A New Non-Blocking Validation Protocol for Eager Replication of Databases over a Decentralized P2P Architecture

Katembo Kituta Ezéchiel, Shri Kant, Ruchi Agarwal

Abstract*: *Replicating a database on a decentralized P2P network using the eager approach is a difficult problem, especially since participants (peers) are dynamic on such kinds of networks. A second defect is conflicting transactions executed concurrently by different peers to update the same data. These problems cause the perpetual abortions of transactions so that replicas remain always inconsistent. Thus, this article introduces a new Four-Phase-Commit (4PC) validation protocol that allows the completion of transactions with available peers and recovers unavailable ones when they re-join the network. Nested transactions and distributed voting technique were used to arrive at an algorithm that was implemented with C#. An experimentation scenario has made it possible to measure its performance and has finally revealed that the new algorithm is effective because in real-time it can replicate a large number of records; it can queue the records of the absent peers in order to distribute these updates to them when they become present again.

Index Terms*: *Eager replication, Two-Phase-Commit (2PC) protocol, Read-One Write-All (ROWA), Peer-to-Peer (P2P).

I. INTRODUCTION

The replication of data on a P2P network generally requires the use of the multi-master approach given the nature of these kinds of networks [1], [2], [3]. These networks are made in such a way that a peer may or may not be available (dynamic character of the P2P network) and the peers have the same responsibilities (absence of the notion master / slave and client / server) [4], [5]. Thus, therefore, when one wants to answer the question "where" must be made updates, the answer deviate "everywhere". However, when it comes to the question of "when" updates need to be spread to other peers, however, it is not long to deny that it is not in real-time that it must take place. This is why the lazy approach has already made a good score in the diffusion of updates on a P2P network because it implies that the fact that the participants are dynamic they will only have to miss the updates in real-time [6], [12].

Nevertheless, in order to maintain the reliability of the replicas at all times and to ensure the effective availability of data on all the peers, it is the eager approach that should be used [8], [10], [15]. While starting the operating principle of the current protocol "Two-Phase-Commit (2PC)" that governs this approach, the transaction that distributes the updates must update all peers at the same time or nothing, thus the rule "Read-One, Write-All (ROWA)" [3], [20], [21]. A more when all the peers have to update the replicas concurrently, when more than one transaction tries to update the same replica there is interlock between transaction. This phenomenon even ends up aborting other pending transactions when the deadline expires [3], [7], [9], [11]. These two problems mentioned above are susceptible to perpetual abortion of transactions. This led us to think in another way and to direct our thinking towards a new" Four-Phase-Commit (4PC)" transaction validation protocol. This new protocol makes it possible to update available peers and recover absentees when they become available again, thus the new standard "Read-One, Write-All Available (ROWA-A). Therefore, to achieve this objective, the outline of this document is organized as follows: after having presented the context of the research as well as the status of the problem in this first section, the second section reviews the literature of the work that also addressed the same angle of idea, the third presents the methodology borrowed, the fourth presents the result obtained and finally the sixth section concludes this study.

II. ANALYSIS OF EXISTING APPROACHES

To manage a transaction, the Distributed Database Management System (DDBMS) always uses a distributed reliability protocol. This protocol is a language that manages the states of a transaction namely the beginning of the transaction, the operators of reading and writing within the transaction and the end of the transaction [11], [15]. The transaction itself is a succession of operations that the user makes to the database by bringing it from a consistent state to another consistent state, as shown in Fig 1. [11], [14].

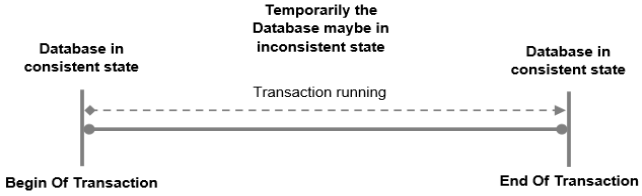


Fig. 1. Protocol of Transaction running.

Revised Manuscript Received on July 5, 2019.

Katembo Kituta Ezéchiel, Department of Computer Science and Engineering, Sharda University, Greater Noida, India, kkitutaezechiel@yahoo.com

Shri Kant, Research and Technology Development Centre, Sharda University, Greater Noida, India.

Ruchi Agarwal, Department of Computer Applications, JIMS Engineering Management Technical Campus, Greater Noida, India.

The begin and end instructions are successively nothing other than to specify that the transaction has just begun and that the transaction has just finished. Thus, the transformation made to the database is achieved by executing all the instructions, usually operators encapsulated in the same transaction, that is to say the execution of all instructions after the begin and before the end. The end of a transaction can be a success or a failure. If it ends with a success we say that the transaction "commit" and therefore in this case all the instructions execute successfully and change the state of the database otherwise we say that the transaction "abort" when the execution of instructions has no effect on the database, hence the principle of atomicity [10], [11], [13], [14], [15].

However, in eager replication, we use the ROWA principle, a distributed reliability technique. This principle optimally manages the end of a transaction to see in which conditions the transaction can be validated. According to this principle, currently the validation can be One-Phase, Two-Phases or Three-Phases [3], [17]. However, after having observed the result of these three validation algorithms, our hypothesis is that: "None of these three protocols can satisfy the validation of a transaction on a P2P network given the dynamicity of the participants". But to arrive at a new protocol that can satisfy a P2P network, we will have to improve the Two-Phase protocol because it is the most used currently and already implemented in almost all DDBMS. First it is the improvement of the one phase. Moreover, the Three-Phases one was made from it but has never been implemented in a commercial DDBMS apart from its implementation as various experiments of researchers [19], [20], [21]. So, in the end we have found that it does not solve also the problem of the validation of a transaction on a topology whose participants are dynamic. Thus, this work sets itself the objective of adapting the Two-Phase-Commit (2PC) validation protocol to the decentralized P2P topology and thus to develop a new Four-Phases protocol (Four-Phase-Commit (4PC)) validation protocol that can take into account the behavior of P2P networks.

The 2PC protocol, being a reference model, consists of a consensus on the execution of a transaction initiated by a master site called coordinator and one or more slave sites called participants. In this consensual process, the validation of a transaction is due to the feedback given by the participants, which cast a “yes” vote for “commit” or “no” for “abort”, so that the general transaction validation decision is made if all participants voted "yes" whereas it is enough that one participant votes "no" so that the transaction be invalidated or abort globally [3], [15].

Two phases validation algorithm work as follow [3], [11]:

Phase 1: Vote

- The application initiator of the transaction requests the coordinator to validate its transaction;
- The coordinator sends a message (a commit request) to each participant “p”;
- Then the coordinator waits for answers;
- If the transaction succeeds on the participant “p” then it gives a feedback “yes” else, it gives “no” as feedback.

Phase 2: Validation

- If all participants have given as feedback “yes” then coordinator send to them a “commit” message else, it sends an “abort” message to all participants which have sending a “yes”;
- Then the coordinator waits for participant acquittals;
- When a participant “p” which has sending a “yes” receive a “commit” or an “abort” message it terminates locally the transaction accordingly;
- Finally, it also sends an acquittal to the coordinator.

However, firstly this algorithm is commonly used by DDBMS developers, our critical analysis proves that it involves multiple messages that the transaction coordinator must exchange with participants. Thus, its use in a computer environment, where a large number of components are networked a number of failures (coordinator, participants, network) may result in the loss of messages, the loss of the "request for validation", the loss of vote, the loss of the vote result, the loss of recognition. etc. is possible [20]. And beyond these aforementioned failures, executing a large number of instructions, as described in the algorithm above, for a transaction that must update several peers certainly that it would not optimize the execution time of the transaction [19].

Secondly, based on the atomicity, one of the execution properties of a transaction in the replicate databases environment, it is expected that the effects of a transaction should be passed on to all sites or no site [3], [15]. So knowing that peers or participating sites of a P2P network may be available or unavailable, pretending to use an eager replication approach on a decentralized P2P architecture to consider mutual convergence of replicas at any time, one must first think of an efficient approach that can use a non-blocking protocol. In this logic the protocol would update the available peers so that the transaction ends and thus avoid possible abortion due to the non-availability of peers and minimize the response time of the transaction. And subsequently to recover the unavailable peers when they become available again. Thus, this article will attempt to improve the Two-Phases validation algorithm by the distributed voting technique [16]. In this new approach, if the quorum of the number of sites is reached, the transaction will be validated each time. Then so, when writing, updates are reflected on all available replicas so that when reading, it is necessary to read enough replicas to ensure you get at least a copy of the most recent value; hence a new standard “Read-One, Write-All Available (ROWA-A)”.

III. METHODOLOGICAL APPROACHES

This search will combine the technique of nested transactions with that of distributed voting to arrive at a 4PC validation algorithm. The goal is to unload the single atomic transaction that was originally supposed to do everything in the eager replication process with the 2PC protocol and share its load with the sub-transactions that should run site-by-site according to the main transaction; hence the notion of nested transactions.

A. Nested transaction technique

The technique of nested transactions is supported by almost all DDBMSs. We use it instead of the flat transaction (from the 2PC protocol) to share the responsibility for the main transaction with the sub-transactions which in turn will perform the assigned tasks but still being dependent on the main transaction. Thus it will be possible to validate or invalidate the sub-transaction on any peer regardless of the main transaction. But the sub-transaction must always depend on the main transaction in the sense that if the principal is validated, the changes made by the sub-transaction that has already been validated remain durable otherwise the changes made by the sub-transaction whatever validated by it alone will not have an effect on the database. The nested transaction language is sequential and comes in the form of the following structure:

```
Begin_Of_Main-Transaction T
   Read/Write operators in T
   Begin_Of_Sub-Transaction T1
      Read/Write operators in T1
   End_Of_Sub-Transaction T1
   Begin_Of_Sub-Transaction T2
      Read/Write operators in T2
   End_Of_Sub-Transaction T2
      …
   Begin_Of_Sub-Transaction Tn
      Read/Write operators in Tn
   End_Of_Sub-Transaction Tn
End_Of_Main-Transaction T
```

This procedure works according to the transaction execution technique, but it can be combined with externally programmed logic to ultimately result in a distributed voting algorithm.

