C and D", it releases the token and other peers like "B" or "C" or "D" can randomly take it, but according to the token request minimum date and time, and do the same, unless a privileged peer requests it.

12 ii. Replication Protocol and Algorithm

Assuming that the database is homogenous, full replicated and each Peer work under a Two-Phase-Locking (2PL) concurrency control technique. The model of the lazy replication over a decentralized Peer-to-Peer Architecture is presented as follows: let W(x) be a write transaction where x is a replicated data item at Peers A, B, C and

D. The Fig. 7, here below depicts how transactions update different copies at all Peers and after commit the refresh transaction, wrapped in the Sync. Mediator-System, forward updates to all peers.

13 iii. Reconciliation Protocol and Algorithm

After a large data transmission, to overcome the problem of data inconsistency due to untimely interruptions of connectivity, network overload and other technical hazards, updates forwarded to each peer in the replication procedure must be reconciled.

The model of the Decentralized Peer-to-Peer Data Reconciliation is presented as follows: let R(x) be a read transaction where x is a replicated data item at Peers A, B, C and D. The Fig. 8, here below depicts how reconciliation is performed on different copies of all peers. After the implementation of these algorithms presented above, the main goal, according to which setting up a synchronizer-mediator for database replication being able to serialize the propagation of updates and their reconciliation in a replicated databases system over a decentralized P2P network is achieved. Although this goal be achieved, it is appropriate to know here that in computing the performance of an algorithm is assessed on the basis of its complexity [18]. The analysis of the theoretical complexity of this algorithm will be more concerned the time complexity than the space complexity especially as the data will be momentarily transit through the buffer to the destination. Nevertheless, the practical time that the execution of this algorithm takes will result from the simulation and will be calculated by the statistical method.

14 c) Statistical method

The performance of a system depends on a certain number of factors. We have to determine the practical time, that makes our system to execute successively transactions of updates propagation or replication (insert, update and delete) and transactions of data reconciliation. To analyse this performance, we will use the linear regression test with the random sampling technique. The linear regression test is a statistical analysis method that describes the variations of an endogenous variable associated with the variations of one or more exogenous variables i.e. the relation between an endogenous variable and one or more exogenous variables. In the case where the study concerns an endogenous variable with one exogenous variable, it's a simple regress ion and when it's an endogenous variable with more than one exogenous variable, it is a multiple regression [19]. This test will be used not only to determine the execution time based on a certain sample, but also to make a linear regression model that will be used to predict the execution time , which is the dependant factor or endogenous variable, based on other independent factors or exogenous variables, namely the number of records, the number of tables in the database and the number of Slave Peers. The following variables are selected:

? Y i : is a random variable to explain "the time the synchronization algorithm takes to broadcast updates and to reconcile replicas for an execution i"; ? X i1 : is an explanatory variable "the number of records the synchronization algorithm broadcast from a Master Peer to Slaves and reconcile between the Master and Slaves for an execution i"; ? X i2 : is an explanatory variable "the number of tables in the database whose records knew updates which need to be broadcasted and reconciled with Slaves for an execution i" ? X i3 : is an explanatory variable "the number of Slave Peers available to receive updates and to be reconciled for an execution i". Given a sample (Y i , X i1 , X i2 , X i3) whose i ? [1, n], we will try to explain, as precisely as possible, the values taken by Y i , the so-called endogenous variable from a series of explanatory variables X i1 , X i2 , X i3 . The model formulated in terms of random variables, takes the form: ?? ?? = ?? 0 + ?? 1 ?? ??1 + ?? 2 ?? ??2 + ?? 3 ?? ??3 + ?? ?? Where:

? i = 1, 2, . . . , n ? b 0 is the constant term;

? b 1 , b 2 and b 3 are coefficients of the regression to be estimated; ? ? i : is the model error that expresses or summarizes the missing information in the linear explanation of the values of Y i from X i1 , X i2 , X i3 (a random variable of zero mathematical expectation in this model i.e. problem of specifications, variables not taken into account, etc.). The intensity of the relation between the independent variables and the dependent variable will be expressed by the correlation coefficient "R", which is the square root of the "R^2", the determination coefficient of a linear regression model. The coefficient of correlation, will be used to determine the degree of linkage between the independent variables and the dependent variable while the coefficient of determination will help to measure the proportion of dependence of the dependent variable explained by independent variables. Thus, two sets of hypothesises are evoked as follow: These hypotheses will be verified at the end of the results which will be produced by a series of experiments perpetrated on a simulation environment which will be described in the following section.

IV.

15 Simulation Environment

The implementation and experimentations will be run on a P2P network consisting of 4 traditional computers depicted in the Fig. 9, with the following properties: Processor: Intel Core i5, CPU 2.40GHz, Memory (RAM): 8.00GB and Storage: 1TB. The network will be based on a desktop switch of 100 Mbps of transmission speed, to establish a simple LAN using twisted -pair cables connection and RJ45 connectors. These computers will run under Windows 10 Professional 64 bits and SQL Server Management Studio 2012 Express as DDBMS, to manage databases and establish the connectivity between them. According to this Fig. 9 above, a node is composed by hardware and software as required previously. But in this same figure one can point out the presence of a ”Mediator” for each peer. The mediator is nothing else than the synchronization system, ”Sync. Mediator-System”, a C# software which has been designed and in which it has been implemente d algorithms, already described in the methodology, to lead to a windows application running under a graphical user interface, as presented in the Multiple-Document Interface (MDI) window here below in the Fig. 10. Thus this mediator must be installed on each node to manage the replication transactions and the reconciliation of replicas. For the execution to be effective, there are prerequisites to fulfil.

16 a) Prerequisites

When designing the global schema of the database, each table must have:

? The name such as ”Data_tbTableName” and the first column as its primary-key to identify data and to make the difference between records. The creation of primary keys by automatic incremental systemprovided by the DBMSs is disadvised, it is preferable to program an automatic primary key combined with the site number to avoid redundancy;

? Bear in mind that the database is homogeneous i.e. the data structure of the replicated database must be uniform on all peers. Before the actual processing phase begins, under expected replication, ”Sync. Mediator-System” provides two procedures that must be performed automatically in advance for each table, as showed in the window, Fig. 11: ? To create one audit table named ”Audit_tbTableName”, to store changes captured by 3 triggers belonging to each table. Each audit table must have its next four last columns to store respectively the updated column name, the audit action, the audit timestamp and the last column to store the synchronization ID;

? To create three triggers to run after Insert, after Update and after Delete, to capture data changes and store them in the specific audit table.

17 The new circulating token algorithm has two phases: i. Data replication

Update transaction serialization: All update transactions must be executed in serial order. Before initiating a refresh transaction, each peer must first receive a single token of a sequential series, to get the order in which the transaction will be executed. Once a token has been assigned to a peer p, this last becomes directly a Master so it performs update transactions to all connected Slave peers, as showed in the window, Fig. 12. Update transaction performing: When a Slave peer receives an executing transaction, it places it according to its Master peer’s token as well as its number (Sync_ID, in Fig. 5) and updates are performed to the Slave peer database. As soon asthe transaction ends on each Slave peer, it sends an appropriate message to the Master peer to certify the transaction commitment. The peers connected during the initiation of the transaction and whose transaction has been aborting during transaction performing, due to any kind of issue to the site which host the peer, must be mentioned in the pending list in order to be updated later in a new procedure reusing the same Sync_ID. Then the main transaction, initiated on the Master peer, ends when it has been executed on all peers and give immediately the relay to the reconciliation procedure.

ii. Data reconciliation Reconciliation transaction serialization: Reconciliation in turn will benefit from the serial order of their ”Mather” update transactions. This phase must begin on the Master peer once the replication is complete. The reconciliation procedure must also initiate transactions to read updates received by Slave peers. These readings consist of a comparison between the data s ent by the Master peer and the data received by the Slave peers. The comparison operation is performed according to data carrying the token of the same Master initiator of the replication transactions, as revealed in the window, Fig. 12. All errors like missing records, duplicate records, incorrect values, missing values, incorrectly formatted values are retained in order to be fixed.

Reconciliation transaction execution: This phase consists of fixing all retained errors so that missing records are inserted, duplicate records are deleted, missing values are added to their respective fields, incorrectly formatted values are replaced by correct values. Data reconciliation process can be however restarted if the first one done didn’t put replicas in consistent state. So procedure can be repeated until all replicas become consistent, then the Master peer can release the token. In the case where the inconsistency persists among data, probably it can be caused by conflicts.

18 c) Conflicts avoidance rules

To avoid potential conflicts among data in the P2P replicated database environment, some rules must be respected:

? When using the database, it is inadvisable not to update the value of the primary key; instead, it is better to delete the entire record and re-insert it; ? When designing an application which communicate with the database, create procedures which cannot allow from a peer to update or to delete a record whose insertion was not performed on that same peer i.e. the modification of a data must be done only and After the configuration be performed as indicated in this section to simulate the replication process on a P2P network, the test and/or experiment sets yielded the results which are presented in the next section.

V.

19 Result

This section is dedicated to testing this new synchronizer of databases, results and evaluating the performance of the newly proposed algorithm. To achieve this, it is necessary to analyse the performance in order to jus tify the effectiveness of the algorithm.

20 a) Performance analysis

Suppose that this algorithm has to broadcast updates emerging from the replicated database over 4 peers A, B, C, and D, local servers of a bank branches. Being fully replicated and homogeneous, the physical schema of this database consists of 3 tables, as presented in Fig. 13. So, for all cases, consider the sample of 12 executions, to operate randomly and based on the reality of the replicated data manipulation in the distributed environment of banking database. However, in all cases, insertions are greater than or equal to updates and deletes. But updates can be more or less than deletions.

After the replication transaction has completed, if there has been an overload or interruption of the network corrupting the replication transaction, then assume that the data that the destination peers have received has experienced some inconsistencies with respect to those of the master peer. Fro m the total replicated data (inserts, updates, and deletes), consider that 25% are missing records that require re-insertion, incorrect values, missing values, and incorrectly formatted values which need to be updated and duplicate records that require deletion, as typically data to be reconciled does not exceed ¼ of that of replication [2], **??**22]. Thus, it resorts the data presented in the table 1 hereafter: For analysing the effectiveness of our algorithm, the experimentation will be realized in four scenarios, namely:

1. Experimentation based one table stored on a master peer with two slave peers ; 2. Experimentation based two tables stored on a master peer with two slave peers ; 3. Experimentation based one table stored on a master peer with three slave peers; 4. Experimentation based two tables stored on a master peer with three slave peers.

To carry out the analysis of the performance, based on the prediction of the execution time according to the data of the sample presented in the Table 1 above, it results the execution times obtained after experimentation and presented successively in the tables and charts below: All basic factors remaining unchanged i.e. one table stored on a master peer with two slave peers, replication and reconciliation models are successively presented as follow : insert operator, Fig. 14(a By varying the factor number of tables, from one to two tables stored on a master peer, dividing the number of records equitably between two tables and maintaining unchanged the factor number of slave peers in "two (2) peers", the replication and the reconciliation models are successively given as follow: Year When we increase the number of tables from one to two, in 1 second, the prediction of the execution time (y), during which this algorithm can successively replicate and reconcile the number of records (x), is calculated from the following way:

? For insert operator ? In replication procedure (Fig. 16(a)) : 1 = 0.021 ?? ? 1.3366 ??0.021 ?? =?1.3366 ? ?? =111.26 ? ?? ? 111 inserted records to be replicate in 1 second. So, as the coefficient of determination R^2 = 0.9846 then the dependence degree of insertion execution time compared to the number of records is 98.46% and as the coefficient of correlation R= ??? 2 ? R = ?0.9846 ? R = 0.9923 then the degree of linking between the insertion execution time and the number of records is 99.23%.

