Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 4, Issue. 3, March 2015, pg.97 – 106

RESEARCH ARTICLE

A Clustering Algorithm in Two-Phase Commit Protocol for Optimizing Distributed Transaction Failure

[1]Teresa K. Abuya*, **[2]Dr. Richard M. Rimiru**, PhD, **[3]Dr. Cheruiyot W.K**, PhD

[1]Computer Science, Kisii University, Kenya

[2,3]Computer Science, Jomo Kenyatta, University of Agriculture & Technology, Kenya

[1] tkwambokaa@gmail.com, [2] rimirurm@gmail.com, [3] wilchery68@gmail.com

Abstract: *The Two-Phase Commit Protocol (2PC) is a distributed algorithm used in computer networks and distributed database systems. It is used when a simultaneous data update should be applied within a distributed database. In this protocol, one node acts as the coordinator, which is also called master and all the other nodes in the network are called participants or slaves. The important issue in transaction management is that if a database was in a consistent state prior to the initiation of a transaction, then the database should return to a consistent state after the transaction is completed. This should be done irrespective of the fact that transactions were successfully executed simultaneously or there were failures during execution. To optimize failure in distributed transactions, a clustering algorithm is simulated in Two Phase Commit(2PC) protocol to demonstrate how coordinator and site failure is minimized while maintaining atomicity and consistency property of transactions. This paper compares the performance of the transaction clustering algorithm with the current 2PC protocol. The Possible failure cases are identified and created to show how it responds to different failure scenarios and recovery. The simulation algorithm was developed using Jcreator with mySQL acting as a back end data manager, the Bitronix transaction manager which is a simple but complete implementation of Java applications whose goal is to provide a fully working transaction manager that has useful error reporting and logging methods which makes it easier to know when an error occurs. Results obtained indicated that by using the proposed algorithm, transaction failures associated with 2PC can be reduced.*

Keywords: *2PC, transaction management, Clustering algorithm, distributed transactions, transaction failure.*

I. Introduction

Distributed database systems pose different problems when accessing distributed data. An important issue in transaction management is that, if a database was in a consistent state prior to the initiation of a transaction, then the database should return to a consistent state after the transaction is completed[1]. This should be done irrespective of the fact that transactions were successfully executed simultaneously or there were failures during execution. Transactions communicate with transaction Managers(TM) and TMs communicate with Data Managers (DM) and DMs manage the data being committed [10].

A Distributed database system (DDSs) implements a transaction commit protocol to ensure transaction atomicity. Several sites need to update their databases with the same information. One client requests information to be uploaded and a site receive the request and start a procedure where he becomes the coordinator of this request. The other sites become participants of the particular request.

Some form of control is necessary to ensure that concurrent execution of transactions in a distributed environment does not jeopardize the integrity of the system as well as its data consistency. The performance factor of concurrency control algorithms depends on systems throughput and transaction response time. Four cost factors influence the performance: local processing, inter-site communication, transaction restarts and transaction blocking [7].

Concurrency control uses two types of commit protocols which include the Two Phase Commit (2PC) and Three-Phase Commit (3PC) protocols.2PC protocol is of prime importance to many distributed transaction processing applications used by financial institutions and other applications that fall within the spectrum of enterprise computing. These types of applications are increasingly being used to harness the availability of commodity processing power scattered in many sites of medium to large scale organizations. Only two phases are executed in 2PC .The prepare and commit phase but it has a blocking disadvantage in which either the coordinator or some participating site is blocked.

3PC protocol was introduced as a remedy to the blocking disadvantage of 2PC protocol. It introduced an extra phase called the pre-commit phase which ensured the non-blocking property of this protocol. Although 3PC protocol overcomes blocking problem, it involves an additional round of message transmission to achieve non-blocking property which further reduces system performance as compared to 2PC protocol[11].To solve the blocking problem and show the effectiveness of 2PC, a clustering algorithm in 2PC is created to demonstrate how transaction blocking is minimized and how data consistency is maintained in a distributed system with concurrent execution of randomly generated transactions.

II. Analysis of Distributed Transaction Failures in Two-Phase Commit Protocol

A. Transaction Scenario

A transaction is defined to provide the properties of atomicity, consistency, integrity and durability(ACID) for any operation it performs. In order to ensure the atomicity of distributed transactions, an atomic commit protocol needs to be followed by all sites participating in a transaction execution to agree on the final outcome, that is, commit or abort. A variety of commit protocols have been proposed that either enhance the performance of the classical two-phase commit protocol during normal processing or reduce the cost of recovery processing after a failure.

In this study we survey a number of two-phase commit variants and optimizations including some recent ones providing an insight in the performance trade-off between normal and recovery processing. We analyze the performance of a representative set of commit protocols analytically using simulation.

Transactions are powerful abstractions that facilitate the structuring of database systems and in distributed systems in generally a reliable manner. Each transaction represents a task or a logical function that involves access to a shared database and assumes as it executes as if no other transactions were executing concurrently and as if there were no program and system failures. In this way programmers are relieved from dealing with the complexity of concurrent programming and failures, and can focus on designing the applications and developing correctly the individual transactions of the applications[13]

The ACID properties are usually ensured by combining two different sets of algorithms. The first set, referred to as concurrency control protocols, ensures the isolation property, whereas the second one, referred to as recovery protocols, ensures atomicity and durability properties. Commonly consistency is satisfied by designing transactions such that each transaction preserves the consistency of the database at its boundaries and is enforced by specifying integrity constraints on a database using triggers and alerters.

In a distributed database system (DDBS) in which the data items are stored at multiple sites interconnected via a communication network, transactions are executed in a distributed fashion at different sites based on the location of the data that they require to access. Since sites and communication links can fail independently, the atomicity property of a distributed transaction cannot be guaranteed without taking additional measures besides concurrency control and recovery protocols. Specifically, for a distributed transaction that executes across multiple sites, the sites need to agree about when and how the transaction should terminate. That is, all the sites participating in a transaction execution need to (1) eventually reach an agreement; and(2) all agree to either commit the transaction, making all its effects persistent, or abort the transaction, obliterating all it's as if the transaction had never executed. A protocol that achieves this kind of agreement is called an atomic commit protocol (ACP)[9].