B. Distributed voting technique

The major problem of P2P networks being the unavailability of peers due to the absence of the network and the conflict between transactions that end up causing the perpetual abortion of transactions, the current algorithm will be based on the distributed voting technique. In this logic before a transaction executes, the peers will have to vote each time with a "Yes" if the connection of a peer is open and if it does not show a transaction conflict. The result of these votes will be calculated each time to deduce the quorum.

Majority consensus method

The quorum being the number of participants a meeting must meet to be able to deliberate, the actual distributing voting technique will be based on the majority consensus method. This method allows a transaction to be performed if the majority of peers voted "Yes", this means that the majority of peers is available or has the connection opened and no transaction conflict has been identified on the performing time of the transaction. It will be of four phases, namely:

1) Voting phase: when a transaction is initiated on a peer, it first check the connected peers and test the potential conflict of the current transaction with other running transaction on the connected peers. If a peer p is connected and no conflict is highlighted from the pending list of running transaction the vote result is "Yes" and the transaction is queued in the pending list else if a peer p is connected and a conflict is highlighted from the pending list of running transaction, but the coordinator or initiator of the running transaction is down, the current transaction take the status "running" and pends the old running one so that the vote result is "Yes" and the transaction is queued in the pending list else the vote result is "No";
   Validation (Acceptance/Rejection) phase: the majority number of vote is calculated so that if the total number of votes with "Yes" as result reach the quorum the transaction is accepted else the transaction is rejected. This quorum can be specified in advance by a user according to percentages which must be equal or greater than 60% (considered as the default quorum if a user didn't specify) and the majority of votes is given by the formula:

$$VoteMajority = \frac{NumberOfAvailabe\ peers\ (\text{"Yes"}\ Votes)}{NumberOfPeers\ (All\ listed\ peers)} * 100.$$

   So, if the Vote majority is equal or greater than the Quorum the transaction is accepted else the transaction is rejected;
2) Performing phase: if accepted, the transaction is performed accordingly and at the end after the commitment of the modification operator be sent to the initiator peer, the transaction is removed from the pending list, else (if abort or rollback) the transaction is removed from the pending list so that all deferred transaction be taken in consideration for their next attempting execution. This process is repeated for each available slave peer;
3) Recovery phase: after the performing phase, all peers which have voted "No" in the first phase and all whose completion result was "Abort or Rollback" in the third phase, are retained as well as the updated data in order to be queued for ulterior updating when the respective peers will become available. The recovery procedure must initiate a new transaction or continue with the old transaction that the same master peer may have initiated and whose slave peer may have interrupted due to its unavailability. Also this same procedure can continue with the transaction that the master peer may have left running and already pending by other transactions having noted the unavailability of the initiator (master peer or coordinator).

Thus, the fourth phase of this algorithm gives rise to the method of the queue and which will be elucidated in the lines that follow.

Queue method

This method is used for two things: the management of the conflicting transactions which will be materialize by the pending of the transaction when it finds another in real execution and the queuing of the updates which will have to be reflected to a subset of qualified unavailable peers when they become available again.

- *Transactions pending list*: As shown in Fig. 2., a table to list pending transactions is required to store the properties of running and pending transactions. The component attributes of this table are: the pending list ID (primary key), the table name, type of the write query, transaction status, the timestamp, the synchronization ID and the initiator peer ID.

	pendListID	tableName	typeQuery	transStatus	pendTimestamp	syncID	peerSiteID
	1	Data_tb1Custom...	Update	Running	2019-01-05 02:20:...	SYNC-MASTER-PEER-001-000001	PEER-001
	2	Data_tb1Custom...	Insert	Running	2019-01-05 02:30:...	SYNC-MASTER-PEER-001-000001	PEER-001
	3	Data_tb1Custom...	Delete	Pending	2019-01-05 02:42:...	SYNC-MASTER-PEER-001-000001	PEER-001
▸*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fig. 2. Pending transactions table.

- *Pending updates*: It will be sufficient for at least one peer to be absent due to unavailability on the network or any other type of problem such as transaction conflict that interferes with the commit of the sub-transaction belonging to that specific peer so that the updates that it is supposed to receive be put on hold. These updates are then temporarily stored in a queued table, as shown in Fig. 3., in order to be transmitted to this peer as soon as it becomes available. This queue table is composed of the following attributes: the queue data ID (primary key), the string of the write query, the unavailable peer ID, the timestamp and the synchronization ID.

	queueDataID	stringQuery	peerSiteID	queueTimesta...	syncID
	27	insert into Data_tb1Custo...	PEER-002	2019-01-05 16:0...	SYNC-MASTER-PEER-001-000003
	28	insert into Data_tb1Custo...	PEER-003	2019-01-05 16:0...	SYNC-MASTER-PEER-001-000003
	29	insert into Data_tb1Custo...	PEER-002	2019-01-05 15:4...	SYNC-MASTER-PEER-001-000004
	32	update Data_tb1Custome...	PEER-003	2019-01-05 17:0...	SYNC-MASTER-PEER-001-000002
	33	delete from Data_tb1Cust...	PEER-003	2019-01-05 17:1...	SYNC-MASTER-PEER-001-000002
▸*	NULL	NULL	NULL	NULL	NULL

Fig. 3. Queued data table.

Algorithmic method

As we indicated before, this new algorithm will have to combine initially and experimentally the internal logic of the execution of a transaction already implemented in commercial DDBMSs and external sequences. This is how we first assume that each peer's DDBMS works with the Two-Phase-Commit (2PC) protocol. Then we assume that the database is homogeneous and fully replicated. Then the protocol of eager replication on a decentralized P2P architecture is as follows: either W (x) a write transaction and x a copy replicated by the peers A, B, C and D. The Fig. 4 here below depicts how transactions update different copies at all Peers and before commit changes are forwarded to all peers.

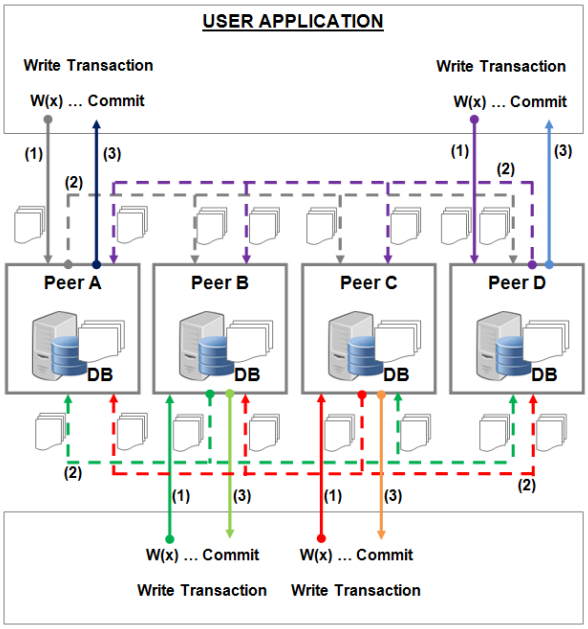


Fig. 4. Protocol of eager decentralized P2P Replication.

Arrows (1) and (3) illustrate the communication with the user application. For each write transaction (main) directed to any peer in the topology, before the user application receives the commit (or abort) message indicated by the arrow (3), the updates are transmitted by arrow (2) to all available peers. It is this arrow (2) which is a sub-transaction, repeated for each available slave pair. Algorithm 1 establishes the pseudo-code of the function that manages the distributed voting technique based on the majority consensus and the queue method for the data manipulation or write operators.