? In reconciliation procedure (Fig. 17 The experimentation of this algorithm on a topology consisting of two (2) slave peers proves that the variation of the number of tables containing data to replicate and reconcile in a P2P replication system has a significant impact o nly for the replication transaction as illustrated in Fig. 18. For all data modification operators , illustrated by graphs of Fig. 18(a), Fig. 18(b) and Fig. 18(c), successively, taken into account in the replication process, the execution time, when record s originate from one (1) table, is greater than the execution time when the same number of records emerge from two (2) different tables while for reconciliation the impact is not too great.

Hence this variation has no significant effect on the execution time of data reconciliation because the number of records to reconcile from one (1) table and average of execution time, calculated in Table 2, are not far different from those to reconcile from two (2) tables and whose average of execution time is calculated in Table 3. This is why the curves of the graphs depicted in Fig. 18(d So, partially we can conclude that this algorithm is efficient for the replication of databases because generally a database does not have one table i.e. data to replicate are scattered in several tables. As for reconciliation, since it takes place only when it is necessary and mostly data to be reconciled do not exceed one quarter of that of replication, little importance should be attached to the computational time of this phenomenon. Year This conclusion was obtained after varying the factor number of tables. However, by keeping unchanged all other factors, except the number of slave peers that vary from two (2) to three (3) peers, using the same sample in Keeping the factor number of table unchanged, one table stored on a master peer with three slave peers, the replication and reconciliation models are successively presented as follow: insert operator, Fig. 19 In 1 second (y) we predict that this algorithm can successively replicate and reconcile following number of records (x):

? For insert operator ? In replication procedure (Fig. 19 Varying the factor number of table stored on a master peer with three slave peers, the replication and reconciliation models are successively presented as follow: insert operator, Fig. 21 After increasing the number of tables from one to two, in 1 second, the prediction of the execution time (y), during which this algorithm can successively replicate and reconcile the number of records (x), is established as follows:

? For insert operator ? In replication procedure (Fig. 21 When running this algorithm on a topology consisting of three (3) slave peers, the experimentation result proves that the variation in the number of tables containing data to replicate and to reconcile in a P2P replication systemhas a significant impact on the execution time of replication and reconciliation transactions, as shown in Fig. 23. Fig. 23: Effectiveness of replication and reconciliation based one table stored on a master peer with three slave peers vs two tables stored on a master peer with three slavepeers.

However, this impact is explained only by the comparison of averages, in Table 4 and 5 Mediation of Lazy Update Propagation in a Replicated Database over a Decentralized P2P Architecture execution time with one table. But, in terms of predictive models, we found that, when the records come from one table, the execution time is greater than the execution time when the same number of records is split and comes from two different tables. This phenomenon is clarified by the successive resolution of the prediction equations of the replication and reconciliation models which proved that the number of records to replicate and reconcile to 1 second, with two tables of origin is greater than those when there is only one table.

Thus, partially we can conclude that this algorithm is effective for the replication of databases, its performance increases with the increase of the tables for a certain number of records. So, since the data to replicate is usually scattered across multiple tables, we can count on its effectiveness. Fig. 24: Effectiveness of replication and reconciliation based one table stored on a master peer with two slave peers vs one table sto red on a master peer with three slave peers.

The result we have achieved so far comes from the analysis of performance by varying the numbers of tables in which the data to be replicated and reconciled originate. Nevertheless, later on, we have to analyse the performance of this algorith m starting from the variation of the slave peers. Thus, Fig. 24 and Fig. 25, show the effectiveness result when increasing the number of slave peers but the data to replicate and reconcile successively from a single table and two table. After increasing the number of slave peers, the execution time of the replication transaction as well as the reconciliation of the data, successively from a table, as illustrated in Fig. 24 and two tables, as shown in Fig. 25, knows a significant increase. This increase in execution time affects negatively the ? Secondly by comparing the predicted values, in this case the prediction of the number of records to replicate and reconcile to 1 second. After the successive resolution of the prediction models equations for replication and data reconciliation, we found that the number of records to replicate and reconcile are declining after increasing a slave peer. However, based on these observations from all the cases i.e. with the data to be replicated and reconciled from one or two tables, we can partially conclude that the increase of the number of slave peers on a Replicated Databases over a Decentralize d P2P topology is causing the loss of performance of the synchronization algorithm.

21 b) Result summary

In view of what we have just achieved as a result, it is necessary to summarize and give a general conclusion. Thus, the Table 6 here below will first give a summary of the results. Starting from the results presented above and summarizin g in Table 6, our first group of hypotheses of the significance test of each independent variable gives the conclusion that each independent variable is a significant predictor of the dependent variable. In other words, the number of records in each table (xi1), the number of tables whose data has chan ged (xi2), the number of peers connected during the propagation of updates (xi3) and other factors (?) like number of columns per table, data types columns, etc., each taken separately predict significantly the execution time (y) of the replication transaction as well as that of reconciliation because almost all coefficient of determination (R^2) are greater than or equal to the confidence level of 95%. In all the cases the execution time depend on other factors beyond 95% and these factors correlate positiv ely and tightly of the totality. This means that the changes made to one of these independent variables affect in 95% or more of the dependant variable and vice versa. Hence, we accept the alternative hypothesis (H1) and thus reject the null hypothesis (H0). As for the second group of hypotheses, since for all experimental scenarios all independent variables (the number of records in each table (xi1), the number of tables whose data has changed (xi2), the number of peers connected during the propagation of updates (xi3) and other factors (?) like number of columns per table, data types columns, etc.,) are significant predictors of the dependent variable which is the replication and reconciliation transaction execution time (y), the overall model of the regression is significant, at the same thresholds significance derived from the combination of factors by the experimental scenarios summarized in the Table 6 above.

The experimental results show that our algorithms are performant since when to 1 second, a time elementary unity, it can replicate and reconcile a considerable number of records, like present the last column in the Table 6, for the present experimental environment. However, since the performance of a computer algorithm is due to its execution time, this is how we assert our main hypothesis that P2P replicated databases systems experience the weak performance, especially since the time of transmission of updates from a Master Peer toward Slave Peers dependent in more than 95% of the number of records, the number of tables whose data know changes, the number of peers connected during the propagation of updates and other factors.

Nevertheless, as we have just seen, when we take two by two experimental scenarios those can be noted successively I: 1 and 2, II: 3 and 4, III: 1 and 3 and finally IV: 2 and 4 of Table 6 above, I made a good performance, II also made a performance gain but not far from the average, III made a loss of performance and IV made a loss as well. Taking III and IV it emerges the variation of number of peers connected whereas from I and II emerge the variation of the tables. During the experiment, it was found that the variation of number of the tables did not lose the performance, contrariwise it improved it. Moreover, among the independent variables, the number of records and the number of tables being factors directly related to the database before even hinting at the data replication, it is clear that it is the growth of number of connected peers which is at the base of the considerable loss of the performance i.e. the increase of the execution time of a synchronization algorithm of distributed databases.

Thus, as a future work to be carried out, as part of improving the performance of this proposed algorithm, the thought will revolve around synchronization algorithm for replicated databases over a decentralized P2P architecture with supernodes or super-peers [31], [32] belonging to peers clusters in order to reduce execution time of transactions and to reach load balancing during data transmission [35].

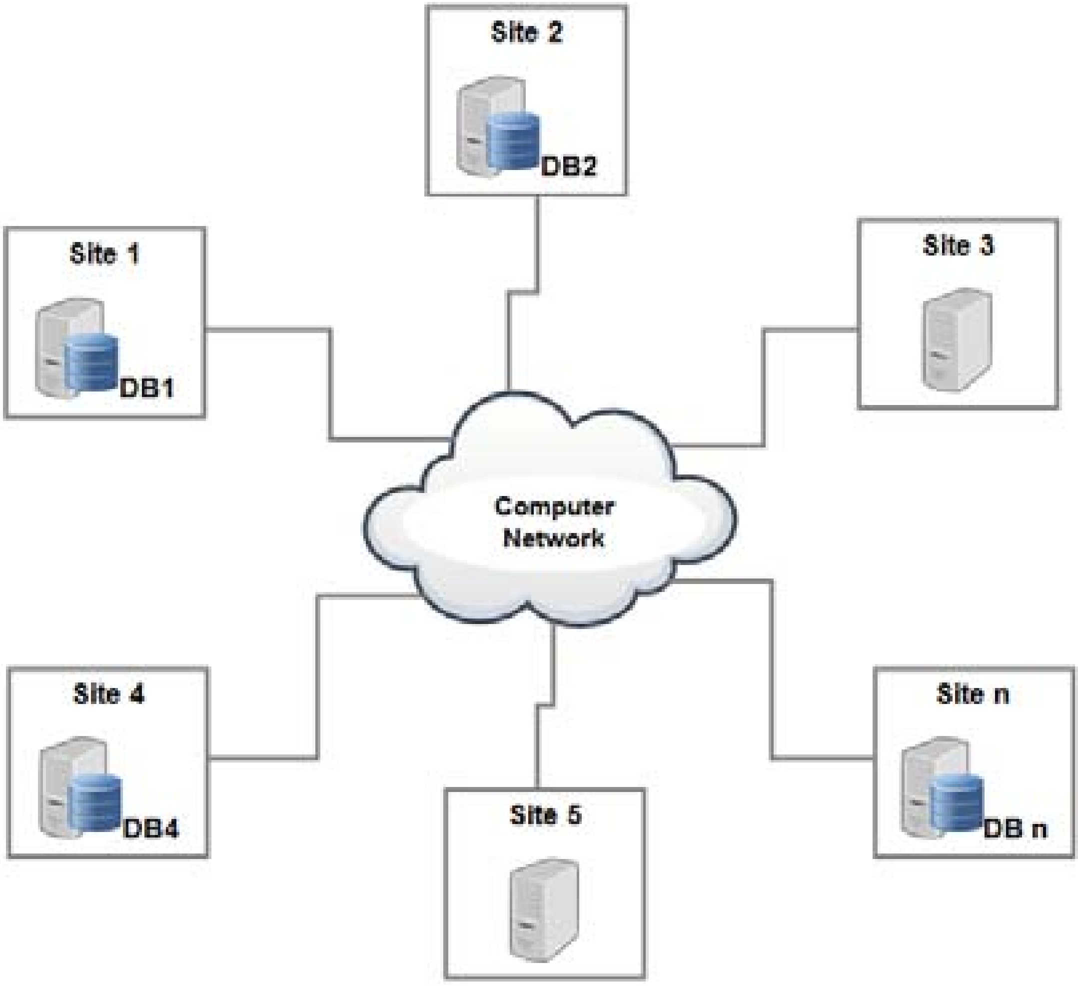
22 VI.

23 Conclusion

This article proposes a prototype of a synchronizer-mediator for lazy replicated databases over a decentralized P2P architecture in a Graphical User Interface. The motivation arises from the common problem of databases replication consisting to maintain consistent replicated databases over a decentralized P2P network.

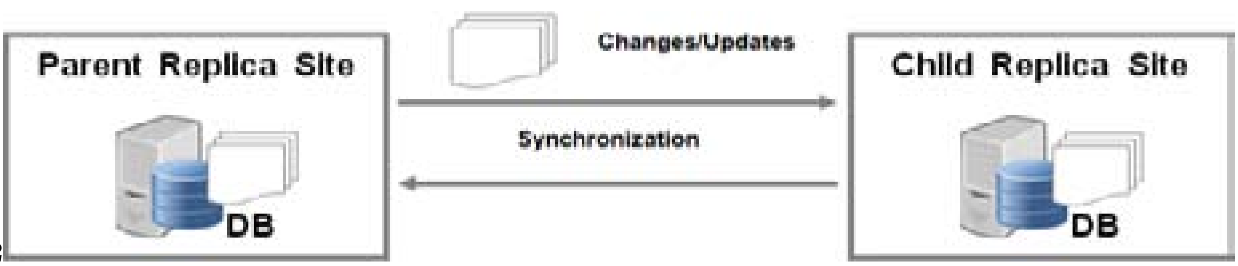
However, two specific problems caught our attention: transactions broadcasting updates from different peers are performed concurrently on a destination peer replica, which always causes transactions conflicts and data conflicts. Moreover, during data migration, connectivity interruptions and network overload corrupt transactions so that destination peer databases can contract duplicated records, unsuitable data o r missing records which make replicas inconsistent. Differen t methodologies have been used to solve these problems : the audit log technique to capture and store data changes in audit tables; the algorithmic method to design and analyse algorithms for transactions serialization, for data replication transactions and the replicas reconciliation transactions end finally the statistical method to analyse the performance of algorithms and to produce prediction models of the execution time.