B. General Overview of Commit Protocols

A commit protocol is an algorithm to ensure atomicity in a distributed transaction with the help of synchronized locking. According to Coulouris[3], Atomic commit protocols are used in distributed systems when several sites need to update their databases with the same information. Several protocols exist that have been used to address atomicity in different protocol platforms. This paper gives an overview of problems of 2PC and 3PC protocols and proposes a solution :-

i.Two-Phase Commit (2PC) protocol scenario

The atomic Two-Phase Commit Protocol (2PC) is a typical distributed commit algorithm used in computer networks and distributed database systems. It has two phases i.e the prepare and commit phase. It is used when a simultaneous data update should be applied within a distributed database. In this protocol, one node acts as the coordinator, which is also called master and all the other nodes in the network are called participants or slaves. The prepare messages from a participant to a coordinator are *YES* or *No* depending on the decision at the participant whether to vote yes or no to the requested transaction. The commit messages from a coordinator to the participant are *GLOBAL_COMMIT* or *GLOBAL_ABORT* depending if all participants have voted yes or not. All decisions at each site are logged in their respective write-ahead-log along with the transaction. The write-ahead-log must be in a stable storage to ensure that data is not lost during a site failure. In its first phase, all these

participants agree or disagree with the coordinator to commit, i.e., vote yes's or no's and in 2nd phase they complete the transaction simultaneously by getting the commit or the abort signal from the coordinator.

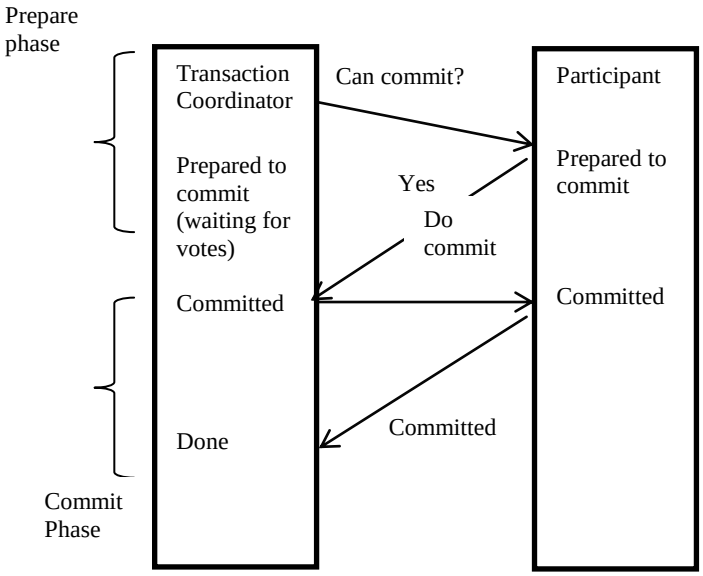

Fig.1.0. Two-Phase commit (2PC) protocol actions (Jumna *et.al,* 2012)

Global commit or abort means all participants must commit or abort, even if there is failure or timeout at any one of the nodes. Timeout means the failure of the other site. The coordinator plays the central role and flags either global commit or global abort.

The former is only shown if all the participants vote to commit and the latter is shown if at least one of the participants votes to abort or the coordinator decides to abort the current transaction. In case there is no failure at any site, the protocol is correct but it is highly desirable to consider the functionality in the presence of failure of any site at any state(Cowling et.al,2010).Consider the scenario that 2PC protocol does not have any failures and the operations are as follows:

a) Prepare phase

Coordinator: Initially the coordinator will broadcast the Begin_commit request message to all participants and enters into wait state.

Participant: When the participant receive the request message, If the participant want to commit the transaction means it respond with the Vote_commit message(Yes) to the coordinator and enters into ready state. Otherwise, the participant responds with the Vote_abort message (No) to the coordinator.

Coordinator: When the coordinator receives the reply from participant it starts 2nd phase.

b) Commit phase

Coordinator: If the participants reply with Vote_commit message(Yes), the coordinator decided to commit the transaction or abort the transaction and it will inform the participant about the outcome of the transaction.

Participant: The Participant follows the coordinator's command and it will acknowledge the coordinator.

However 2PC protocol having less communication overhead and less expensive, it has a main drawback that is blocking transaction problem(Jamuna*et.al*,2012).

Blocking problem in 2PC

The Blocking problem is described with the given circumstances that, if the coordinator fails to operate and at the same time some participant has confirmed itself to commit state. The participants keep locks on resources until they receive the next message from the coordinator after its recovery. For instance consider a situation that a participant has sent VOTE-COMMIT message to the coordinator and has not received either GLOBAL-COMMIT or GLOBAL-ABORT message due to the coordinator's failure. In this case, all such participants are blocked until the recovery of the coordinator to get the termination decision. The blocked transactions continue to keep all the resources until they obtain the final decision from the coordinator after its recovery (Schapiro & Milistein,2012).

ii.Three-Phase Commit(3PC) scenario

A 3PC is a non-blocking protocol which eliminates the blocking problem faced by 2PC protocol. Three Phase Commit protocol operations is similar with the two phase commit protocol, only difference is it has extra phase called Pre_commit phase where it takes the preliminary decision. The Three Phase Commit protocol (3PC) performs the operation in three phases are Prepare phase, Pre-commit phase, Commit/Abort phase. Among the three phases Pre-Commit phase eliminates the blocking problem but it comes with an extra cost of message transfers. It's represented below;