Algorithm 1: *P2P Distributed Voting Algorithm for write operators*

Input: A set of slave peers, Quorum, Master table name, Data set, Number of rows
Output: Transaction Commitment or Abortion
begin votingFunction(SlavePeers, Quorum, MasterTableName, DataSet, NumberOfRows)
1: **begin** votingMainTransaction
2: **select** *all ColumnNames from MasterTableName*
3: **if** (*QueryType = Insert*) **then**
4: **for** *(cn ← 0 to NumberOfColumns – 1)* **do**
5: Columns ← Columns & "," & MasterTableName.Column(cn)
6: **end for** *cn*
7: **end if**
8: **for** *(rm ← 0 to NumberOfRows – 1)* **do**
9: **run** {***insert***/***update***/***delete***} operators on *MasterTableName*
10: **end for** *rm*
11: **select** *all SlavePeers*
12: **for** *(p ← 0 to NumberOfSlavePeers – 1)* **do**
13: **if** *(SlavePeer(p).ConnexionState = "True")* **then**
14: **select** *all transaction* **from** *TransPendingListTable*
15: **for** *(t ← 0 to NumberOfTransactions – 1)* **do**

```
16:         if (SlavePeer(p).TransactionConflict = "True") then
17:           if (Transaction(t).CoordinatorConnexionState = "True") then
18:             SlavePeer(p).Availability ← "False" //Vote = "No"
19:             end for t
20:           else
21:             update PendingListTable set "Pending" ConflictingTransaction
22:             SlavePeer(p).Availability ← "True" //Vote = "Yes"
23:             insert in to PendingListTable CurrentTransaction
24:           end if
25:         else
26:           if (t = NumberOfSlavePeers – 1) then
27:             SlavePeer(p).Availability ← "True" //Vote = "Yes"
28:             insert in to PendingListTable CurrentTransaction
29:           end if
30:         end if
31:       end for t
32:     else
33:       SlavePeer(p).Availability ← "False" //Vote = "No"
34:     end if
35:   end for p
36:   select all Available Slave Peers
37:   VoteMajority ← (NumberOfAvailabePeers ("Yes" Votes) / NumberOfPeers (All listed peers)) *100
38:   for (p ← 0 to NumberOfAvailableSlavePeers – 1) do
39:     begin insertVotingSubTransactionPeer(p)
40:       if (VoteMajority ≥ Quorum) then //Accepted
41:         for (rs ← 0 to NumberOfRows – 1) do
42:           insert in to Slave(p)TableName (Columns) values (DataSet)
43:         end for rs
44:         delete from PendingListTable CurrentTransaction
45:       else //Rejected
46:         delete from PendingListTable CurrentTransaction
47:       end if
48:       if (TransactionStatus = "NoError") then
49:         SlavePeer(p).TransCompletionStatus ← "Commit"
50:       else
51:         SlavePeer(p).TransCompletionStatus ← "Abort"
52:       end if
53:     end insertVotingSubTransaction(Commit or Abort)
54:   end for p
55:   if (VoteMajority < Quorum) then
56:     end insertVotingMainTransaction(Abort)
57:   else if (VoteMajority < 100%) then
          //Call function to queue data for unavailable slave peers
58:     QueueDataFunction(args)
59:     end if
60:   end votingMainTransaction(Commit or Abort)
61:   return Transaction Commitment or Abortion
end votingFunction
```

The fourth phase of the consensus majority method provides that the data that should be received by the unavailable slave peers and those whose transaction (3rd phase) has aborted because of erroneous data or other type of error during execution, must be queued for a later recovery procedure. However, the 2nd algorithm below is a function that locally puts the data of each peer responding to one of two conditions in the queue.

Algorithm 2: *P2P Distributed Voting Algorithm for Data Queuing*

```
Input: A set of slave peers, Number of rows
Output: Data Set in Queue Table
  begin QueueDataFunction(SlavePeers, NumberOfRows)
1:      select all Slave Peers where Availability = "False" or TransCompletionStatus = "Abort"
2:      for (p ← 0 to NumberOfNonUpdatedSlavePeers – 1) do
3:        for (rs ← 0 to NumberOfRows – 1) do
4:          insert in to QueueTable values (DataSet & NonUpdatedSlavePeers)
5:        end for rs
6:      end for p
end QueueDataFunction
```

However, to maintain referential integrity and data relationships, because there may be updates that require prior reflection of pending data before running algorithm 1 again, the recovery procedure must first attempt to recover copies of other peers who have not participated in previous transactions, if they are finally available. Thus, for each synchronization process, the system asks the user if he first wants to apply pending updates (by algorithm 3 here below) before continuing, when he wants to execute the transaction. The recovery procedure is nothing more than a function that will not be specific to a modification operator, but will run according to the operators encountered in the queue table, in the query string. But also in order not to burden the execution process of the algorithm 1 (insert, update and delete transaction), the developer can choose to implement the third function (algorithm 3) in a separate process and ensure that it automatically triggers after a laps of time to propagate and distribute pending updates to the relevant peers as soon as they are available again.

Algorithm 3: *P2P Recovery Management Algorithm*

```
Input: A set of queued data and relevant peers
Output: Transaction Commitment or Abortion
  begin recoveryFunction(Queued data set, Relevant peers)
1:    for (p ← 0 to NumberOfRelevantPeers – 1) do
2:      begin recoveryMainTransaction
3:        select all Updates from QueuedDataSet where peerID = RelevantPeerID(p)
```

```
4:      if (RelevantPeer (p).ConnexionState = "True")
        then
5:        begin recoverySubTransactionRelevantPeer(p)
6:          for (r ← 0 to NumberOfRowsInQueueDataSet –
            1) do
7:            run StringQuery(r) {insert/update/delete}
8:          end for r
9:          if (TransactionStatus = "NoError") then
10:           Peer(p).TransCompletionStatus ← "Commit"
11:           delete from QueuedDataSet where peerID =
              RelevantSlavePeerID(p)
12:         else
13:           Peer(p).TransCompletionStatus ← "Abort"
14:         end if
15:       end recoverySubTransaction(Commit or Abort)
16:     end recoveryMainTransaction(Commit or Abort)
17:   end for p
18:   return Transaction Commitment or Abortion
end recoveryFunction
```

Subsequently, the statistical technique will be used to compute the practical complexity as well as analyse the performances of this algorithm. However, before proceeding to implementation in the form of direct experimentation, let us present this technique in the following section.

C. Statistical technique

One of the criteria for comparison between algorithms is its time-effective complexity, that is to say which results in computation time measurements [17]. This section will present the method of illustration of the efficiency of these new algorithms according to a certain number of parameters, namely: execution time, number of records, and number of peers. Thus, to analyse this effectiveness, the linear regression test with the random sampling technique will be used in order to predict the execution time based on the number of records and the number of peers. The purpose of the regression analysis is to explain a variable Y using one (X) or more variables (X_1... X_n). The variable Y is called the dependent variable, or variable to be explained, and the variables X_i (i = *1*... n) are called independent variables, or explanatory variables [18].

For this case, starting from a sample taken from the execution time, the number of records and the number of peers, we will explain values taken by the execution time (Y_i) which is the dependent variable and the number of records (X_{i1}) and the number of peers (X_{i2}) which are independent variables with $i \in [1, n]$. So, the prediction model will be formulated in terms of random variables as follow: $Y_i = b_0 + b_1 X_{i1} + b_2 X_{i2} + \varepsilon_i$ where:

- $i = 1, 2, \ldots, n$
- b_0 is the constant term;
- b_1 and b_2 are coefficients of the regression;
- ε_i: is the model error that expresses the missing information in the linear explanation of the values of Y_i from X_{i1}, X_{i2} (random factors not taken into account).

Thus, to perform statistically the test of the significance of each independent variable (X_{i1}, X_{i2}) the hypothesises are evoked as follow:

✓ Null hypothesis (H_0): X_{ik} is not a significant predictor of Y_i.

✓ Alternative hypothesis (H_1): X_{ik} is a significant predictor of Y_i.

So, the coefficient of determination (R^2) will be calculated to determine the ratio of the proportion of variation of the dependent variable explained by the regression model, that is to say to determine the degree of dependence of the execution time (Y_i) based on the number of records (X_{i1}) and the number of peers (X_{i2}). From this will emerge the correlation coefficient (R) which will determine the relationship between the variable to be explained (Y_i) and the explanatory variables (X_{i1} and X_{i2}).

D. Experimental technique

The experiment is performed on a physical star topology using a low speed switch with a maximum transmission of 100 Mbps, in order to establish a simple local area network using twisted pair cables with RJ45 connectors as shown in the figure. Fig. 5. But the logical topology or virtually the network works as a decentralized P2P architecture. The configuration of the hardware point of view of the nodes is such that the processor: Intel Core i5, CPU 2.40 GHz, Memory (RAM): 8.00 GB and Storage: 1 TB. These computers will run on Windows 10 Professional 64 bits and SQL Server RDBMS to manage databases and establish connectivity between them.

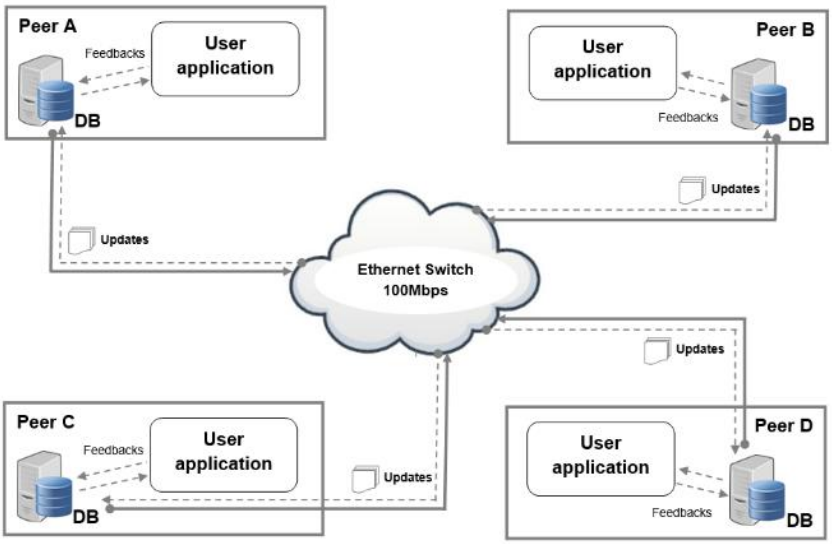


Fig. 5. Protocol of distributed voting eager replication.

The network topology shown above indicates that a peer is composed of hardware and software features. Thus, to make eager replication efficient by using a non-blocking 4PC protocol over a decentralized P2P architecture, the functions have been implemented in a "user application" that can be identified for each peer in Fig. 5. So in the context of this research, the algorithm is translated to C# to result in a prototype, a Windows application with a graphical user interface, as illustrated by the window of Fig. 6, which at the same time presents an application menu in a multiple document interface (MDI) and the replication execution window (synchronization).