The C # prototype software has been designed to implement algorithms and permit to execute the test in order to make out the effectiveness of each experimental scenarios. Afterwards it has been shown that the algorithm has a good performance because it can replicate and reconcile a considerable number of records to 1 second. Finally, the assumption according to which "The execution time of replication and reconciliation transactions totally depends on independent factors" has been affirmed. [1] [2]



1

Figure 1: Fig. 1 :



2

Figure 2: Fig. 2 :

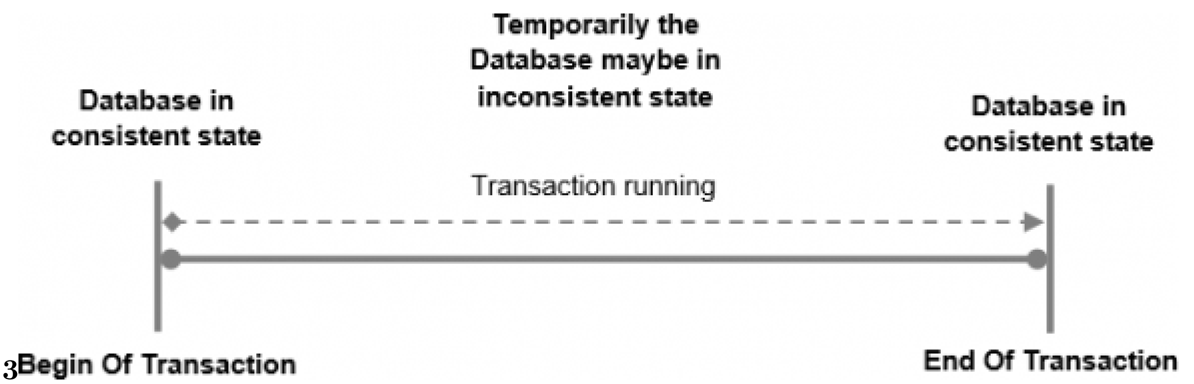
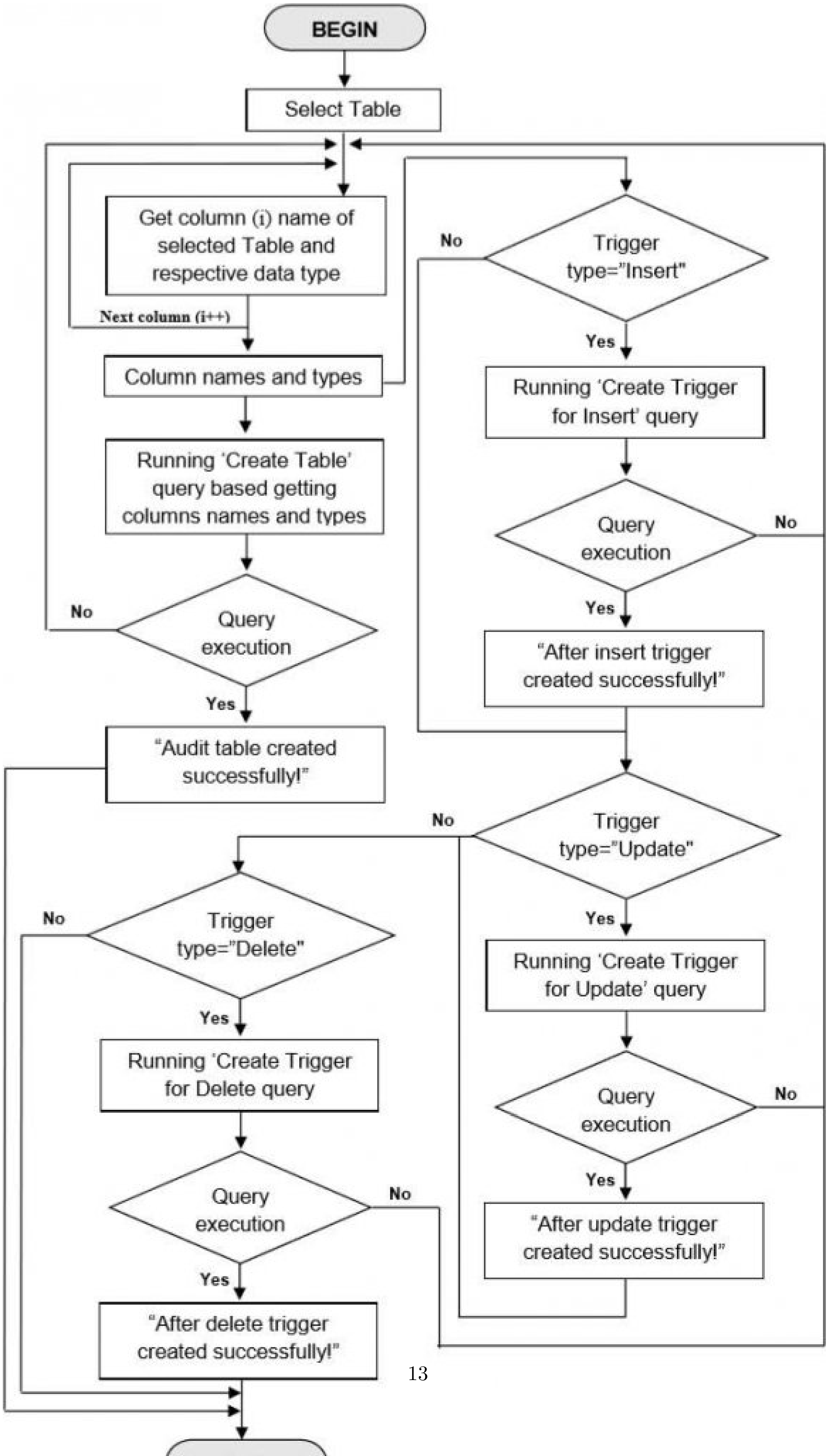
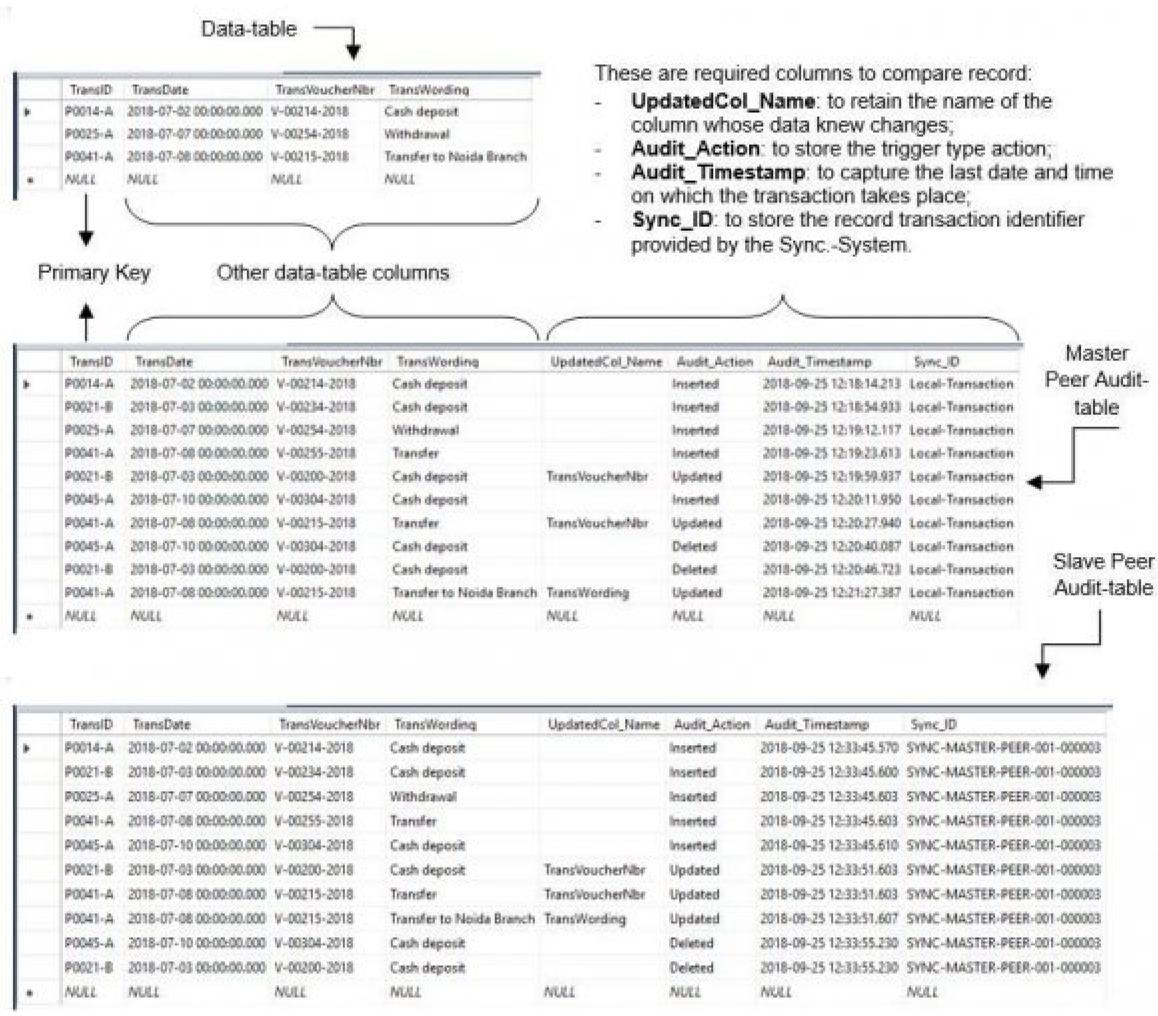


Figure 3: Fig. 3 :


BEGIN
Select Table
Get column (i) name of selected Table and respective data type
Next column (i++)
Column names and types
Running 'Create Table' query based getting columns names and types
No
Query execution
Yes
"Audit table created successfully!"
No
Trigger type="Insert"
Yes
Running 'Create Trigger for Insert' query
Query execution
No
Yes
"After insert trigger created successfully!"
No
Trigger type="Update"
Yes
Running 'Create Trigger for Update' query
Query execution
No
Yes
"After update trigger created successfully!"
No
Trigger type="Delete"
Yes
Running 'Create Trigger for Delete query
Query execution
No
Yes
"After delete trigger created successfully!"