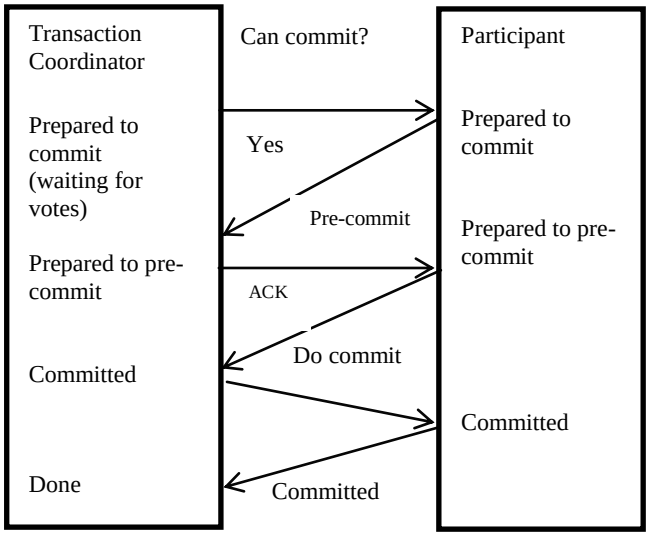

Fig 1.1 Three-Phase Commit Protocol architecture (Jamuna *et.al*,2012)

a) Prepare phase

Coordinator: Initially the coordinator will broadcast the Begin_commit request message to all participants and enters into wait state.

Participant: When the participant receive the request message, If the participant want to commit the transaction means it respond with the Vote_commit message(Yes) to the coordinator and enters into ready state. Otherwise, the participant responds with the Vote_abort message(No) to the coordinator(Jamuna *et al*, 2012).

Coordinator: When the coordinator receives the reply from participant it starts 2nd phase.

b) Pre-Commit or Buffering

Coordinator: When the coordinator receives Vote_commit message within the time from the participant, the coordinator broadcast the Pre-Commit message to all participants .At this phase preliminary decision can be made and it moves to prepared state.

Participant: When the participant accepts the Pre_commit message it will send acknowledge message the coordinator.

Coordinator: When the Coordinator receive ACK message from participant it starts 3rd phase.

c) Commit/Abort phase

Coordinator: The coordinator decided to commit the transaction or abort the transaction and it will inform the participant about the outcome of the transaction.

Problems with 3PC

Three-Phase Commit Protocol is problematic only when there are multiple site failures. For example, let's consider a case where the coordinator is in pre-commit state and fails just after sending a commit message and the slave also fails just before or after receiving this message as so by its failure, the slave moves to the aborted state but according to the protocol specifications, the coordinator goes to the committed state, either it fails or receives acknowledgement. Hence, the coordinator moves to the committed state without receiving acknowledgement and the failed slave moves to the aborted state without sending the acknowledgement. In this way, coordinator and participant show different final states due to their failures. Although 3PC protocol eliminates the blocking problem, it involves an extra overhead of one more cycle and in turn increases time taken for the transaction to complete (Singh *et.al*, 2011).Because of high communication overhead 3PC has not been implemented so far.

III. Types of failures introduced in distributed systems

A transaction clustering algorithm in 2PC addresses the following failures introduced in a distributed system environment:-

A. Site failure

A failure of any type is normally detected by the absence of an expected message. Site failures are usually due to software or hardware failures. These failures result in the loss of the main memory contents. In distributed database, site failures are of two types:

i. Total Failure where all the sites of a distributed system fail.

ii. Partial Failure where only some of the sites of a distributed system fail.

Site failures are modelled by a failure transition, which is a special kind of local state transition. Such a transition occurs at the failed site the instant that it fails. The resulting local state is the state initially occupied by the failed site upon recovering. An underlying assumption is that a site can detect when it has failed.

B. Coordinator failure

The 2PC protocol is the simplest and the best known protocol which serves as an object to ensure the atomic commitment of a distributed transaction. It is a centralized control mechanism based on the coordinator, which coordinates the actions of the others called participants. A coordinator sends transaction request to participants and waits for their replies in the first phase. After receiving all replies, the coordinator sends a final decision to participants in the second phase.

The atomicity property of the protocol means that the transaction must be performed at all sites or not at all; this is achieved by letting all participants vote YES or NO to the particular transaction depending if they can commit it or not. Only when all sites are ready to commit, the coordinator sends a GLOBAL-COMMIT to the participants as confirmation that they can commit the transaction. A site can be both coordinator and participant at the same time but for different transactions. If a coordinator site crashes and participants waits for a final answer from the coordinator, if he should commit the transaction or not, the participants are blocked as long as the coordinator is down. This is the problem with 2PC algorithm which reduces high degree of data availability [14]

IV. Tools and Specifications

In order to show coordinator and site failure in two-phase commit protocol, the following tools and specifications were adopted:-

i) *Britonix Transaction Manager (BTM)* - is a simple but complete implementation of the Java Transaction API (JTA)1.1 API(application programming interface). It is a fully working XA transaction manager that provides all services required by the JTA API while trying to keep the code as simple as possible for easier understanding of the XA semantics.

ii) *MySQL database Management System* - to act as the backend data resource manager. This should be MySQL 5.1 or higher version.

iii) Java Development environment – to provide virtual machine environment

A. Bitronix Transaction Manager(BTM)

The BTM is a simple but complete implementation of Java transaction applications whose goal is to provide a fully working transaction manager that provides all services required by the Java applications while trying to keep the code as simple as possible for easier understanding of semantics. BTM is such important in transaction management in that it has useful error reporting and logging methods which make it easier to know when an error occurs. BTM configuration settings are stored in a Configuration object. It can be obtained by calling *TransactionManagerServices.getConfiguration().*

BTM is a perfect choice for a project using transaction capabilities by using Java Transfer manager (JTM) facade. It is possible to integrate BTM in web containers like Tomcat or Jetty and get raw access to a JTA implementation.. BTM has proved to be stable and mature enough to be used in production. Currently BTM is very stable and usable. JDBC resources are working pretty well and recovery of crash works fine.

B. Bitronix JTA Transaction Manager With Mysql

The Java Transaction API (JTA) allows applications to performs distributed transactions, to access and update systems having multiple transaction resources: databases, message queues, custom resource, or resources accessed

from multiple processes or on multiple hosts as participants in a single transaction.