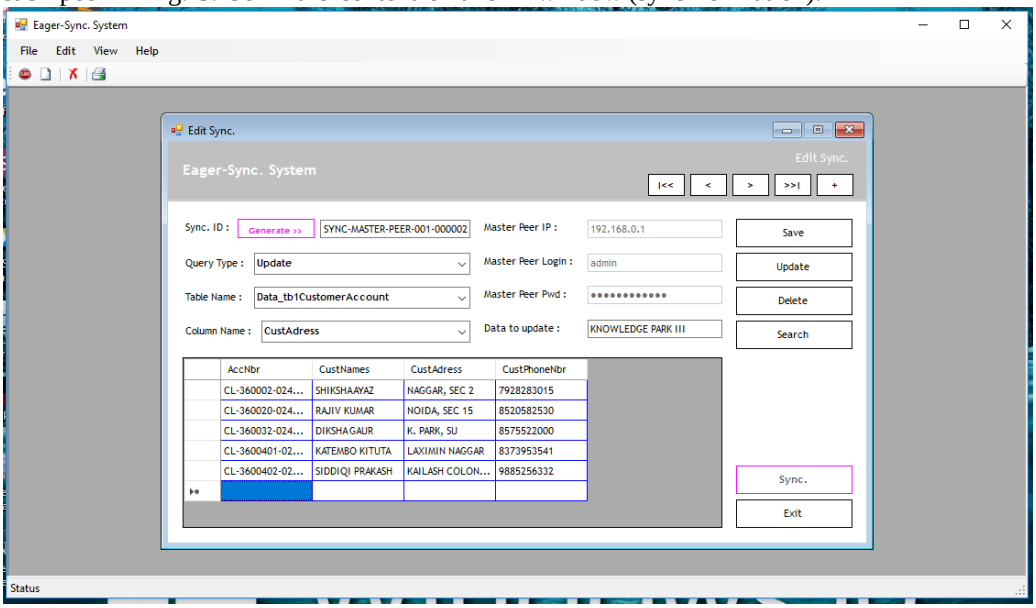


Fig. 6. Eager-Sync. Sytem editor window.

Once the configuration is done as above and after implementing the algorithm to make it a user application running on a decentralized P2P architecture to simulate the new non-blocking protocol, let's analyse the performance and observe the results of the experiment in the next section.

IV. EFFECTIVENESS ANALYSIS

In this section, we will evaluate the proposed solution for synchronous replication on a decentralized P2P network. Our evaluation will explore three major aspects: (i) What is the impact of the factors number of records, number of slave peers (participants) on the execution time of replication transactions? (ii) How many records can be replicate (insert, update, and delete) per second? (iii) What is the effect of varying the number of peers (participants or slaves) on the execution time of replication transactions? Consider a random sample of 12 executions shown in Table I. The test will be performed with the number of records yield from standard data and will take in account all data modification operators that a replication transaction support, namely the insert command, the update command, and the delete command.

Table I. Records number Sample data

Nbr. Obs.	Number of records to replicate
1.	723
2.	900
3.	120
4.	2500
5.	1253
6.	80
7.	3000
8.	5000
9.	450
10.	4860
11.	600
12.	235
Mean	1643.42
Total	19721

A. Experimentation based on two slave peers

The execution time obtained (in sec.) based on the sample data in table I, after experimenting algorithms successively for three data modification operators, namely insert operator, update operator and delete operator, taking into account two slaves peers is presented in the Table II below:

Table II. Result of the experimentation based on two slave peers

Sample numbering		Insert execution time (in Sec.)	Update execution time (in Sec.)	Delete execution time (in Sec.)
Nbr. Obs.	Master Peer			
1.	B	3	3	3
2.	A	4	3	4
3.	C	0	0	1
4.	C	10	10	10
5.	A	5	5	5
6.	A	1	1	0
7.	B	12	13	12
8.	B	19	20	20
9.	A	2	1	2
10.	C	19	20	19
11.	C	2	3	3
12.	B	1	1	1
Mean		6.50	6.67	6.67
Total		78	80	80

On the basis of the factors directly related to the physical schema of the data, namely the number of records and other factors not taken into account in the linear model (such as the number of columns, the data types, etc.), the prediction model of the time of replication is presented as follows: for insert operator depicted in Fig. $7_{(a)}$ $y = 0.0038x + 0.18 + \varepsilon$, for update operator presented in Fig. $7_{(b)}$ $y = 0.0041x - 0.0677 + \varepsilon$ and delete operator shown in Fig. $7_{(c)}$ $y = 0.0039x + 0.2361 + \varepsilon$.

a) b) c)

Fig. 7. Replication execution time: (a) Insertion, (b) Update and (c) Delete operators results from the experimentation based two slave peers.

Anything remaining equal as otherwise, it is necessary to predict the number of record (x) to replicate in one second (y). Thus, algorithms 1, 2 and 3, successively for the data insertion, the updating and the deletion procedures can replicate:

- In insertion procedure (Fig. $7_{(a)}$): $1 = 0.0038x + 0.18 \Rightarrow -0.0038x = -0.82 \Longrightarrow x = 215.78 \Longrightarrow x \approx 216$ records to be insert in 1 second for each eager replication execution. So, as the coefficient of determination R^2 = 0.9974 then the insertion execution time depend on 99.74% of the number of records and as the coefficient of correlation $R = \sqrt{R^2} \Rightarrow R = \sqrt{0.9974} \Rightarrow R = 0.9987$ then the degree of linkage between the insertion execution time and the number of records is 99.87%.
- In updating procedure (Fig. $7_{(b)}$): $1 = 0.0041x - 0.0677 \Rightarrow -0.0041x = -1.0677 \Longrightarrow x = 260.41 \Longrightarrow x \approx 260$ records to be update in 1 second for each real-time replication execution. Thus, as the determination coefficient R^2 = 0.9950 then the updating execution time depend on 99.50% of the number of records and as the correlation coefficient $R = \sqrt{R^2} \Rightarrow R = \sqrt{0.9950} \Rightarrow R = 0.9975$ then the linkage degree between the updating execution time and the number of records is 99.75%.
- In deletion procedure (Fig. $7_{(c)}$):

$1 = 0.0039x + 0.2361 \Rightarrow -0.0039x = -0.7639 \Rightarrow x = 195.87 \Rightarrow x \approx 196$

records to be delete in 1 second for each synchronous replication execution. Hence, as the coefficient of determination $R^2 = 0.9986$ then the delete execution time depend on 99.86% of the number of records and as the coefficient of correlation $R = \sqrt{R^2} \Rightarrow R = \sqrt{0.9986} \Rightarrow R = 0.9993$ then the correlation between the deletion execution time and the number of records is 99.93%.

One of our major concerns is also to analyse the impact of upward variation of number of slave peers. First of all, it should be noted that Table II and the graphs of Fig. 7 above presented successively the execution time obtained after syncing records according to the sample presented in Table I and the trend curves in Fig. $7_{(a)}$, $7_{(b)}$ and $7_{(c)}$ which have been used to predict the number of records to replicate (insert, update and delete) in one second. However, let's try to vary the number of slave peers upwards and observe once again the execution time in real-time for synchronous replication.

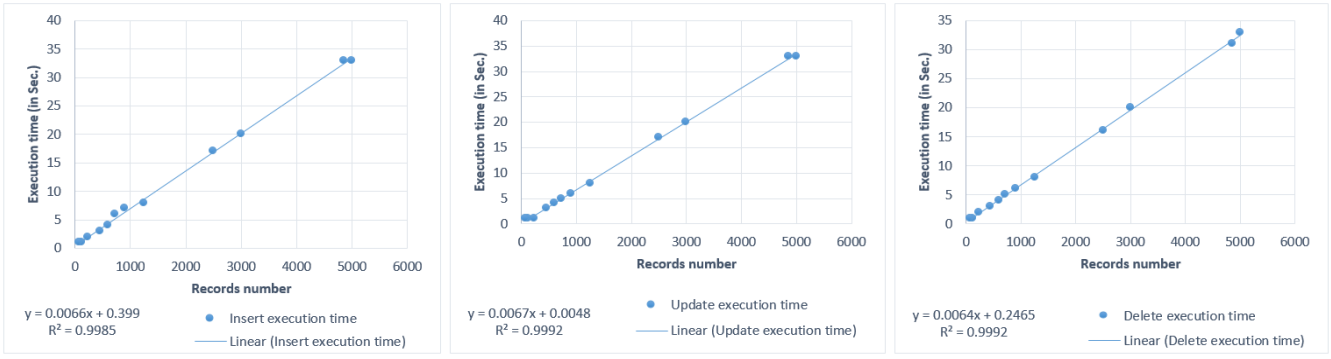
B. Experimentation based on three slave peers

After varying the number of slave peers that come from two (2) to three (3) slave peers, the execution time obtained (in sec.) is presented successively in Table III and in graphs of the Fig. 8. On the basis of the data sample of Table I, after successively experimenting algorithms of three data modification operators, namely insert operator, update operator and delete operator, the result is presented here below.

Table III. Result of the experimentation based three slave peers

Sample numbering		Insert execution time (in Sec.)	Update execution time (in Sec.)	Delete execution time (in Sec.)
Nbr. Obs.	Master Peer			
1.	B	6	5	5
2.	A	7	6	6
3.	C	1	1	1
4.	C	17	17	16
5.	D	8	8	8
6.	A	1	1	1
7.	B	20	20	20
8.	D	33	33	33
9.	A	3	3	3
10.	C	33	33	31
11.	D	4	4	4
12.	B	2	1	2
Mean		11.25	11.00	10.83
Total		135	132	130

The variation of the factor number of slave peers (x_2) obviously has an effect on the execution time (y); and this can be observed based on averages of execution times that have already experienced the increase, according to the Table III comparatively to those of Table II and curves of graphs depicted in the Fig. 9. However, our analysis will not stop at this level, it is appropriate to study the regression models to estimate the number of records (x_1) which can be replicated in one second (y): for insert operator shown in Fig. $8_{(a)}$ $y = 0.0066x + 0.399 + \varepsilon$, for update operator presented in Fig. $8_{(b)}$ $y = 0.0067x + 0.0048 + \varepsilon$ and for delete operator depicted in Fig. $8_{(c)}$ $y = 0.0064x + 0.2465 + \varepsilon$.



a) b) c)

Fig. 8. Replication execution time: (a) Insertion, (b) Update and (c) Delete operators results from the experimentation based three slave peers.