TransID	TransDate	TransVoucherNbr	TransWording
P0014-A	2018-07-02 00:00:00.000	V-00214-2018	Cash deposit
P0025-A	2018-07-07 00:00:00.000	V-00254-2018	Withdrawal
P0041-A	2018-07-08 00:00:00.000	V-00215-2018	Transfer to Noida Branch
NULL	NULL	NULL	NULL

TransID	TransDate	TransVoucherNbr	TransWording	UpdatedCol_Name	Audit_Action	Audit_Timestamp	Sync_ID
P0014-A	2018-07-02 00:00:00.000	V-00214-2018	Cash deposit		Inserted	2018-09-25 12:18:14.213	Local-Transaction
P0021-B	2018-07-03 00:00:00.000	V-00234-2018	Cash deposit		Inserted	2018-09-25 12:18:54.933	Local-Transaction
P0025-A	2018-07-07 00:00:00.000	V-00254-2018	Withdrawal		Inserted	2018-09-25 12:19:12.117	Local-Transaction
P0041-A	2018-07-08 00:00:00.000	V-00255-2018	Transfer		Inserted	2018-09-25 12:19:23.613	Local-Transaction
P0021-B	2018-07-03 00:00:00.000	V-00200-2018	Cash deposit	TransVoucherNbr	Updated	2018-09-25 12:19:59.937	Local-Transaction
P0045-A	2018-07-10 00:00:00.000	V-00304-2018	Cash deposit		Inserted	2018-09-25 12:20:11.950	Local-Transaction
P0041-A	2018-07-08 00:00:00.000	V-00215-2018	Transfer	TransVoucherNbr	Updated	2018-09-25 12:20:27.940	Local-Transaction
P0045-A	2018-07-10 00:00:00.000	V-00304-2018	Cash deposit		Deleted	2018-09-25 12:20:40.087	Local-Transaction
P0021-B	2018-07-03 00:00:00.000	V-00200-2018	Cash deposit		Deleted	2018-09-25 12:20:46.723	Local-Transaction
P0041-A	2018-07-08 00:00:00.000	V-00215-2018	Transfer to Noida Branch	TransWording	Updated	2018-09-25 12:21:27.387	Local-Transaction
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

TransID	TransDate	TransVoucherNbr	TransWording	UpdatedCol_Name	Audit_Action	Audit_Timestamp	Sync_ID
P0014-A	2018-07-02 00:00:00.000	V-00214-2018	Cash deposit		Inserted	2018-09-25 12:33:45.570	SYNC-MASTER-PEER-001-000003
P0021-B	2018-07-03 00:00:00.000	V-00234-2018	Cash deposit		Inserted	2018-09-25 12:33:45.600	SYNC-MASTER-PEER-001-000003
P0025-A	2018-07-07 00:00:00.000	V-00254-2018	Withdrawal		Inserted	2018-09-25 12:33:45.603	SYNC-MASTER-PEER-001-000003
P0041-A	2018-07-08 00:00:00.000	V-00255-2018	Transfer		Inserted	2018-09-25 12:33:45.603	SYNC-MASTER-PEER-001-000003
P0045-A	2018-07-10 00:00:00.000	V-00304-2018	Cash deposit		Inserted	2018-09-25 12:33:45.610	SYNC-MASTER-PEER-001-000003
P0021-B	2018-07-03 00:00:00.000	V-00200-2018	Cash deposit	TransVoucherNbr	Updated	2018-09-25 12:33:51.603	SYNC-MASTER-PEER-001-000003
P0041-A	2018-07-08 00:00:00.000	V-00215-2018	Transfer	TransVoucherNbr	Updated	2018-09-25 12:33:51.603	SYNC-MASTER-PEER-001-000003
P0041-A	2018-07-08 00:00:00.000	V-00215-2018	Transfer to Noida Branch	TransWording	Updated	2018-09-25 12:33:51.607	SYNC-MASTER-PEER-001-000003
P0045-A	2018-07-10 00:00:00.000	V-00304-2018	Cash deposit		Deleted	2018-09-25 12:33:55.230	SYNC-MASTER-PEER-001-000003
P0021-B	2018-07-03 00:00:00.000	V-00200-2018	Cash deposit		Deleted	2018-09-25 12:33:55.230	SYNC-MASTER-PEER-001-000003
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

5

Figure 5: Fig. 5 :

6

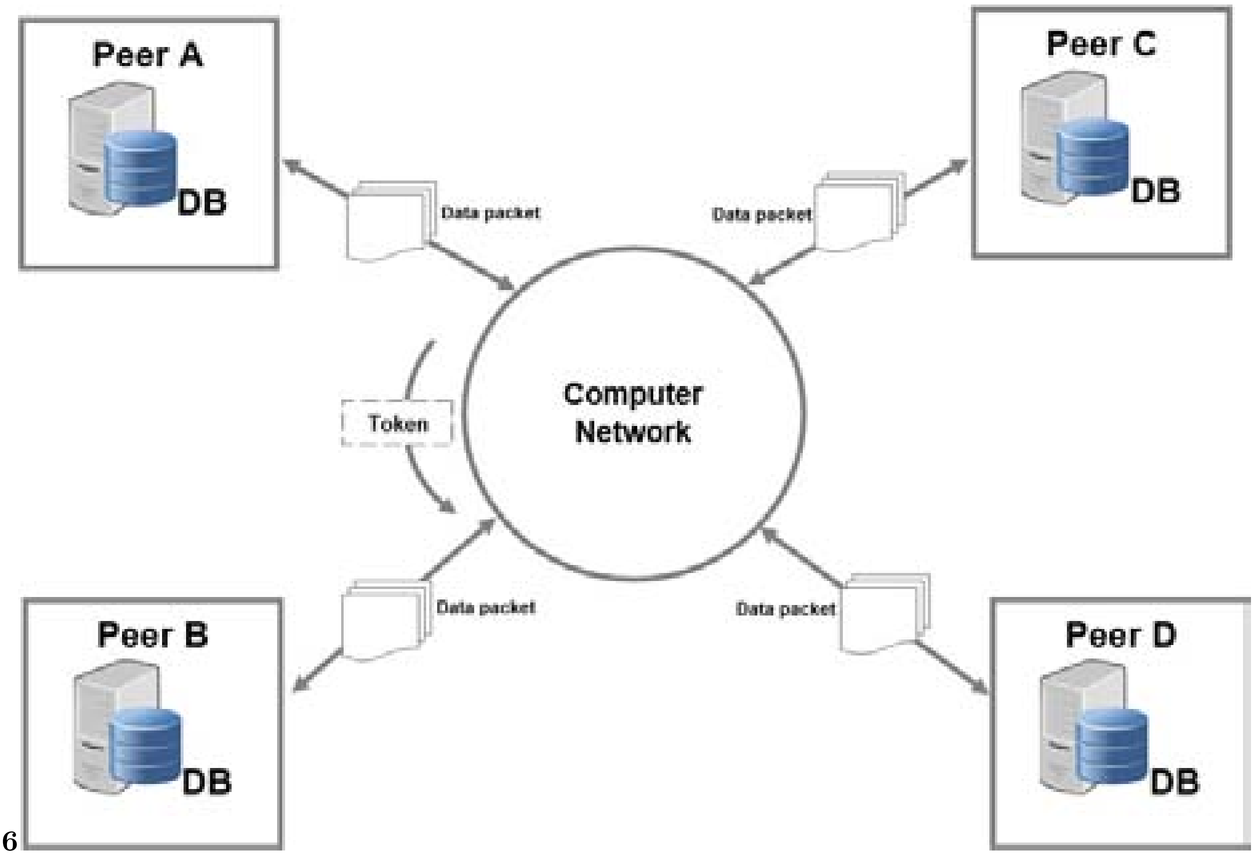
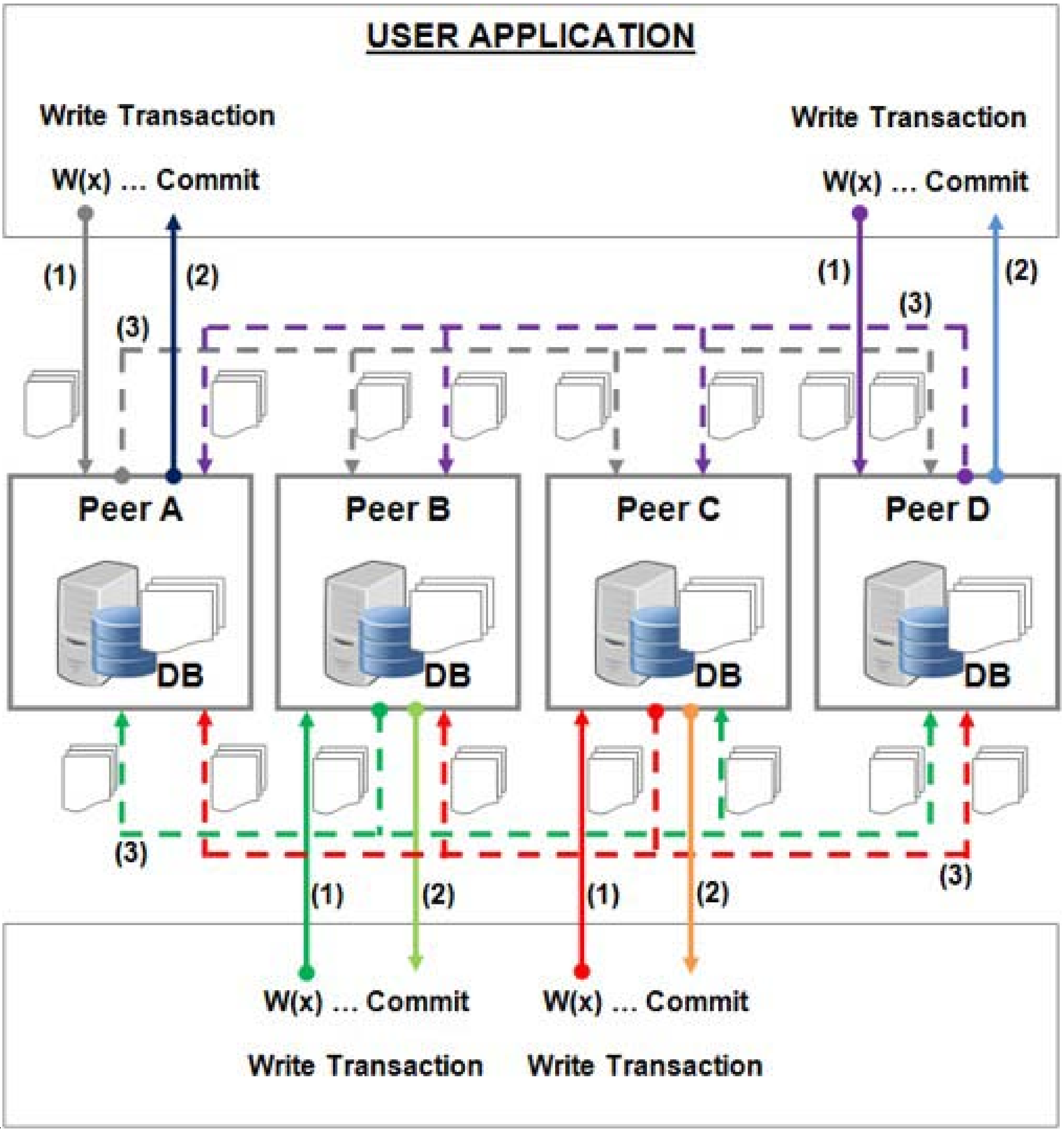
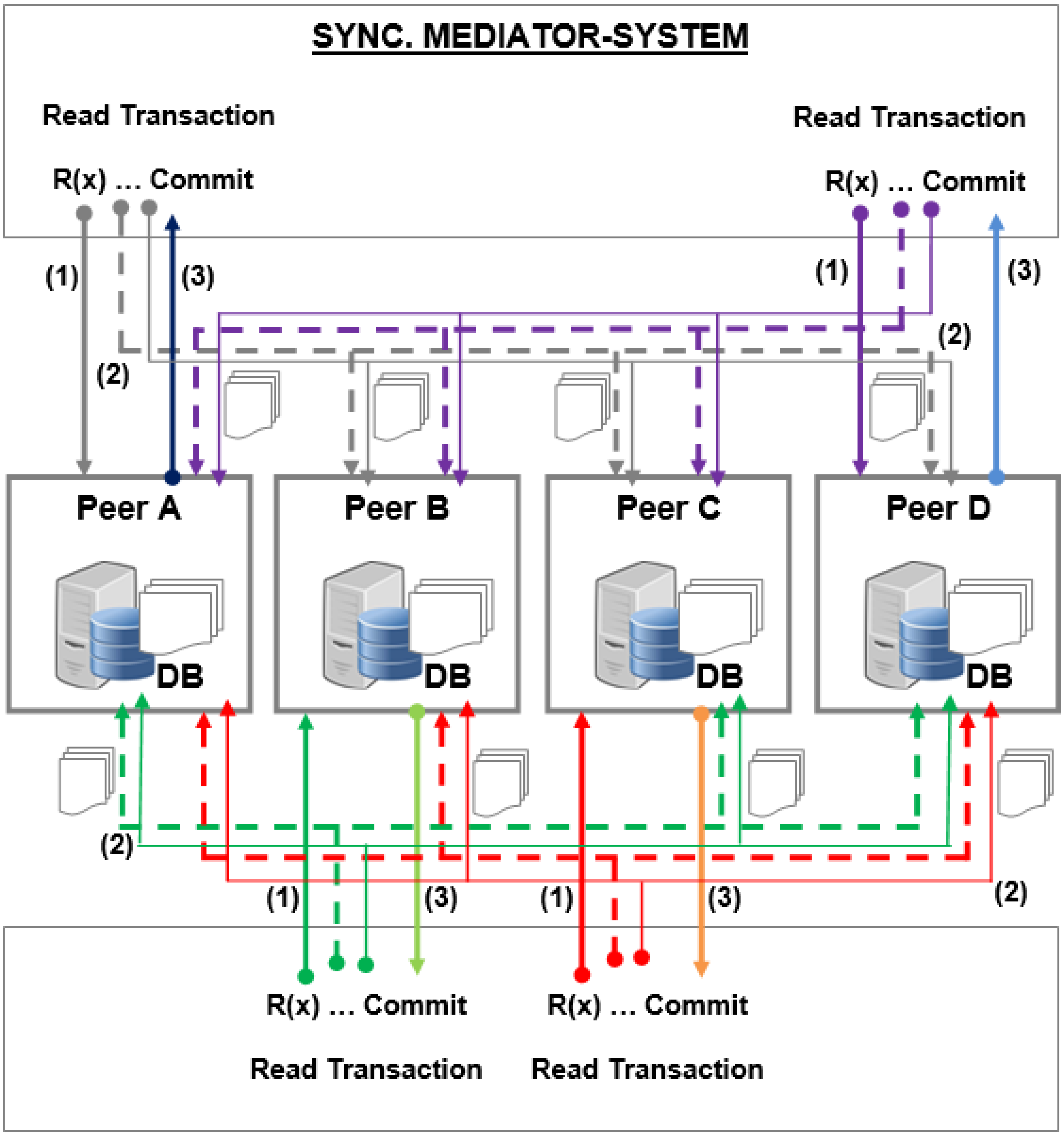