V.Simulation of 2PC Coordinator Failure

To demonstrate the fact that the Two–phase is blocking, the used Bitronix transaction manager, *mySQL* server and *JCreator* IDE to simulate a two-phase commit protocol failure. Two practical implementations of coordinator failures were carried out. The first one demonstrated coordinator failure for distributed transactions on a single database while the second one demonstrated coordinator failure for distributed transactions on distributed data resources. In both cases:

i) The *mySQL* database acted as *resource manager*.

ii) The JDBC driver, in this case, *mysql-connector-java-5.1.10-bin.jar*, acted as *Resource Adapter*.

iii) The *main Java* classes in the projects, acted as *Coordinators. T*wo main classes, were used that were namely, the *TwoPCCoordinatorFailureClass* and the and *TwoPCCoordinatorFailureClass1*. The first main class, which is the, *TwoPCCoordinatorFailureClass* was to show the coordinator failure for distributed transactions involving one database while the second class which is the, *TwoPCCoordinatorFailureClass1* was to show coordinator failure for distributed transactions on distributed databases.

iv) *Bitronix Transaction Manager (BTM)* was used *as a Transaction Manager.* Its function was to receive messages from the coordinator and participant and forwards the messages to the corresponding participants and coordinators.

v) The transaction classes, keep *Transaction ID*, For example the code below, *"bitronix.tm.BitronixTransactionManager.<init >(BitronixTransactionManager.java: 64)" has transaction ID of 64.*

vi) The *transaction sub-classes* send a request for the transaction to the transaction manager-through message calling. For example, the code "*btm.commit();",* is a call made to the Bitronix transaction manager to commit a transaction.

vii) A *transaction branch*- is associated with a request to each resource manager involved in the distributed transaction. Each transaction branch must be committed or rolled back by the local resource manager. For example, the code,

```
"catch (Exception ex) {
        ex.printStackTrace();
try {
                                btm.rollback();
                        } catch (Exception e) {
                                e.printStackTrace();
                        }"
```

is a transaction branch that results when the transaction class cannot commit a transaction.

The relationships among these entities are shown Figure 1.2 below. The transaction manager was responsible for making the final decision either to *commit* or *rollback* any distributed transaction. A commit decision should have lead to a successful transaction; rollback leaves the data in the database unaltered. JTA specified standard Java interfaces between the transaction manager and the other components in a distributed transaction: the application, the application server, and the resource managers.

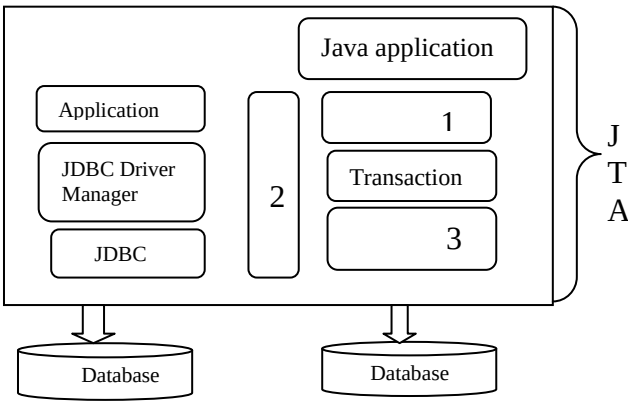

Figure 1.2: Relationship Among Distributed System Entities

The numbered boxes, 1, 2 and 3 around the transaction manager correspond to the three interface portions of JTA. The box number 1 is the *userTransaction*, which is an interface that provides the application the ability to control transaction boundaries programmatically. The second (2) is the *transaction manager*, which is an interface that allows the application server to control transaction boundaries on behalf of the application being managed. Lastly, the *XAResource* is box number 3, and is a Java mapping of the industry standard XA (extended Architecture). XA is used for communication with the transactional resources. The two databases were created in MySQL and were named *KisiiBranch* and *NairobiBranch*.

A. Simulation Procedure

1. Two databases were created in MySQL server. These were given names *KisiiBranch* and *NairobiBranch*.
2. Two tables were created, one in each of these databases, with the name *bankcustomer*.
3. Table *bankcustomer* had five columns, namely *CustomerID, CustomerName ,Address, City and AccountBalance.* table1.0 below shows the structure *of these tables.*

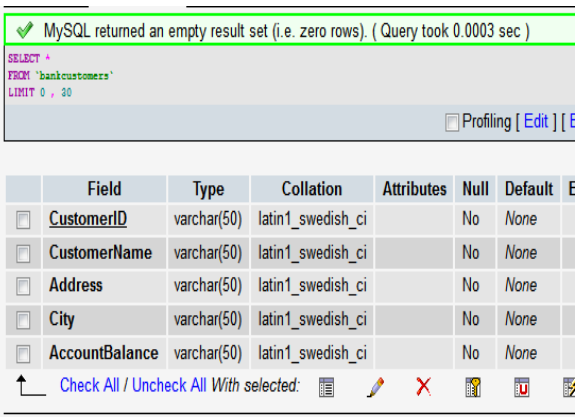

Table 1.0 .Structure Of The Database Tables, Bankcustomers

B.Site Failure Demonstration in 2PC

Table 1.1 2PC site failure

*bitronix.tm.BitronixTransactionManager logVersion… Atbitronix.tm.BitronixTransactionManager.<init>(BitronixTransactionManager.java:64) at bitronix.tm.TransactionManagerServices.getTransactionManager (TransactionManagerServices.java:62) AtTwoPCProtocol.TwoPCCoordinatorFailure.main(TwoPCCoordinatorFailure.java:32)Caused by: com.mysql.jdbc.exceptions.jdbc4.CommunicationsException: "**Communications link failure"** The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server.*

The first line of this table gives an overview of the obtained results, that is, the information shown is the one contained in bitronix transaction log file. The second line is the initialization of the bitronix transaction manager, which has been given a unique ID of 64. The third line initializes the transaction manager services, and has been assigned a unique ID of 62. The '*TwoPCProtocol*', is the package name while '*TwoPCCoordinatorFailure*', is the name of the main class, which became our coordinator. This was given a unique ID of 32. The next line gives information on the status of the communication link to the data resource. It is evident here that the data resource is down, as indicated by the statement, '*Caused by: com.mySQL.jdbc.exceptions.jdbc4.CommunicationsException:* ***Communications link failure***'.