When the number of slave peers know the increase from two (2) to three (3) slave peers, in 1 second, the prediction of the execution time (y), during what our algorithms can successively insert, update and delete records (x), is calculated from the following way:

- In insertion procedure (Fig. $8_{(a)}$) : $1 = 0.0066x + 0.399 \Rightarrow -0.0066x = -0.601 \Rightarrow x = 91.06 \Rightarrow x \approx 91$

  records to be insert in 1 second for each real-time replication execution. So, as the determination coefficient $R^2 = 0.9985$ then the dependence degree of insertion execution time compared to the number of records is 99.85% and as the coefficient of correlation $R = \sqrt{R^2} \Rightarrow R = \sqrt{0.9985} \Rightarrow R = 0.9992$ then the degree of linking between the insertion execution time and the number of records is 99.92%.
- In updating procedure (Fig. $8_{(b)}$):

$1 = 0.0067x + 0.0048 \Rightarrow -0.0067x = -0.9933 \Rightarrow$
$x = 148.25 \Rightarrow x \approx 148$

records to be updated in 1 second for each eager replication execution. Thus as the coefficient of determination $R^2 = 0.9992$ then the update execution time depend on 99.92% of the number of records and as the coefficient of correlation $R = \sqrt{R^2} \Rightarrow R = \sqrt{0.9992} \Rightarrow R = 0.9996$ then the degree of the relation between the update execution time and the number of records is 99.96%.

- In deletion procedure (Fig. $8_{(c)}$):

$1 = 0.0064x + 0.2465 \Rightarrow -0.0064x = -0.7535 \Rightarrow$
$x = 117.73 \Rightarrow x \approx 118$

records to be delete in 1 second for each synchronous replication execution. So, as the coefficient of determination $R^2 = 0.9992$ then the delete execution time depend on 99.92% of the number of records and as the coefficient of correlation $R = \sqrt{R^2} \Rightarrow R = \sqrt{0.9992} \Rightarrow R = 0.9996$ then the correlation between the insertion execution time and the number of records is 99.96%.

When looking at the graphs of Figs. 7 and 8, we realize that trend lines are increasing; this means that each time the number of records to replicate (to insert, update and delete) is increasing then the execution time also increases. In all cases it has been noticed that the rate of this growth is above 99%, i.e. the execution time depends on the number of records at around 100% and therefore the factor number of records and execution time correlate to about 100%.

a) b) c)

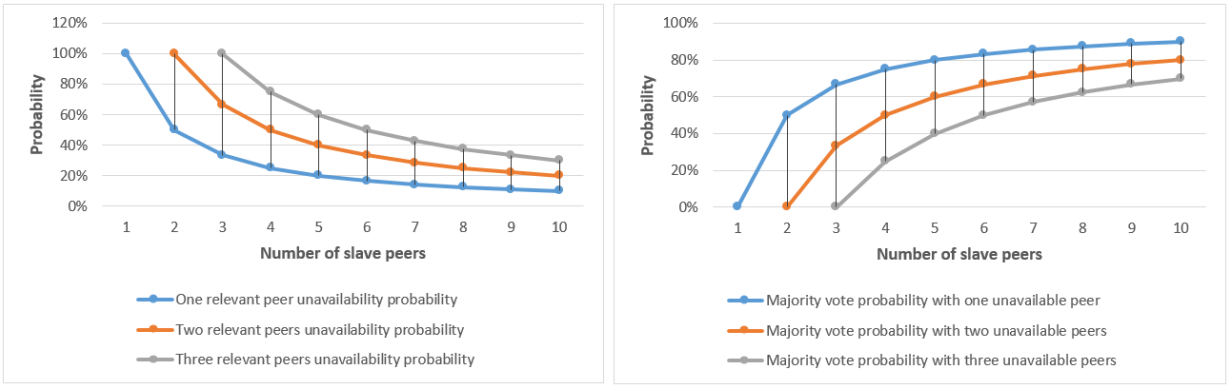
Fig. 9. Effectiveness of replication: (a) Insertion, (b) Update and (c) Delete operators based two slave peers vs. three slave peers.

In the same way, the impact due to the upward variation of the number of peers during synchronous replication on a P2P topology is presented in the graphs of Fig. 9 above, successively for the insertion operator (Fig. $9_{(a)}$), the update operator (Fig. $9_{(b)}$) and the delete operator (Fig. $9_{(c)}$). The execution time gap of data modification operators with three slave peers far exceeds that with two slave peers. This causes the (orange) curve of the execution time with three slave pairs to be beyond the (blue) curve of the execution time with two slave pairs.

The two execution scenarios that were the subject of this experiment were performed without taking into account a peer that may not have been available during the execution of the transaction. So the 4th phase of the algorithm was not taken into account in the execution time. The phase 4 of the algorithm, referring to the recovery management of unavailable peers, consists first of queuing or locally backing up the data, but also to recover those peers when they become available again. However, the algorithms 1, 2 and 3 run according to three alternatives:

(1) If the vote majority is greater than or equal to the quorum and if the vote majority is 100% then all peers are updated at the same time; well, this is the case for which the above analysis has just been done;
(2) Then, if the vote majority is strictly inferior to the quorum, the replication transaction is rejected (Abort);
(3) However, at the end if the majority vote is greater than or equal to the quorum but less than 100%, then at least one peer is unavailable. So data must be kept for later reflection to peers which did not vote the transaction on the performing time.

Now, it is this last case that draws our attention especially since there is a certain time needed to queue data and also to later recover peers which were unavailable when they become available again. But on a P2P network, the probability of having x_{i22} slave pair absent during a replication transaction performing is x_{i22} / x_{i21}, where x_{i21} is the total number of slave peers and x_{i22} the number of unavailable peers. And according to the graph of Fig. $10_{(a)}$, this probability decreases as the number of slave peers increases. Moreover, based on the majority consensus method, the possibility of a majority vote when a peer is unavailable is given by $(x_{i21} - x_{i22}) / x_{i21}$, where x_{i21} is the total number of peers and x_{i22} the number of unavailable peers. Thus, starting from the first term $x_{i21} - x_{i22}$, we can observe in the graph of the Fig. $10_{(b)}$ from how many unavailable slave peers x_{i22} in the total number of slave peers x_{i21} can the transaction be rejected, the quorum being 60%.



a) b)

Fig. 10. Probability: (a) of a relevent slave peer unvailability (b) of majority vote.

Taking in consideration the quorum of 60%, according to the Fig. $10_{(b)}$ it is clear that with one (1) or two (2) slave peers, it's enough that one (1) of them be unavailable so that the transaction be rejected because in this case when one (1) peer is absent, the vote majority is 0% and for two (2) peers it is 50%, all less than 60% the quorum... But from three (3) peers, even if one peer is unavailable, the transaction is accepted and performed to available peers and updates are queued for late updating to be performed to unavailable peers when they become present again.

However, starting from what we already know about P2P networks today, the number of peers is still considerable in the sense that the probability of not having x_{i22} available peer is still conceivable. This is why the need for managing unavailable peer recovery is imperative. Thus, one realizes that the total execution time (y_i) will be the sum of the time (y_{i1}) of the alternative (1), to perform the transaction to the majority of available slave peers (x_{i21}) and the time (y_{i2}) of the local backup of data to be sent later to a certain number of slave peers unavailable (x_{i22}). But we already know the time (y_{i1}), analysed previously in the two scenarios of the experimentation (A) and (B). So here we have to find the time (y_{i2}), the time of queuing data of the unavailable slave peers. In the same way we have to find (y'_i), the recovery time of a slave pair by the data already stored locally and waiting that the concerned peers be available.

running three data modification operators namely the insertion operator, the update operator and the delete operator, the queuing time when there is an unavailable peer as well as the recovering time when it becomes available again is given in Table IV as follow:

C. Experimentation taking into account the recovery management

Considering always the sample of the number of records in Table I, to be queued and recovered successively after

Table IV. The result of queuing and recovering data based the experimentation with a slave peer that was unavailable for a moment and then available

Sample numbering		Data queuing time (in Sec.)			Recovery execution time (in Sec.)		
Nbr. Obs.	Master Peer	Inserted data	Updated data	Deleted data	Insert	Update	Delete
1.	B	0	0	0	3	3	2
2.	A	0	0	0	2	3	2
3.	C	0	0	0	1	1	0
4.	C	1	0	1	7	7	7
5.	A	1	0	0	4	3	3
6.	A	0	0	0	0	0	1
7.	B	1	1	1	9	8	7
8.	B	2	1	2	14	15	13
9.	A	0	1	0	2	1	1
10.	C	1	2	1	14	13	12
11.	C	0	0	0	2	1	1
12.	B	0	0	0	0	1	1
Mean		0.50	0.42	0.42	4.83	4.67	4.17
Total		6	5	5	58	56	50

These data originate from the experimentation which measured the practical temporal complexity of the 4th phase of the algorithms 1, 2 and 3 in which the data queuing function (algorithm 4) is performed and the recovering function (algorithm 5) which executes a transaction of recovery of peers which have been unavailable for a moment. Thus, once again, the linear regression models are presented in order to derive approximately the number of records (x_{i1}) that can be successively queued and applied in order to recover a slave peer (x_{i2}), that becomes available again, in a certain time (y_{i2} and y'_i): for queuing inserted data, operator shown in Fig. $11_{(a)}$ $y = 0.0003x - 0.0518 + \varepsilon$, for queuing updated data, operator depicted in Fig. $11_{(b)}$ $y = 0.0003x - 0.0507 + \varepsilon$ and for queuing deleted data, operator presented in Fig. $11_{(c)}$ $y = 0.0003x - 0.1535 + \varepsilon$ and consecutively for recovering by insertion, operator shown in Fig. $11_{(d)}$ $y = 0.0028x + 0.2121 + \varepsilon$, for recovering by updating, operator presented in Fig. $11_{(e)}$ $y = 0.0028x + 0.0802 + \varepsilon$ and for recovering by deletion, operator depicted in Fig. $11_{(f)}$ $y = 0.0025x + 0.0339 + \varepsilon$.