Figure 6: Fig. 6 :



1

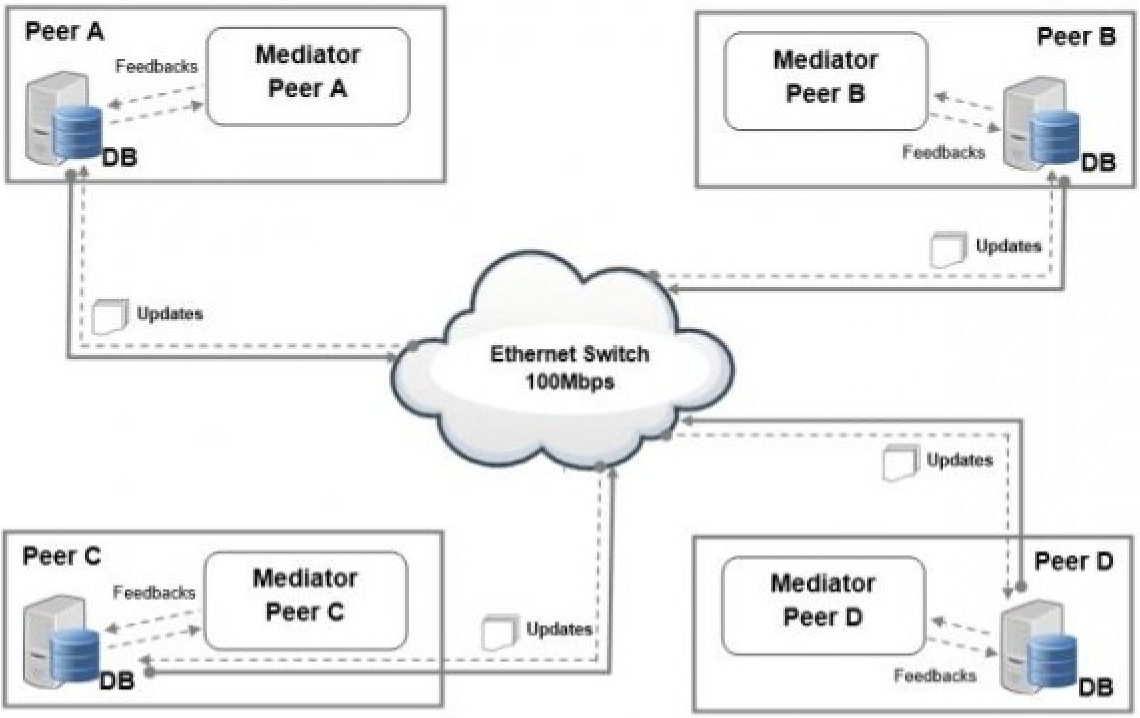
Figure 7: Algorithm 1 :



7

Figure 8: Fig. 7 :

Figure 9: Figure legend



1

Figure 10: 1 .

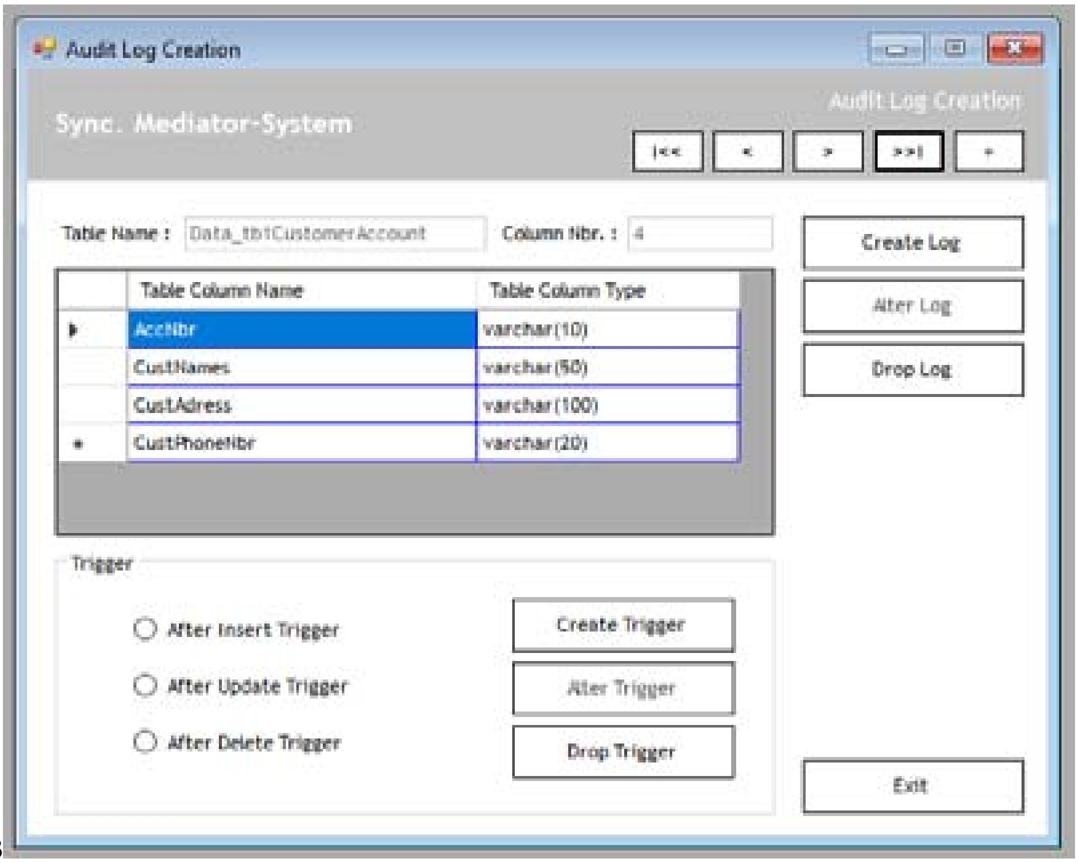
8


Figure 11: Fig. 8 :

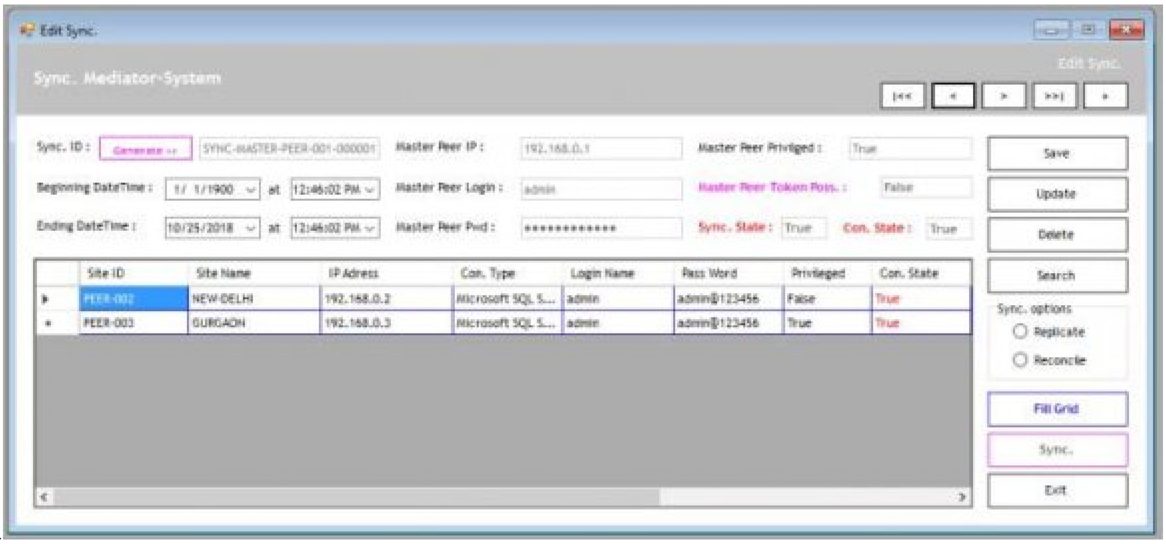
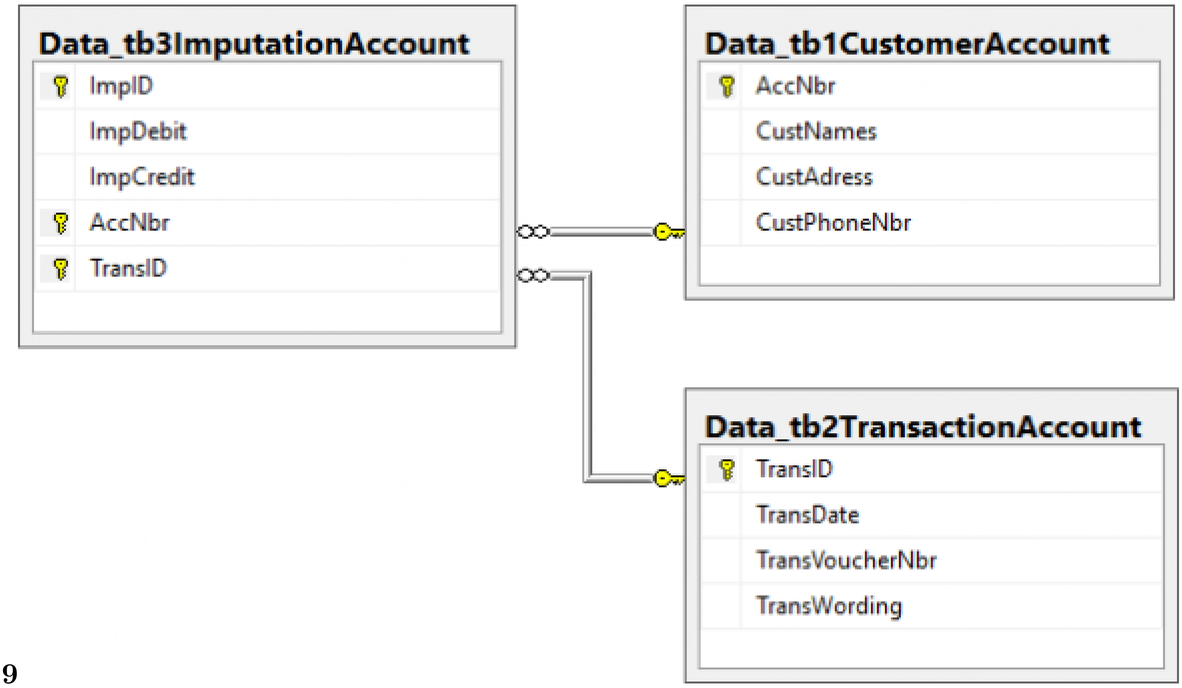
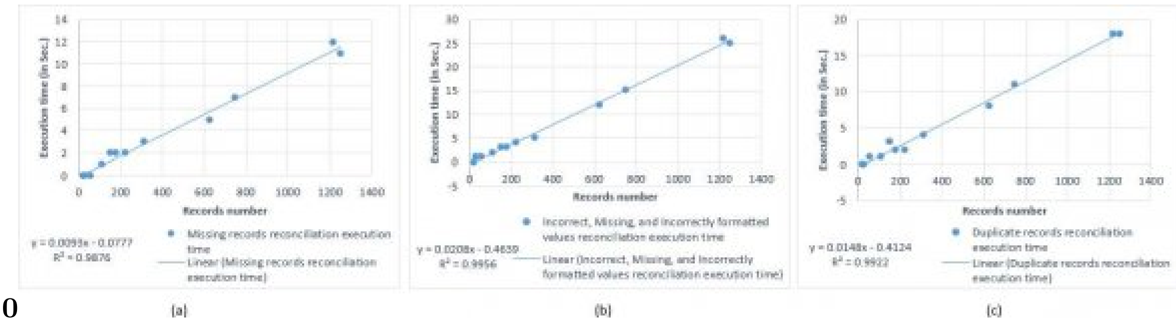
1