The last line further explains that *Bitronix*, through the JDBC driver, which was '*mySQL-connector-java-5.1.10-bin.jar',* had sent a packet successfully. However, the data resource could not give any response. This was because we intentionally put the *mySQL* server offline.

C. Coordinator Failure Demonstration in 2PC

To demonstrate coordinator failure, the data resource was put online and ran the source codes of coordinator failure involving distributed transactions directed towards a distributed data resource. Table 1.2 below gives a snippet of the output of the Bitronix transaction manager log file where there is a coordinator failure in distributed transactions and distributed data resource.

Feb 5, 2015 7:12:52 AM bitronix.tm.recovery.Recoverer recoverAllResources WARNING: error running recovery on resource 'TwoPCCoordinatoFailureClass1', ***resource marked as failed*** *(background recoverer will retry recovery) bitronix.tm.recovery.RecoveryException: cannot start recovery on a PoolingDataSource containing an XAPool of resource TwoPCCoordinatoFailureClass1 with 0 connection(s) (0 still available) atbitronix.tm.resource.jdbc.PoolingDataSource.startRecovery(PoolingDataSource.java:227) at bitronix.tm.recovery.Recoverer.recover(Recoverer.java:253) atbitronix.tm.recovery.Recoverer.recoverAllResources(Recoverer.java:223) at bitronix.tm.recovery.Recoverer.run(Recoverer.java:138)*

Table 1.2: 2PC coordinator failure

As shown in the table, the *Bitronix* transaction manager starts by obtaining the Java Virtual Machine unique ID, which is the address of the *localhost*, that is, 127.0.0.1.

Line three of the snippet above clearly indicates that '*TwoPCCoordinatoFailureClass1*', which was our main class in the Java application and hence our coordinator, has failed. This is evident by *Bitronix* output statement, '*resource marked as failed (background recoverer will retry recovery)*'.

D.Snippet of the current 2PC protocol

The algorithm in **table 1.3** was compiled and run in *Jcreator* IDE.

As shown in the table, the algorithm consisted of three queries, two for inserting while the other one for retrieving records from the database. Observation of this algorithm reveals that it consists of six of these nested statements, three for inserting and retrieving data and the three for error handling when errors are detected in the

Table 1.3 Snippet of the current 2PC protocol

```
private static final String INSERT_QUERY="insert into
Bankcustomers(CustomerID,CustomerName,Address,City,AccountBal
ance)values (2356,Teresa,Kisii,Nairobi,25000)";
private static final String INSERT_QUERY1="insert into
Bankcustomers(CustomerID,CustomerName,Address,City,AccountBal
ance)values (50, Susan,Nakuru,Nairobi,25000)";
private static final String SELECT_QUERY="select * from
bankcustomers";
……
BitronixTransactionManager btm
=TransactionManagerServices.getTransactionManager();
try                    { btm.begin();
Connection connection =mySQLDS.getConnection(USER_NAME,
PASSWORD);
PreparedStatement pstmt
=connection.prepareStatement(INSERT_QUERY);
for(int index = 1; index <= 5; index++) {
                       pstmt.setInt(1,index);
                       pstmt.setString(2, "Customers_" + index);
                       pstmt.setString(3, "" + (4 + index));
                       pstmt.setString(4, "Nairobi");
```

algorithm or the data resources. This is the root of the concurrency and blocking problems in two-phase commit protocol. This is because these transactions are partitioned; hence each of them is transmitted to the data resources independent of each other. Hence if one of them fails to respond, the others are blocked, waiting for its recovery.

VI.Simulation of the Transaction Clustering Algorithm

To address the concurrency and blocking problem in two-phase commit protocol, a simulated transaction clustering algorithm for optimizing distributed transactions is designed.

A. Transaction Clustering Algorithm Architecture

The algorithm consists of the following components:

i) *Transaction manager*- the purpose of this component was to send and receive messages from the coordinator and participant. It also contains the recovery procedures to deal with transaction failures. *Bitronix Transaction Manager* was taken to be the transaction manager.

ii) *Coordinator*- its function is to monitor and execute atomic transactions. The Java main class, Coordinator was taken to be the coordinator, which was declared as follows:

public class Coordinator {
……}

iii) *Resource manager*- its function was to keep a record of stable committed transactions in storage. MySQL database was used for this perspective.

iv) *Resource Adapter*- The function of this component t was to provide database connectivity. It was taken to be the *mysql-connector-java-5.1.10-bin.jar* .

v) *Participants*-the function of these components was to take part in the voting process and take appropriate actions locally, which could be transaction commit or transaction abort. These were taken to be the three sub-transactions, two of which were to insert data into the database (*KisiiBranch* and *NairobiBranch*), while the remaining one was to update the *HeadOffice* database

vi) *Data managers*- the function of these components was to manage data transfer between its replica and other sites. These were taken to be the *Connection* constructs that provided the path to the databases.These were declared as follows;

```
Connection connection =
DriverManager.getConnection(

        "jdbc:mysql://localhost:3306/NairobiBranch", "root", "");
……
Connection connection1 =
    DriverManager.getConnection(

        "jdbc:mysql://localhost:3306/KisiiBranch", "root", "");
……
Connection  connection2=
DriverManager.getConnection(

        jdbc:mysql://localhost:3306/HeadOffice", "root", "");
```

B. Overall Simulation Architecture

Figure 1.3 below shows the overall simulation architecture. Its shows the relationships among the above mentioned entities. As shown, the *bitronix transaction manager* directly communicates with the *coordinator*, which in turn communicates with *data managers*. The *data managers* communicate with *participants*. The *participants* communicate with the *resource managers* via the *resource adapter*.