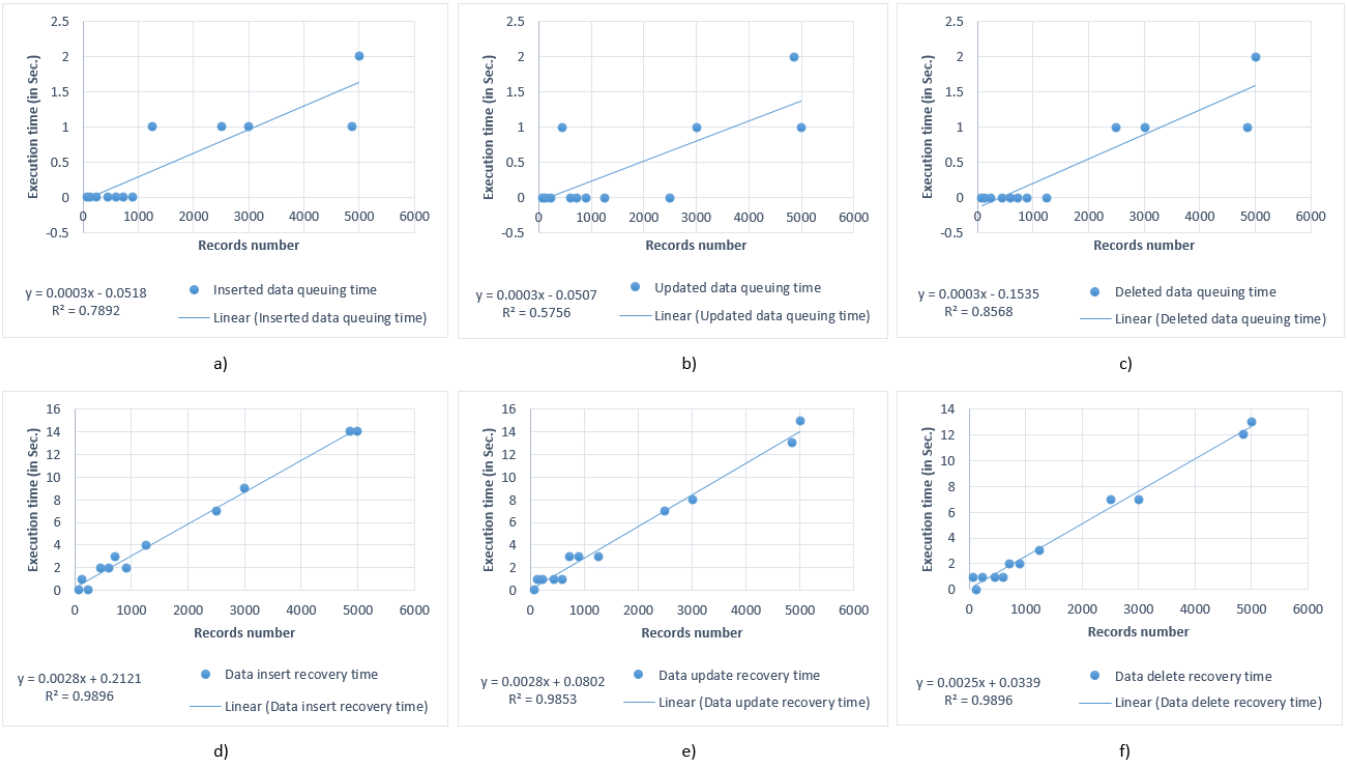


Fig. 11. Execution time for queuing (a) Inserted data, (b) Updated data and (c) Deleted data and for recovering (d) by insert, (e) by update and (f) by delete operators results from the experimentation based one unvailable slave peer.

When considering that a slave peer had been unavailable when the transaction was running, the queuing as well as the recovering of modified records (x) and applying them on the concerned slave peer when it becomes available again can be predicted in 1 second (y) as follows:

- For insert operator
  - ✓ In queuing procedure (Fig. $11_{(a)}$): $1 = 0.0003x - 0.0518 \Rightarrow -0.0003x = -1.0518 \Rightarrow x = 3506$ records to be queued in 1 second for each queuing run. So, as the determination coefficient $R^2 = 0.7892$ then the dependency degree of inserted data queuing time compared to the number of records is 78.92% and as the coefficient of correlation $R = \sqrt{R^2} \Rightarrow R = \sqrt{0.7892} \Rightarrow R = 0.8884$ then the degree of linking between the inserted data queuing execution time and the number of records is 88.84%.
  - ✓ In recovering procedure (Fig. $11_{(d)}$): $1 = 0.0028x + 0.2121 \Rightarrow -0.0028x = -0.07879 \Rightarrow x = 281.4 \Rightarrow x \approx 281$ records to be recovered in 1 second for each recovering execution. The coefficient of determination R^2 being of 0.9896 then inserted data recovering time depend on 98.96% of the number of records and as the correlation coefficient $R = \sqrt{R^2} \Rightarrow R = \sqrt{0.9896} \Rightarrow R = 0.9948$ then the degree of relation between the inserted data recovering time and the number of records is 99.48%.

- For update operator
  - ✓ In queuing procedure (Fig. $11_{(b)}$): $1 = 0.0003x - 0.0507 \Rightarrow -0.0003x = -1.0507 \Rightarrow x = 3502.33 \Rightarrow x \approx 3502$ records to be queued in 1 second for each queuing execution. Thus as the coefficient of determination $R^2 = 0.5756$ then the updated data queuing time depend on 57.56% of the number of records and as the coefficient of correlation $R = \sqrt{R^2} \Rightarrow R = \sqrt{0.5756} \Rightarrow R = 0.7587$ then the degree of the relation between the updated data queuing time and the number of records is 75.87%.
  - ✓ In recovering procedure (Fig. $11_{(e)}$): $1 = 0.0028x + 0.0802 \Rightarrow -0.0028x = -0.9198 \Rightarrow x = 328.5 \Rightarrow x \approx 328$ records to be recovered in 1 second for each recovering run. As the determination coefficient $R^2 = 0.9853$ then the updated data recovering time depend on 98.53% of the number of records and as the coefficient of correlation $R = \sqrt{R^2} \Rightarrow R = \sqrt{0.9853} \Rightarrow R = 0.9926$ then the degree of relation between the updated data recovering time and the number of records is 99.26%.
- For delete operator:
  - ✓ In queuing procedure (Fig. $11_{(c)}$): $1 = 0.0003x - 0.1535 \Rightarrow -0.0003x = -1.1535 \Rightarrow x = 3845$ records to be queued in 1 second in every queuing execution. So, as the coefficient of determination $R^2 = 0.8568$ then the deleted data queuing time depend on 85.68% of the number of records and as the coefficient of correlation $R = \sqrt{R^2} \Rightarrow R = \sqrt{0.8568} \Rightarrow R = 0.9256$ then the correlation between the deleted data queuing time and the number of records is 92.56%.
  - ✓ In recovering procedure (Fig. $11_{(f)}$): $1 = 0.0025x + 0.0339 \Rightarrow -0.0025x = -0.9661 \Rightarrow x = 386.44 \Rightarrow x \approx 386$ records to be recovered in 1 second for every run. As the coefficient of determination $R^2 = 0.9896$ then the deleted data recovering time depend on 98.96% of the number of records and as the coefficient of correlation $R = \sqrt{R^2} \Rightarrow R = \sqrt{0.9896} \Rightarrow R = 0.9948$ then the degree of relation between the deleted data recovering time and the number of records is 98.96%.

Starting from these regression models of Fig. 11, successively illustrating and predicting the queuing time of the data and the time of recovery and the application of updates having been put on hold to the slave peers having been unavailable, our first observation is that queuing takes a minimal time compared to the recovering. Referring to Table IV above, however, we can see that the queuing time of the data of an unavailable slave peer is worth about 10% of the time of its recovery when it becomes available again. This is because the queuing of the data is done locally, i.e. on the same node, so the data traffic from the main memory to the auxiliary memory which keeps the database while the recovery consists of joining a remote peer by the network architecture through its DBMS and then execute queries for data modification on its database stored in its auxiliary memory. Thus, in the total cost of the replication transaction, this time does not represent much, a little like that is illustrated in the graphs of Fig. 12. (orange bars) and therefore the number of records to queue in 1 second is very considerable, beyond 3500 records for each data modification operator.

The fact that the factors number of records and queuing time are not too closely related, correlation of less than 95% and degree of dependence and / or independence of less than 90%, is due to non-estimation of some observations which were directly reduced to 0 second while they could perhaps be between 0 to 1 second. So as the minimum time unit we considered is the second, such observations took 0 second directly. But, for the recovery of data, between the factors number of records and recovery time of a slave peer, there is a pure and perfect correlation, i.e. about 100% and the degree of dependency of the recovering time of the number of records is beyond 95%.