Figure 12: Figure legend 1 .



9

Figure 13: Fig. 9 :



10

Figure 14: Fig. 10 :

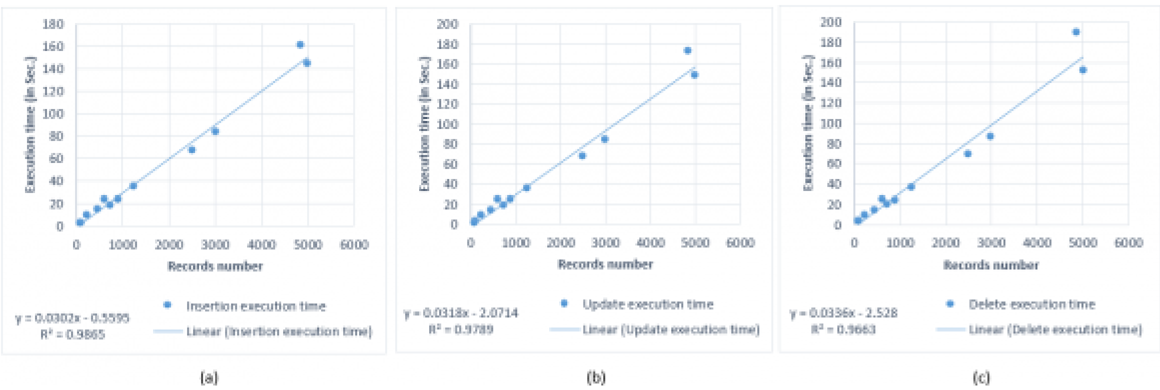


Figure 15:

11

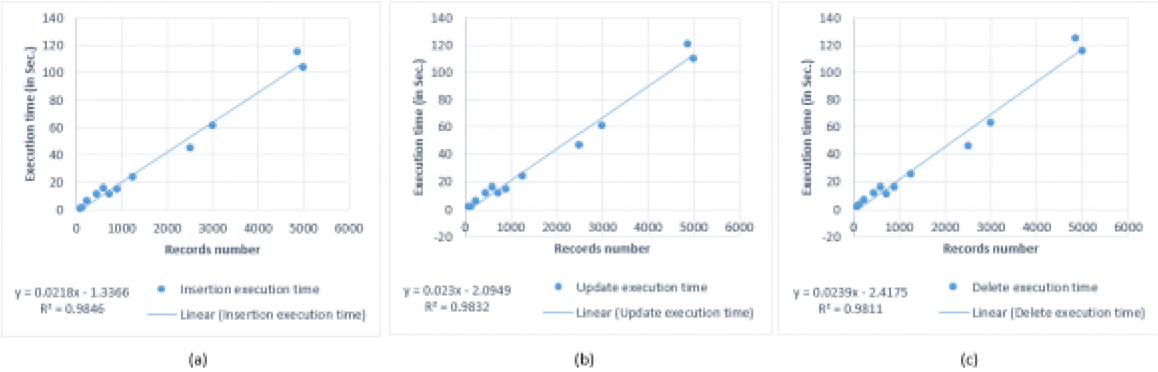


Figure 16: Fig. 11 :

12

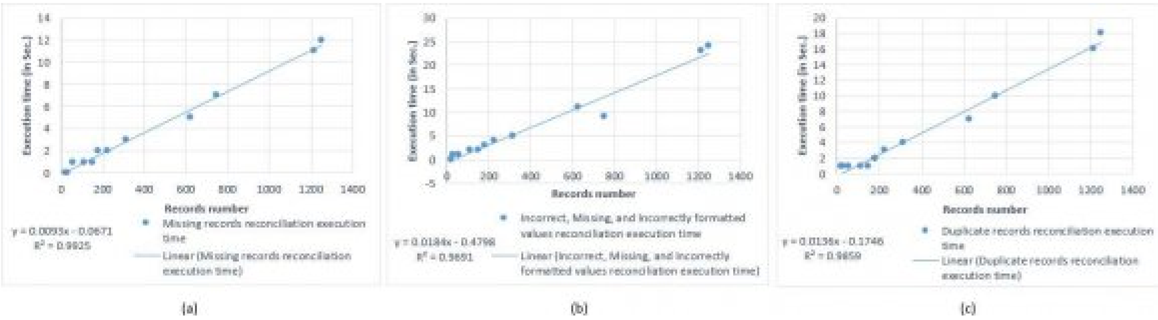


Figure 17: Fig. 12 :

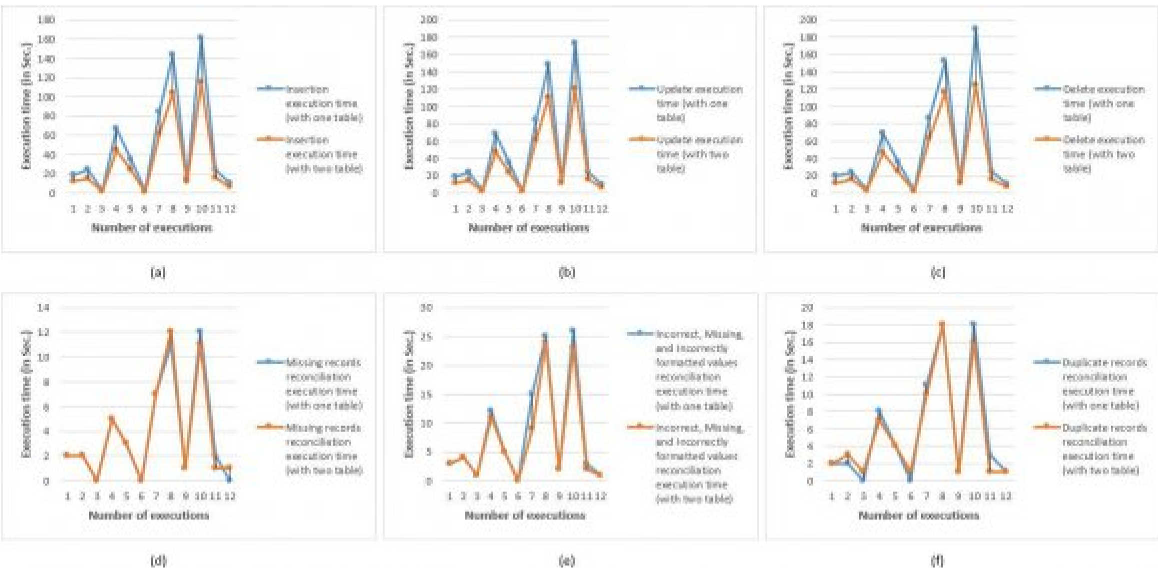
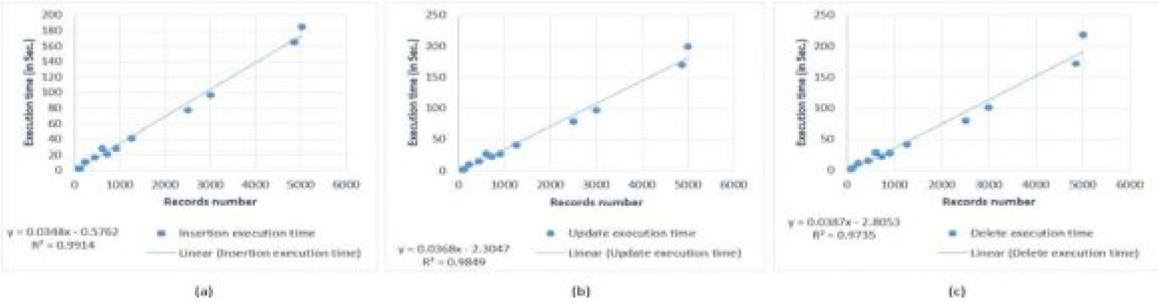


Figure 18:



13

Figure 19: Fig. 13 :