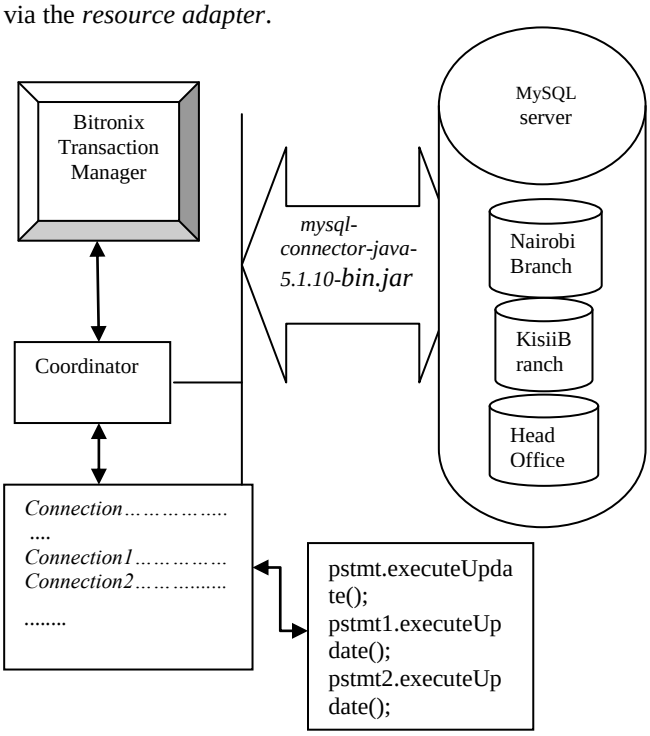

Fig 1.3 Overall simulation architecture

The system use diagram for this transaction clustering algorithm simulation consisted of all the above entities, which interact as shown in Figure 1.4 below.

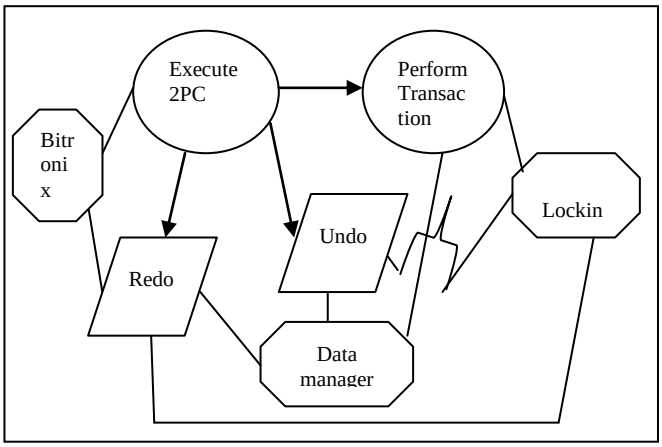

Figure 1. 4: Transaction Clustering Algorithm Use Diagram

This figure shows that the two possible operations for transactions is either redo or undo. There is a strong inter-relationships among the various entities, which work together to achieve a given functionality, either successfully updating the database or successfully performing a rollback so that data consistency is maintained.

The diagram below shows the class diagram for the package *TwoPCCoordinator* package. *Coordinator* is the main class in the *TwoPCCoordinator* package. Its main function initiates *Coordinator* and *Participant* and to keep a list of them when a transaction occurs. The

Participant thread will execute, redo or undo a sub-transaction. It carries out the proper procedures of the 2PC as participant by following instruction from the coordinator. It carries out recovery in case of failures.

The Package diagram

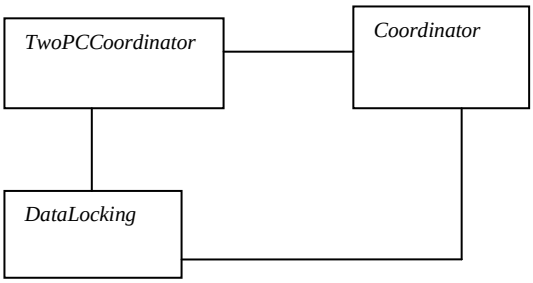

Figure1.5: Transaction Clustering Algorithm Package Diagram

The *datalocking* component is responsible for locking data so that only one transaction have access to it at any particular moment. This is important if database inconsistency in distributed databases is to be avoided.

VII. Two Phase Commit Protocol Clustering transaction algorithm output

----------Configuration: <Default>----------
WARNING: cannot get this JVM unique ID. Make sure it is configured and you only use ASCII characters. Will use IP address instead (unsafe for production usage!).
Feb 15, 2015 8:39:42 PM bitronix.tm.Configuration buildServerIdArray
Feb 15, 2015 8:39:42 PM bitronix.tm.recovery.Recoverer run
INFO: recovery committed 0 dangling transaction(s) and rolled back 0 aborted transaction(s) on 0 resource(s) [] (restricted to serverId '127.0.0.1')

NAIROBI_BRANCH_VOTE :TRANSACTION_COMMIT

KISII_BRANCH_VOTE :TRANSACTION_COMMIT

HEAD_PFFICE_VOTE :TRANSACTION_COMMIT

COORDINATOR_DECISION :GLOBAL_COMMIT

Table 1.4. Two Phase Commit output

When an algorithm was run all the sites voted TRANSACTION-COMMIT and the coordinator decision was GLOBAL-COMMIT.

A check on the three sites confined that these transactions had actually committed as shown in table 1.5,1.6 and 1.7 below.