In the second place, our observation concerns the recovering time of data to be applied to a slave peer that has been absent momentarily. This time is very considerable because it exceeds the half of the replication execution time with two slave peers as can be seen by the comparison of the totals of Tables II and IV (for recovery execution time), but also by the regression models which predict the number of records to replicate with two slave peers and the number of records to be recovered by a slave peer that becomes available again. Here below, this same comparison will be graphically illustrated in Figs. $12_{(a)}$, $_{(b)}$ and $_{(c)}$ (the blue and yellow bars), respectively for the replication execution time and the recovering time. In the same way this time has been compared to the replication execution time with three slave peers as shown in Figs. $12_{(d)}$, $_{(e)}$ and $_{(f)}$, but not exceeding the half of the replication execution time with three slave peers.

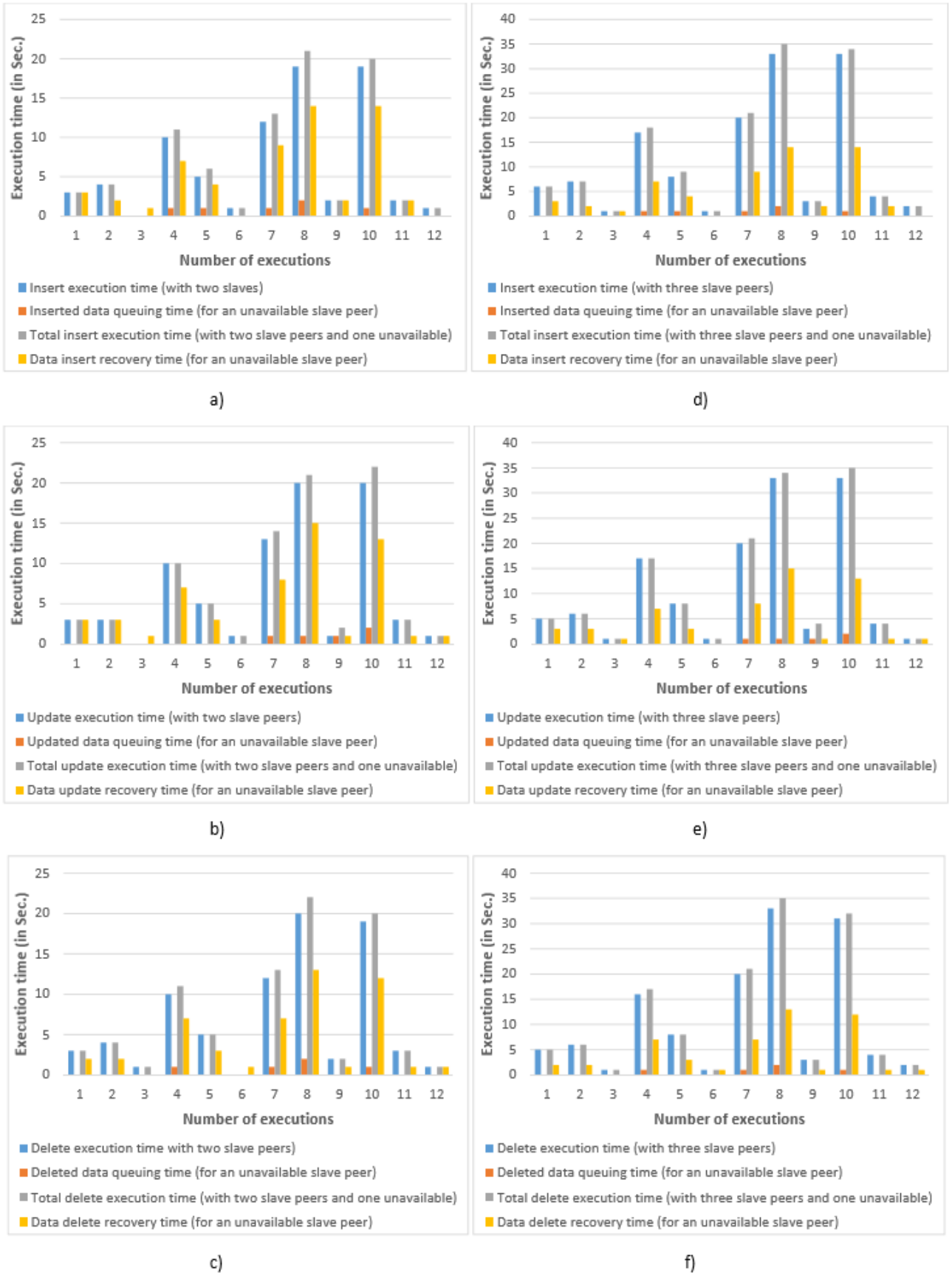


Fig. 12. Effectiveness of replication and recovery management : (a) Insertion, (b) Update and (c) Delete operators based two slave peers vs. (d) Insertion, (e) Update and (f) Delete operators based three slave peers.

While the experiment was performed with one (1) unavailable peer that needed to be recovered, so we can expect that recovering a certain number of slave peers (x_{i22}) which was unavailable costs more in terms of time (y'_i) than replicating the data in real time (y_i) with an available peer. Hence the need for each time independently implemented the recovery procedure (algorithm 5) of the replication procedure (algorithms 1, 2 and 3) to balance the execution time. Starting from all the above and having already in full all the necessary information upcoming from all cases and scenarios of experimentation, the Table V below will present the summary of this result.

Table V. The condensed result

Experimental scenarios	Transaction	Operator	Model	R^2	R	Prediction (to 1 Sec.)
A. Experimentation based on two slave peers	Replication	Insert	$y = 0.0038x + 0.18 + \varepsilon$	99.74%	99.87%	216 records
		Update	$y = 0.0041x - 0.0677 + \varepsilon$	99.50%	99.75%	260 records
		Delete	$y = 0.0039x + 0.2361 + \varepsilon$	99.86%	99.93%	196 records
B. Experimentation based on three slave peers	Replication	Insert	$y = 0.0066x + 0.399 + \varepsilon$	99.85%	99.92%	91 records
		Update	$y = 0.0067x + 0.0048 + \varepsilon$	99.92%	99.96%	148 records
		Delete	$y = 0.0064x + 0.2465 + \varepsilon$	99.92%	99.96%	118 records
C. Experimentation taking into account the recovery management	Queuing	Insert	$y = 0.0003x - 0.0518 + \varepsilon$	78.92%	88.84%	3506 records
		Update	$y = 0.0003x - 0.0507 + \varepsilon$	57.56%	75.876%	3502 records
		Delete	$y = 0.0003x - 0.1535 + \varepsilon$	85.68%	92.56%	3845 records
	Recovering	Insert	$y = 0.0028x + 0.2121 + \varepsilon$	98.96%	99.48%	281 records
		Update	$y = 0.0028x + 0.0802 + \varepsilon$	98.53%	99.26%	328 records
		Delete	$y = 0.0025x + 0.0339 + \varepsilon$	98.96%	99.48%	386 records

However, alluding to the P2P network theory, with several peers on the network, having an unavailable peer is still always probable. Therefore, for an eager P2P replication, taking into account a peer that may has been unavailable during the transaction running, the computation of the replication transaction time must imperatively include the time of the data queuing, even if minimal as we have just seen. Hence, the relation $y_i = y_{i1} + y_{i2}$, where y_i is the total time of replication, y_{i1} the time of replication without data queuing and y_{i2} the data queuing time. There is no relation between the total replication time y_i and the recovering time y'_i because these two transactions run in different times. Thus, with this in mind, we accept our alternative hypothesis (H_1) according to what: "independent factors, i.e. the number of records (x_{i1}) and the number of slave peers (x_{i2}) and other factors (ε) like number of columns per table, data types columns, etc. are significant predictors of the execution time (y_i and/or y'_i)" and so we reject the null hypothesis (H_0).

This level of significance is proved by the fact that, apart from queuing the data, the replication transactions successively experimented with two slave peers and with three slave peers as well as the recovery transaction have coefficients of determination (R^2) above 95% which is an acceptable threshold of significance. In this logic, it is implied that the modifications made to the independent factors (x_{i1}, x_{i2} and ε) influence more than 95% the dependent factor (y_i and/or y'_i). In the same way, the correlation coefficient (R) being more than 99%, this attests that the linking between the independent factors (x_{i1}, x_{i2} and ε) and the dependent factor (y_i and/or y'_i) is pure and perfect. Although the queuing of data has not scrupulously verified all these properties, we will not be much upset because it runs after the replication sub-transaction commitment and more it happens locally. Besides the prediction has substantiated that it is efficient because there is a very considerable number of records that can be queuing in one second.

Thus, in a general way, this result is just enough to prove that this proposed approach is efficient because in one second, a very elemental temporal unit to which we can attribute the event "click of a mouse" and then directly observe the effect, a considerable number of records can be replicated eagerly, queued or recovered by a certain number of slave peers. The last column of Table V above illustrates this clearly. And we consider that this performance is not negligible especially in the reality users, here we are referring to developers who will reuse these functions (this algorithm), call regularly and successively to insert, update and delete the data. So as often in a database application, manipulation does not often involve a large amount of data but rather basic data to make a database in the long period, up to a certain level the performance will remain verifiable.