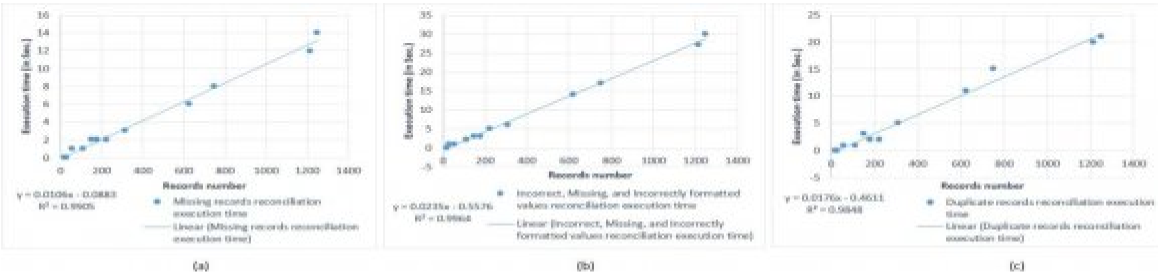
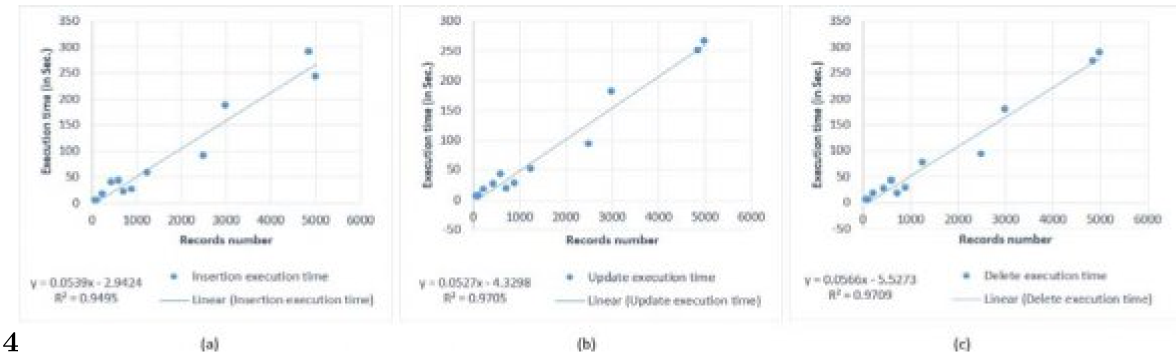
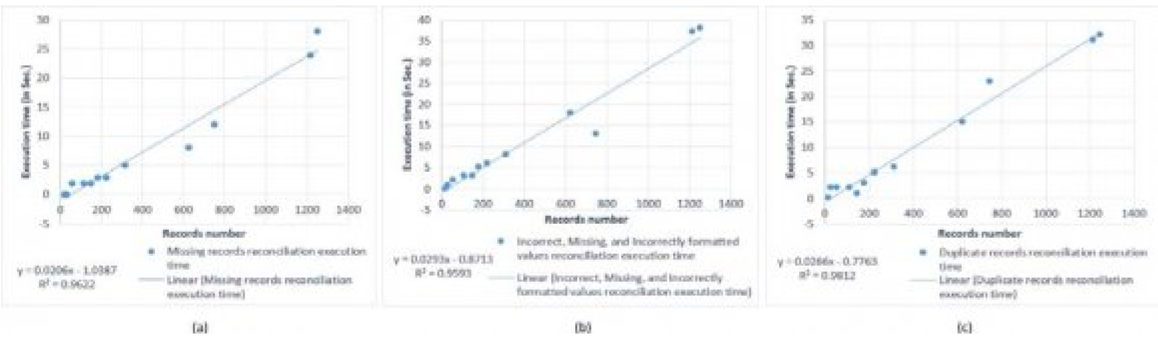


Figure 20:



14

Figure 21: Fig. 14 :



15

Figure 22: Fig. 15 :

16

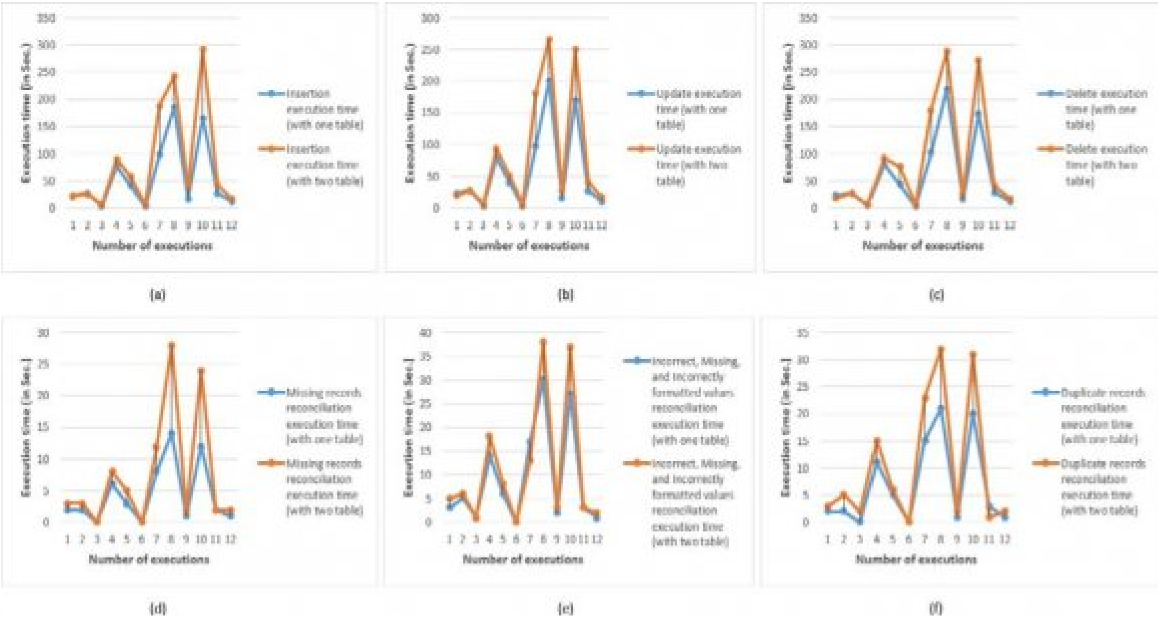


Figure 23: Fig. 16 :

17

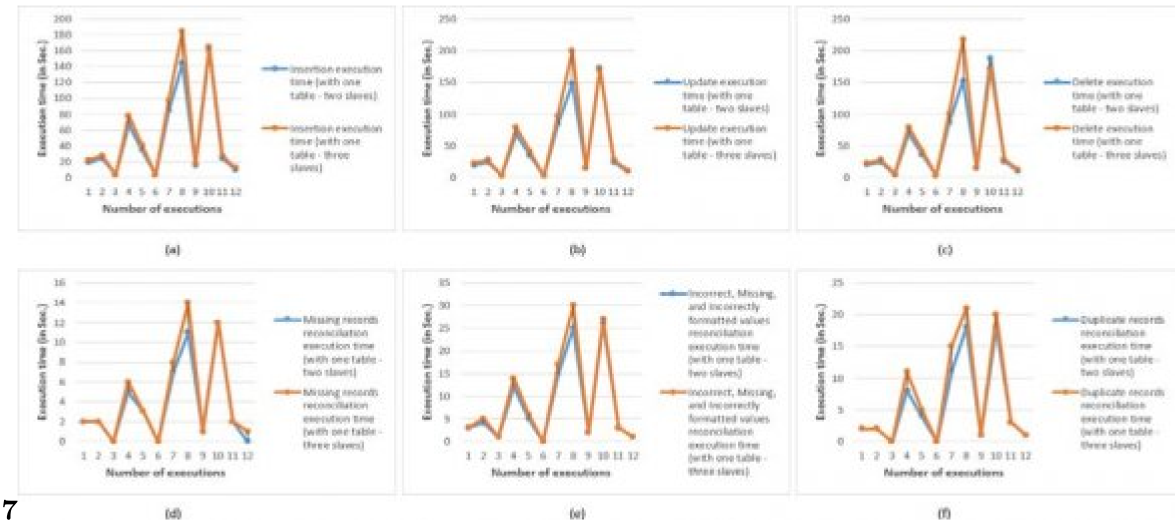


Figure 24: Fig. 17 :

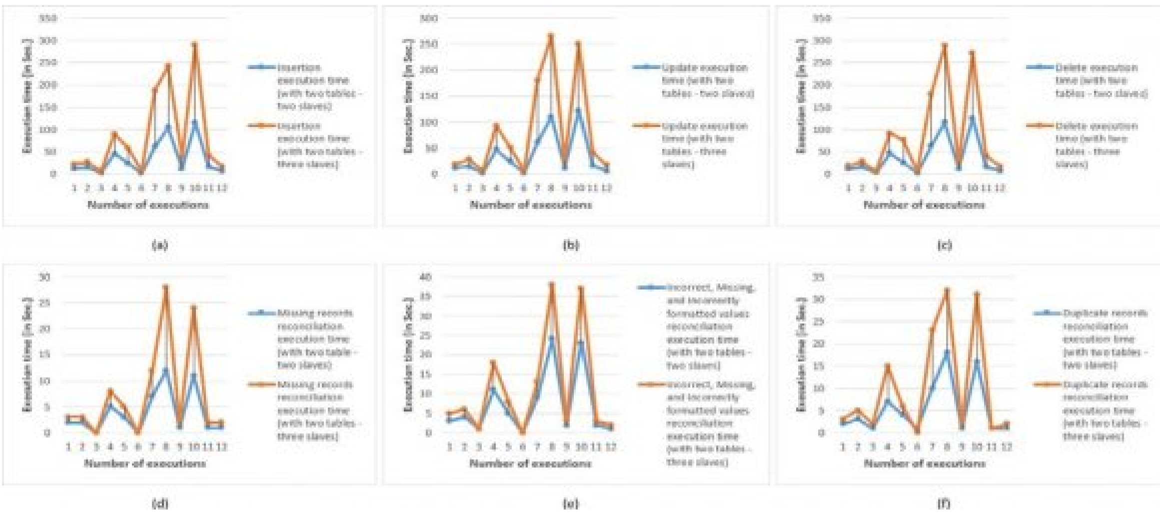


Figure 25:

( ) C
concerned record, with the new data that has just been set, and inserts it in the audit table, as shown in Fig. 6, row 6 to 8 in Slave Peer Audit-table;

[Note: 6, row 1 to 5 in Slave Peer Audit-table; ? After each Update operation of a column of data table, the "update trigger" captures the Mediation of Lazy Update Propagation in a Replicated Database over a Decentralized P2P Architecture]

Figure 26: ?

Algorithm 3: P2P Replication Algorithm for Data Insertion
Input: Master peer inserted records
Output: Transaction Commitments or Abortions Algorithm 5: P2P Replication Algorithm for Data Dele

Year 9: 10: and AuditTimeStamp ?BeginningDateAndTime and AuditTime
2
019
12 11: 12: for(cts?0 to NumberOfColumnNamesInDataTable(ts)OfSlavePee
13: Values ?Values & Row[rtm]Column[cts]
14: end for cts
15: insert in toDataTableNames(ts)InSlavePeer(p)Database (Column
16: end for rtm
17: end for ts
20: endinsertSubTransaction(Commit or Abort)
21: end for p
22: endinsertMainTransaction(Commit or Abort)
23: returnTransaction Commitments or Abortions
endinsertFunction

After records which have been inserted be replicated to slave peers, the algorithm 4 here below, which has the instructions in transactions of the update function, also runs in turn.

( ) C

Algorithm 4: P2P Replication Algorithm for Data Update Input: Master peer updated records
Output: Transaction Commitments or Abortions
beginupdateFunction()
1: beginupdateMainTransaction
2: selectall Available Slave Peers
3: for(p ?0 toNumberOfAvailableSlavePeers -1)do
4: beginupdateSubTransactionPeer(p)
5: selectall Audit Table Names in Mater Peer Database
6: selectall Data Table Names in Slave Peer(p) Database
7: for(ts?0 toNumberOfDataTableNamesInSlavePeer(p)Database -1
8: selectall Rows in Audit Table(ts) of Master Peer Databasewhere and AuditTimeStamp?BeginningDateAndTime and AuditTimeStamp?EndingDateAndTime
9: for(rtm?0 toRowsInAuditTable(ts)OfMasterPeerDatabase -1)do
10: selectall Column Names in Data Table(ts) of Slave Peer(p) Data
11: for(cts?0 toNumberOfColumnNamesInDataTable(ts)OfSlavePeer
12: if(ColumnName(cts)InDataTable(ts)OfSlavePeer(p)Database = UpdatedColumnName)then

Mediation of Lazy Update Propagation in a Replicated Database over a Decentralized P2P Architecture

Year 2019

2: Processing: ? Reading forwarded to other replicas by the refresh of updates independently transaction

14 Algorithm 6: P2P Algorithm for Data Reconciliation 45: end for rts 46: end for rtm Input: Master peer

51: updateIncorrectValuesFu

52: end if

53: end for cts

( ) C

11: 12: 13: 14: endreconcileFunction Table(ts)OfMasterPeerDatabase -1)then if(NumberOfRowsInAudit

15: To insert missing records, the algorithm 7 here is called. rts?0

16: Algorithm 7: Function to insert missing records for(rtm?0 toNumberOfRowsInAuditTable(ts)OfMaste