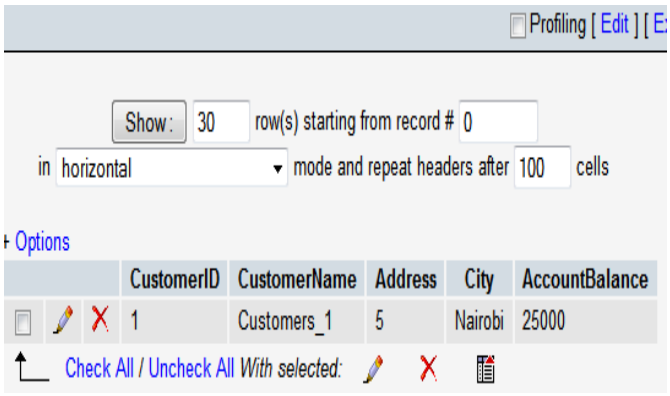

Table 1.5 headoffice site final status after commit

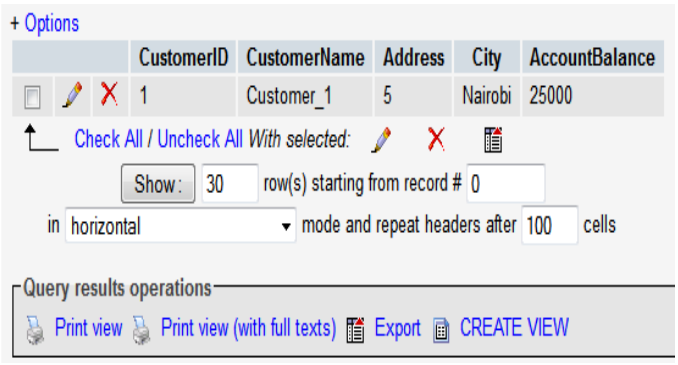

Table 1.6 Nairobi Branch site final status after commit

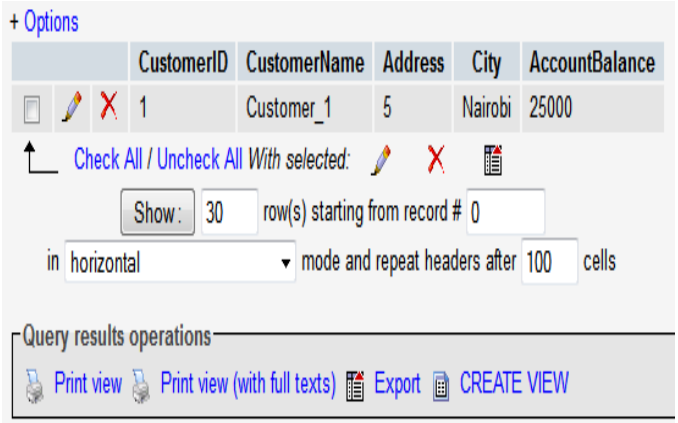

Table 1.7. .KisiiBranch site final status after commit

A. Handling of site failure by transaction clustering algorithm

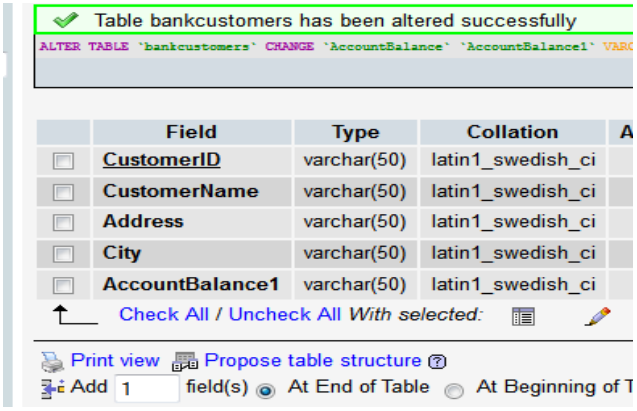

Table 1.8: NairobiBranch modification procedure

To investigate how this new algorithm handles site failures, one of the sites, the *NairobiBranch* was intentionally modified by setting the fifth column to be *AccountBalance1*,instead of *AccountBalance* as contained in the algorithm.**Figure 1.8** above shows how this modification was done.

The algorithm was then re-run. Table 1.9 below shows the *Bitronix* Transaction Manager output.

----------Configuration: <Default>----------
Feb 15, 2015 9:56:32 PM bitronix.tm.recovery.Recoverer run INFO: recovery committed 0 dangling transaction(s) and rolled back 0 aborted transaction(s) on 0 resource(s) [] (restricted to serverId '127.0.0.1') Feb 15, 2015 9:56:32 PM bitronix.tm.BitronixTransactionManager shutdown INFO: shutting down Bitronix Transaction Manager
Feb 15, 2015 9:57:32 PM bitronix.tm.BitronixTransaction timeout WARNING: ***transaction timed out****: a Bitronix Transaction with GTRID [3132372E302E302E310000014B8E99657300000000],* ***status=MARKED_ROLLBACK****, 0 resource(s) enlisted (started Sun Feb 15 21:56:32 EAT 2015)*

Table 1.9 transaction manager output

The concurrency and blocking control in the new clustered algorithm is demonstrated in this table.

If the coordinator poorly managed the transactions, the other two sites would have voted, *TRANSACTION_COMMIT*, since their sites were never affected by the changes that were carried out. However, none voted, and hence the developed algorithm can manage concurrency access to distributed databases. Moreover, since none of the sites voted *TRANSACTION_COMMIT*, they cannot claimed to have been blocked by the failure of the site, *NairobiBranch.* Had they voted, *TRANSACTION_COMMIT*, they could have claimed to have been blocked by *NairobiBranch*, since they belonged to the same coordinator and therefore according to the two phase commit protocol requirement of atomicity, they were expected to commit together as a group.

VIII. Conclusions

This work focused on site and coordination failures as common drawbacks in the two-phase commit protocol. The developed algorithm had four distinct steps. The first step was for the Coordinator to send a *VOTE_REQUEST* message to all participants, in this case, all the data resources that were to be manipulated by the coordinator transactions. When a participant received a *VOTE_REQUEST* message, it returned either a *VOTE_COMMIT* message to the coordinator telling the coordinator that is prepared to commit or a *VOTE_ABORT* message. Each participanst that voted for a commit waited for the final reaction by the coordinator. If a participant received a *GLOBAL_COMMIT* message, it locally committed the transaction else it aborted the local transaction. The commit process was characterized by database update while the abort process left the database unaltered, in accordance with the atomicity principle. All these were accomplished in *Jcreator IDE* and *mySQL* database. The research objectives were achieved as already stated above. The new algorithm had an improved performance in as far as site failure and coordinator failures were concerned. It was shown that all transactions committed or aborted and the databases were left in a consistency state after a commit, or in unchanged state in case of a transaction abort process.