Nevertheless, this performance being appreciable by the optimization of the execution time, it is thus guided by the two flagship variables namely the number of records (x_{i1}) and the number of slave peers (x_{i2}). Still in the interest of further optimizing it, future investigations will have to focus on minimizing the number of slave peers (x_{i2}) because when we have varied x_{i2} from 2 to 3, there has been an extravagant execution time. This phenomenon was well clarified by comparing the observations as well as the averages of Tables II and III, and the graphs of Figs. 9 and 12, successively the blue and orange curves in graphs of Figs. $9_{(a)}$, $9_{(b)}$ and $9_{(c)}$, for the execution time with two slave peers and three slave peers and the gray bars of graphs in Figs. $12_{(a)}$, $12_{(b)}$ and $12_{(c)}$ for the total execution time with two slave peers and graphs of Figs. $12_{(d)}$, $12_{(e)}$ and $12_{(f)}$ for the total execution time with three slave peers. For this same cause we do not yet find an optimization track with the number of records factor (x_{i1}) since the amount of data to be handled remains the choice of users.

But as for the distributed environment based on the P2P architecture, the optimization of the large number of peers (x_{i2}), which is valued at hundreds or even thousands [6], deserves a particular analysis.

This is how we plan for the future to conduct a new investigation in the context of improving the performance of this proposed algorithm. Thus, the thought will revolve around the synchronization algorithm of replicated databases on a decentralized P2P architecture with super-nodes or super-pairs [4], [5] belonging to peer groups in order to reduce the time of executing transactions and achieving load balancing when transmitting data [22], [23].

V. CONCLUSION

The use of an atomic commit protocol of a transaction (typically 2PC) on a P2P network to perform eager replication don't reflect the reliability of replicas, especially since it is based on the ROWA standard. Moreover, a P2P network requiring the multi-master approach being given the equality between peers is still candidate for the conflict of transactions emerge concurrently from different peers to update the same replica. Both of these problems are likely to cause a perpetual abortion of transactions. To address these problems, the combination of nested transaction technique and distributed voting has introduced a new Four-Phase-Commit (4PC) protocol that validates transactions with the majority of available peers and recovers those that are not available when they become available, hence the new ROWA-A principle. After the implementation of the new algorithm in C#, the statistical technique as well as the experimentation made it possible to study its effectiveness. This study which has been based on the number of records, the number of slave peers and other independent factors and the execution time as the dependent factor, revealed that theses independents factors influence the execution time to more than 95% and their degree of linking is pure and perfect because it measures more than 99%.

Finally, it was found that the new algorithm produces an acceptable efficiency because in one second the replication of a large number of records can be performed, a considerable quantity of data can be queued in order to be reflected later to peers which have been temporarily unavailable when they become available again.

ACKNOWLEDGMENT

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.

REFERENCES

1. Nicoleta-Magdalena, I. C. (2011). The replication technology in e-learning systems, *Procedia-Social and Behavioral Sciences*, 28(2011), pp. 231-235.
2. Maarten, V. S. and Andrew, T. (2016). A brief introduction to distributed systems, In Distributed Systems, Principles and Paradigms (2nd edition), *Netherlands, Publisher: Springer*.
3. Özsu, M., T., and Valduriez, P. (2011). Principles of Distributed Database Systems (3rd Ed.), New York, United State: *Springer Science+Business Media, LLC*.
4. Fatos, X., Kolici, V., Potlog, A., Spaho, E., Barolli, L., and Takizawa, M. (2012). Data Replication in P2P Collaborative Systems, *IEEE Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing Victoria, BC, Canada*, pp. 49-57.
5. Spaho, E., Barolli, A., Fatos, X., and Barolli, L. (2015). P2P Data Replication: Techniques and Applications, In: Xhafa F., Barolli L., Barolli A., Papajorgji P. (eds) Modeling and Processing for Next-Generation Big-Data Technologies. Modeling and Optimization in Science and Technologies, *Publisher: Springer*, Vol. 4, pp. 145-166.
6. Vu, Q. H., Lupu, M., and Ooi, B. C. (2009). Peer-to-peer computing: Principles and applications. Berlin, Heidelberg, Germany: *Springer Science & Business Media*.
7. Kituta, K., Kant, S., Agarwal, R. (2018). Analysis of database replication protocols, *International Journal of Latest Trends in Engineering and Technology*, Special Issue ICRMR-2018, pp. 075-083.
8. Zhou, T. and Wei. Y. (2013). Database Replication Technology having high Consistency Requirements, *IEEE 3th International Conference on Information Science and Technology March 23-25, 2013: Yangzhou, Jiangsu, China*, pp. 793- 797.
9. Kituta, K., Kant, S. Agarwal, R. (2019). A systematic review on distributed databases systems and their techniques, *Journal of Theoretical and Applied Information Technology*, 96(1), pp. 236-266.
10. Souri, A., Pashazadeh, S., and Navin, A. H. (2014). Consistency of data replication protocols in database systems: A review, *International Journal on Information Theory (IJIT)*, 3(4), pp. 19-32.
11. Srivastava, A., Shankar, U. and Tiwari, S. K. (2012). Transaction management in homogenous distributed real-time replicated database systems, *International Journal of Advanced Research in Computer Science and Software Engineering*. 2(6), pp. 190-196.
12. Meroufela, B. and Belalem, G. (2013). Managing Data Replication and Placement Based on Availability, *Procedia - AASRI Conference on Parallel and Distributed Computing Systems: Elsivier*. Vol. 5, pp. 147 – 155.
13. Sinde, V. and Aware, P. (2016). Concurrency control in distributed database systems, *International Journal for Research in Engineering Application & Management (IJREAM)*, 1(10), pp. 1-5.
14. Kaur, M. and Kaur, H. (2013). Concurrency Control in Distributed Database System, *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(7), pp. 1443-1447.
15. Kumar, S., Sharma, A. and Swaroop, V. (2011). Replication: Analysis & Tackle in Distributed Real Time Database System, *International Journal of Recent Trends in Electrical & Electronics Engg.*, 1(2), pp. 43-48.
16. Wiesmann, M., Pedone, F., Schiper, A., Kemme, B. and Alonso, G. (2000). Understanding replication in databases and distributed systems, *Proceedings 20th IEEE International Conference on Distributed Computing Systems*, pp. 464-474.
17. Thomas, H. C., Charles, E. L., Ronald, L. R., and Clifford, S. (2012). Introduction to Algorithms (4th Ed.). London, England: *The MIT Press*.
18. Kothari, C., R., and Garg, G. (2014). Research methodology methods and techniques (3rd Ed.). New-Delhi, India: *House, Ed., M.P Printers*.

19. Sheikh, H. J., Rimiru, R. M. and Kimwele, M. W. (2017). Alternative Model to Overcoming Two Phase Commit Blocking Problem, *International Journal of Computer (IJC)*, 26(1), pp. 71-88.
20. Abuya, T. K., Rimiru, R. M. and Cheruiyot, W. K. (2015). Clustering Algorithm in Two-Phase Commit Protocol for Optimizing Distributed Transaction Failure, *International Journal of Computer Science and Mobile Computing*, 4(3), pp. 97-106.
21. Kumar, N., Sahoo, L. and Kumar, A. (2014). Design and implementation of Three Phase Commit Protocol (3PC) algorithm, *IEEE International Conference on Reliability Optimization and Information Technology (ICROIT)*, pp. 116-120.
22. Santana, M., Enrique, J. and Francesc, D. (2015). Evaluation of database replication techniques for cloud systems. *Computing and Informatics*, Vol. 34, pp. 973-995.
23. Hababeh, I. (2010). Improving network systems performance by clustering distributed database sites, Journal of Supercomputing, *Publisher: Springer Science+Business Media, LLC*, 59(2012), pp. 249–267.

AUTHORS PROFILE

Mr. Katembo Kituta Ezéchiel obtained her MIT in the Department of Management and IT at the Faculty of Economics and Management of the Adventist University of Lukanga (Congo-Kinshasa). He was appointed Graduate Assistant and Assistant Lecture, simultaneously, at Butembo's Institute of Building and Public Works (IBTP-Butembo / Congo-Kinshasa). Registered in the doctoral program of Sharda University (India), School of Engineering and Technology, Department of Computer Science and Engineering, since July 2016, he presented his research project in Distributed Database Systems (DDS) as research area, before being enrolled as a fulltime Ph. D. Student on the same program, since July 2017. Apart from this aforementioned field, his interests are also focused on DDS for wireless sensor networks, cloud computing and software project management. He is author of one journal paper and two international conference papers.

Dr. Shri Kant has received his Ph. D. in applied mathematics "Special Functions" from applied mathematics departments, institute of technology BHU in 1981. Thereafter he served M/O Defence in various capacity and involved in Research and Development and Academic activity for about thirty-five (35) years in Armed Forces Headquarter, Joint Cipher Bureau (JCB) and Scientific Analysis Group (SAG) of DRDO. His areas of interest are Special Functions, Cryptology, Pattern Recognition, Cluster Analysis, Soft Computing Model, Machine Learning and Data Mining. He has published more than 70 research papers in international and national journals and conferences. He has also published several classified technical reports for internal consumption by the departments. He has eight copyright to his credit for about 182 algorithms grouped under eight categories. He has guided more than 30 students of MCA, M. Phil, M. Tech projects and Ph. D. thesis.

Dr. Ruchi Agarwal is a Ph. D. (Technology) from Birla Institute of Technology (BIT), Mesra, Ranchi in the field of Data Analytics. She has more than 16 years of Academic and Industrial experience. Currently she is working with JIMS Engineering Management Technical Campus, Greater Noida. She has published various papers in international journals and conference proceedings. She has guided various B Tech and M Tech level projects. She is guiding Ph.D. students in the area of Big Data Analytics. Her research interest areas are Big Data Analytics, Data Mining, and Customer Analytics.