17: Input: DataTable(ts)OfSlavePeer(p)Database, cts, rtm repeat

18: Output: Nothing if(rts?NumberOfRowsInA Table(ts)OfSlavePeerDataba -1)then

19: begininsertMissingRecordFunction(args) if(Row[rtm]Column[0]InAudit Table(ts)OfMasterPeerDataba Row[rts]Column[0]InAuditTable(ts)OfSlavePeer(p)Database)then 1: for(cts?0 to NumberOfColumnNames

20: 2: Continue(rts ++) ColumnNa

21: 3: end repeat Values ?Values &

22: 4: end for cts else

//Call function to insert missing records 5: insert in toDataTableNames(ts)InSlavePeer(p) Database (Col

23: endinsertMissRecordFunction insertMissingRecordFunction(arguments)

24: 25: To delete duplicated records, the algorithm 8 here is called. end if else

//Call function to insert missing records Algorithm 8: Function to delete duplicated records 26: insertMis

>NumberOfRowsInAuditTable(ts)OfMasterPeerDatabase -1)then To update incorrect values, the algorith

//Reconcile duplicated records process start

© 2019 Global Jour-

1

Nbr. Obs.	Number of rows to replicate	Number of rows to reconcile
1.	723	181
2.	900	225
3.	120	30
4.	2500	625
5.	1253	313
6.	80	20
7.	3000	750
8.	5000	1250
9.	450	113
10.	4860	1215
11.	600	150
12.	235	59
Mean	1643.42	410.92
Total	19721	4931

Figure 29: Table 1 :

2

Sample numbering		Insert execution time (in Sec.)		Update execution time (in Sec.)		Delete execution time (in Sec.)	
Nbr. Obs.	Master Peer	Repli cation	Reconci liation	Repli cation	Reconci liation	Repli cation	Reconci liation
1.	B	19	2	19	3	20	2
2.	A	24	2	24	4	24	2
3.	C	3	0	3	1	4	0
4.	C	67	5	68	12	69	8
5.	A	35	3	35	5	36	4

Figure 30: Table 2 :

2

Figure 31: Table 2 ,

3

Sample numbering		Insert execution time (in Sec.)		Update execution time (in Sec.)		Delete execution time (in Sec.)	
Nbr. Obs.	Master Peer	Repli cation	Reconci liation	Repli cation	Reconci liation	Repli cation	Reconci liation
1.	B	12	2	12	3	11	2
2.	A	15	2	15	4	16	3
3.	C	2	0	2	1	3	1
4.	C	45	5	47	11	46	7
5.	A	24	3	24	5	25	4
6.	A	1	0	2	0	2	1
7.	B	61	7	61	9	63	10
8.	B	104	12	110	24	116	18
9.	A	12	1	12	2	12	1
10.	C	115	11	121	23	125	16
11.	C	16	1	16	2	16	1
12.	B	7	1	6	1	7	1
Mean		34.50	3.75	35.67	7.08	36.83	5.42
Total		414	45	428	85	442	65

Figure 32: Table 3 :

1

Sample numbering		Insert execution time (in Sec.)		Update execution time (in Sec.)		Delete execution time (in Sec.)	
Nbr. Obs.	Master Peer	Repli cation	Reconci liation	Repli cation	Reconci liation	Repli cation	Reconci liation
1.	B	22	2	23	3	23	2
2.	A	28	2	28	5	28	2
3.	C	3	0	3	1	5	0
4.	C	78	6	79	14	80	11
5.	D	41	3	41	6	42	5
6.	A	3	0	2	0	3	0
7.	B	97	8	97	17	101	15
8.	D	185	14	200	30	218	21
9.	A	17	1	16	2	16	1
10.	C	165	12	170	27	172	20
11.	D	28	2	28	3	29	3
12.	B	12	1	11	1	12	1
Mean		56.58	4.25	58.17	9.08	60.75	6.75
Total		679	51	698	109	729	81

Figure 33: Table 1 ,

4

( ) C

Figure 34: Table 4 :

4

Figure 35: Table 4 ,

5

Sample numbering		Insert execution time (in Sec.)		Update execution time (in Sec.)		Delete execution time (in Sec.)	
Nbr. Obs.	Master Peer	Repli cation	Reconci liation	Repli cation	Reconci liation	Repli cation	Reconci liation
1.	B	22	3	19	5	18	3
2.	A	26	3	28	6	28	5
3.	C	6	0	7	1	6	2
4.	C	90	8	93	18	92	15
5.	D	58	5	51	8	76	6
6.	A	6	0	6	0	6	0
7.	B	188	12	181	13	180	23

Figure 36: Table 5 :

6

Experimental scenarios	Transaction	Operator	Model	R^2	R	Prediction (to 1 Sec.)
1. Experimentation based one table stored on a master peer with two slave peers	Replication Reconciliation	Insert Update Delete Insert Update Delete	??=0.0302???0.5595+? ??=0.0318???2.0714+? ?? = 0.0336?? ? 2.528 + ? ??=0.0093???0.0777+? ??=0.0208???0.4639+? ??=0.0148???0.4124+?	98.65%	99.34%	52 records
2. Experimentation based two tables stored on a master peer with two slave peers	Replication Reconciliation	Insert Update Delete Insert Update Delete	??=0.0210???1.3366+? ??=0.0230???2.0949+? ??=0.0239???2.4175+? ??=			
3. Experimentation based one table stored on a master peer with three slave peers	Replication Reconciliation	Insert Update Delete Insert Update Delete	?? = 0.0348?? ? 0.5762 + ? 99.14% 99.57% 45 records ?? = 0.0368?? ? 2			
4. Experimentation based two tables stored on a master peer with three slave peers	Replication Reconciliation	Insert Update Delete Insert Update Delete	?? = 0.0539?? ? 2.9424 + ? 94.95% 97.44% 73 records ?? = 0.0527?? ? 4			

Figure 37: Table 6 :

.1 Acknowledgement

Firstly, we are grateful to the Grace of Almighty God. We would also like to thank the academic corps of the Butembo (D. R. Congo) Institute of Building and Public Works for their encouragement and follow-up of our investigations. On finish, we thank the Research Technology and Development Centre (RTDC) of Sharda University, for its facilities to realize this work.

[Oracle Corporation web site ()] , *Oracle Corporation web site* 2018.

[Oracle Corporation web site ()] , *Oracle Corporation web site* 2018.

[ApexSQL LLC web site ()] , *ApexSQL LLC web site* 2018.

[Zhang ()] 'A Novel Replication Model with Enhanced Data Availability in P2P Platforms'. T Zhang . *International Journal of Grid and Distributed Computing* 2016. 9 (4) p. .

[Kituta et al. ()] 'A systematic review on distributed databases systems and their techniques'. K Kituta , S Kant , R Agarwal . *Journal of T heoretical and Applied Information Technology* 2019. 96 (1) p. .

[Kudo ()] 'An implementation of concurrency control between batch update and online entries'. T Kudo . *18 th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems -KES2014, Procedia Computer Science*, 2014. 35 p. .

[George and Balakrishnan ()] 'An optimized strategy for replication in peer-to-peer distributed databases'. A George , C Balakrishnan . *IEEE International Conference on Computational Intelligence and Computing Research*, 2012.

[Kituta et al. ()] 'Analysis of database replication protocols'. K Kituta , S Kant , R Agarwal . *International Journal of Latest T rends in Engineering and T echnology* 2018. 2018. p. . (Special Issue ICRMR)

[Filip et al. ()] 'Considerations about an Oracle Database Multi-Master Replication'. I Filip , C Vasar , R Robu . *IEEE 5th International Symposium on Applied Computational Intelligence and Informatics*, 2009.

[Souri et al. ()] 'Consistency of data replication protocols in database systems: A review'. A Souri , S Pashazadeh , A Navin , H . *International Journal on Information Theory (IJIT)* 2014. 3 (4) p. .

[Cormen ()] T Cormen , H . *Introduction to Algorithms*, (London, England) 2012. The MIT Press. (4th ed.)

[Fatos ()] 'Data Replication in Collaborative Systems'. X Fatos . *IEEE Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, 2012.

[T Ing and Yu ()] 'Database Replication T echnology having high Consistency Requirements'. Z T Ing , W Yu . *IEEE Third International Conference on Information Science and Technology*, 2013.

[Database Schema Difference Reconciliation ()] `https://www.perpetual-beta.org/weblog/mysql-diff.html` *Database Schema Difference Reconciliation*, (Jonathan, H., MySQL_Diff) 2018. 2018.

[Silberschatz et al. ()] *Database system concepts*, A Silberschatz , H F Korth , S Sudarshan . 1997. New York: McGraw-Hill.

[Diallo et al. ()] 'Distributed Database Management T echniques for Wireless Sensor Networks'. O Diallo , Joel Rodrigues , J Sene , M Lloret , J . *IEEE T ransactions on Parallel and Distributed Systems* 2015. 26 (2) p. .

[Shahin et al. ()] *Dynamic Data Allocation with Replication in Distributed Systems. 30 th IEEE International Performance Computing and Communications Conference*, K Shahin , G Pedram , D Khuzaima . 2011.

[Mansouri and Buyya ()] 'Dynamic replication and migration of data objects with hot -spot and coldspot statuses across storage data centers'. Y Mansouri , R Buyya . *Journal of Parallel and Distributed Computing* 2018. 126 p. . (Publisher: Elsivier)

[Santana and Francesc ()] 'Evaluation of database replication techniques for cloud systems'. M Santana , Enrique , J Francesc , D . *Computing and Informatics* 2015. 34 p. .

[Experian Ltd web site ()] *Experian Ltd web site*, `https://www.edq.com/uk/glossary/data-reconciliation/` 2018.

[Sebastian ()] *Fundamentals of SQL Server*, M Sebastian . 2012. 2013. New York, United States of America: Simple T alk Publishing.

[Pandey and Shanker ()] 'IDRC: A Distributed Real-T ime Commit Protocol'. S Pandey , U Shanker . *th International Conference on Smart Computing and Communications ICSCC 2017*, 2017. 125 p. . (Publisher: Elsivier)

[Kothari and Garg ()] 'In-House'. C Kothari , R Garg , G . *Research methodology methods and techniques*, 2014. 2018. 20. (Microsoft Corporation web site)

[Kirtikumar ()] *Oracle Streams 11g Data Replication*, D Kirtikumar . 2011. New York, United States of America: McGraw-Hill.

[Vu et al. ()] *Peer-to-Peer Computing -Principles and Applications*, Q Vu , M Lupu , C Ooi . 2010. Springer.

[Pragmatic Works Inc. web site ()] `https://dbconvert.com/30` *Pragmatic Works Inc. web site*, 2018. 2018.

[Spaho ()] E Spaho . *Modeling and Processing for Next -Generation Big-Data T echnologies. Modeling and Optimization in Science and Technologies*, F Xhafa, L Barolli, A Barolli, P Papajorgji (ed.) 2015. Springer. 4 p. . (P2P Data Replication: T echniques and Applications)

[Kituta et al.] 'Synchronous and Asynchronous Replication'. K Kituta , R Agarwal , B Kaushik . *International Conference on Machine Learning and Computational Intelligence*, 2017. (International)

[Gudakesa et al. ()] 'T woways database synchronization in homogeneous DBMS using audit log approach'. R Gudakesa , M Sukarsa , G Sasmita . *Journal of T heoretical and Applied Information Technology* 2014. 65 p. .

[Magdalena ()] 'The Replication Technology in E-learning Systems'. N Magdalena , I . *Procedia -Social and Behavioral Sciences* 2011. 28 p. . (Publisher: Elsivier)

[Wiesmann ()] 'Understanding Replication in Databases and Distributed Systems'. M Wiesmann . *IEEE 20th International Conference on Distributed Computing Systems*, 2002.

[Özsu and Valduriez ()] M T Özsu , P Valduriez . *Principles of Distributed Database Systems*, (New York, USA) 2011. Springer Science & Business + Media. (3rd ed.)