IX. Recommendations and future works

Coordinator and site failure are common problems in the current two phase commit protocol. As such, several efforts have been made to overcome them. The design of the three phase commit protocol was meant to overcome these challenges by introducing an extra phase, called the pre-commit phase. However, this approach is complicated to implement, has more communication overheads and maintaining inconsistency towards network partitioning is a serious problem. The developed algorithm has been shown to solve the coordinator failure and site failures without introducing extra overheads, which is a serious problem in three phase commit protocol. Moreover, the algorithm has been shown to be ideal in maintaining consistency of the database and chances of partitioning are rare because all transactions are clustered and hence either commit or rollback as a group. Therefore we recommend it adoption in distributed transaction handling. The possible improvement areas include the design of this algorithm so that the explicit *TRANSACTION_COMMIT* voting can be part of the responses received from the participants. Moreover, there is need to implement this algorithm in other backbends, such as *SQL* and oracle servers.

References

[1] K. Reddy , Kitsuregawa M. *Reducing the blocking in two-phase commit with backup sites.* Information Processing Letters, v 86, n 1, p 39-47, April 15, 2006
[2] A. Tanenbaum , Van Steen M. 2007.*Distributed systems – Principles and Paradigms*, second edition.
[3] G.Coulouris, Dollimore J., Kindberg T. 2007. *Distributed systems – Concepts and Design*, fourth edition.

[4] V. Goebel(2011), "Distributed Database Systems", Department of Informatics, University of Oslo.
[5] T. Byun ., Moon S. *Non-blocking two-phase commit protocol to avoid unnecessary transaction abort for distributed systems.* Cheongryang 130-012 Seoul South Korea. Journal of Systems Architecture . Volume 43, Issues 1-5, March 2012, Pages 245-254
[6] Arun Kumar Yadav and Ajay Agarwal,"*A Distributed Architecture for Transactions Synchronization in Distributed Database Systems*", International Journal on Computer Science and Engineering Vol. 02, No. 06, 2011.
[7] MengQingyuan, Wang Haiyang, XuChunyang,"*A New Model for Maintaining Distributed Data Consistence",* International Conference on Computer Science and Software Engineering, IEEE 2008.
[8] K. Tabassum , Taranum F, Damodaram A"*A Simulation of Performance of Commit Protocols in Distributed Environment"*, in PDCTA, CCIS 203, pp. 665–681, Springer, 2011.
[9] S. Fahd and Tarek Helmy . Al-Otaibi," *Dynamic Load-Balancing Based on a Coordinator and Backup Automatic Election in Distributed Systems*", International Journal of Computing & Information Sciences Vol. 9, No. 1, April 2011.
[10]Peng-Yeng Yin, Shiuh-Sheng Yu, Pei-Pei Wang and Yi-Te Wang. *A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems* [J].Computer Standards & Interfaces, Volume 28, Issue 4, April 2006, 441-450.
[11] Y, Amir, B.A. Coan, J.Kirsch, and J. Lane," *Byzantine Replication under Attack*," Proc. IEEE Int'l Conf. *Dependable Systems and Networks*, pp. 105-144, June 2010.
[12] Syam Menon. *Allocating fragments in distributed databases* [J]. IEEE *Transactions on Parallel and Distributed Systems*, 2005, 16: 577-585.
[13] ToufikTaibi, Abdelouahab Abid, Wei Jiann Lim, YeongFeiChiam, and Chong Ting Ng, "*Design and Implementation of a Two-Phase Commit Protocol Simulator",* The international Arab Journal of Information Technology, Vol. 3, No. 1, January 2006.
[14] P.Jamuna,S.Sathiyadevi,S.Tamilarasi*"Backup Two Phase Commit Protocol(B2PC)renders trustworthy coordination problem over distributed transactions"*,International Journal for Advanced Research in Computer Science and Software Engineering,Vol 2,issue 9 Sept,2012
[15] R. Schapiro and R. Milistein, *"Failure recovery in a distributed database system,*" in Proc. 1978COMPCONConf.,Sept. 2012.
[16]V.Manikandan, R.Ravichandran, R.Suresh, F.SagayarajFrancis"*An efficient Non-blocking Two Phase Commit Protocol for Distributed Transactions*" Vol.2 Issue 3 May 2012 pp 778-791.
[17] R. Kotla, L. Alvisi, M. Dahlin, A.Clement, and E. Wong,"Zyzzyva: *Speculative Byzantine Fault Tolerance,"* Proc. 21st ACM Symp. Operating Systems Principles,pp. 45-58,Oct.2010.

Authors

Teresa K. Abuya

Received Bsc. Information Technology from Jomo Kenyatta University of Agriculture & Technology (JKUAT),Kenya; Msc .in Computer Systems from JKUAT. Serving as an assistant lecturer at Kisii University, Kenya. Her research interests include but are not limited to: Distributed database systems, Mobile & cloud computing, Internet of Things, ICT for Development, Data mining & Knowledge discovery.

Dr.Richard M. Rimiru

Received Bsc.in Statistics & Computer Science from Jomo Kenyatta University of Agriculture & Technology(JKUAT),Kenya; Msc.in Computer Science from National University of Science & Technology, Zimbambwe; PhD in Computer Science & Technology,Central South University(CSU) ,Changsha,P.R.China.Serving as a senior lecturer in School of Computing & Information Technology at JKUAT, Kenya. His research interest include but not limited to: Database systems,artificial intelligence,knowledge based systems,computer networks & mobile computing.

Dr.Cheruiyot W. Kipruto

Received Bsc. In Statistics & Computer Science from Jomo Kenyatta University of Agriculture & Technology(JKUAT),Kenya; Msc.inComputer Application Technology, CentralSouth University(CSU),P.R.China; PhD in Computer Science &Technology ,Central South University(CSU)ChangshaP.R.China. Serving as a senior lecturer in School of Computing & Information Technology at JKUAT,Kenya. His Research interest include but not limited to: Multimedia Data Retrieval, Internet of Things, Evolutionary Computation for Optimization, Digital Image Processing and ICT for Development.

