International Journal of Computer (IJC)

ISSN 2307-4523 (Print & Online)

© **Global Society of Scientific Research and Researchers**

http://ijcjournal.org/

An Alternative Model to Overcoming Two Phase Commit Blocking Problem

Hassan Jafar Sheikh Salah[a]*, Dr. Richard M. Rimiru[b] ,
Dr. Michael W. Kimwele[c]

[a,b,c]*Computer Science, Jomo Kenyatta University of Agriculture and Technology Kenya*

[a]*Email: xsnjcfr10@gmail.com*

[b]*Email: rimirurm@gmail.com*

[c]*Email: mkimwele@jkuat.ac.ke*

Abstract

In distributed transactions, the atomicity requirement of the atomic commitment protocol should be preserved. The two phased commit protocol algorithm is widely used to ensure that transactions in distributed environment are atomic. However, a main drawback was attributed to the algorithm, which is a blocking problem. The system will get in stuck when participating sites or the coordinator itself crashes. To address this problem a number of algorithms related to 2PC protocol were proposed such as back up coordinator and a technique whereby the algorithm passes through 3PC during failure. However, both algorithms face limitations such as multiple site and backup coordinator failures. Therefore, we proposed an alternative model to overcoming the blocking problem in combined form. The algorithm was simulated using Britonix transaction manager (BTM) using Eclipse IDE and MYSQL. In this paper, we assessed the performance of the alternative model and found that this algorithm handled site and coordinator failures in distributed transactions using hybridization (combination of two algorithms) with minimal communication messages.

Keywords: Distributed transactions; atomicity; two phase commit protocol; three phase commit protocol.

1. Introduction

In databases, transactions are unit of work that should be treated as whole. Transactions are required to keep the consistence of the database prior to the initiation and after the execution of the transactions.

----

* Corresponding author.

The transactions have four properties; atomicity, consistence, isolation and durability. These properties should be ensured. The atomicity property defines that transactions should be done or nothing. The consistence property states that transaction should leave the database in a consistence state. These two properties are concerned when transaction become distributed. And distributed transactions might involve in processes at different sites. This phenomenon raised the concern about commit/ abort decisions when transactions are distributed [1]. However, a number of distributed algorithms were developed among them; one phase commit protocol (1PC) two phase commit protocol (1PC), three phase commit protocol (3PC).

1.1. Transactions in Distributed Systems

Transaction is a sequence of operations performed on a single logical unit of work. In other words, is a collection of operations performed against the data items of the database [2]. To preserve the integrity of database ACID properties of transactions should be ensured i.e., atomic, consistency, isolated and durable.

Atomicity

The atomicity property is referred to as all-or-nothing, which means that all of transaction's operations should be done all or not at all. This keeps the atomic requirement of the transaction whether transactions are centralized or distributed. Atomic commitment protocol algorithms such as 2PC and 3PC were designed to obey the atomicity of transaction in distributed environments. Atomicity mainly deals with distributed transaction.

Consistency

This property ensures the consistence of database prior to initiation and after transactions. Data should be consistent when transactions were completed. Transactions are moving database from consistent state to a consistence state. This is consistency basic requirement.

Isolation

The isolation property requires that transaction should not interfere each other. Transactions should be independent. Transaction should be visible to other

1.2. Two Phase Protocol

According to Lin and his colleagues [3], in the field of distributed databases, the most commonly used protocol is the two-phase commit protocol. As the term two describes the algorithm consist of two phases as explained in the following scenarios:

Phase 1: Site 1 (the coordinator) sends the transaction to site 2 (the only slave). Site 2 starts the local transaction, executes the transaction but doesn't commit. If the execution is successful, site 2 sends a "YES" message too site 1 to commit; otherwise it sends "NO" message to abort. Phase 2: Site 1 receives the response (vote) from site 2. If the vote is "YES" and site 1 agrees, then it sends a "commit" message to site 2; otherwise it sends abort message [2]. This shows that algorithm is composed two phases. Some scholars argue that the execution of the transaction should be recognized as phase, however, researchers call the algorithm two phase commit protocol. This scenario above works when there is no failure in the system. The main problem of this algorithm appears when site or coordinator fails. The following diagram is going to show the states of the algorithm. The picture consists of two finite state machines one for the coordinator and the other for the participant.

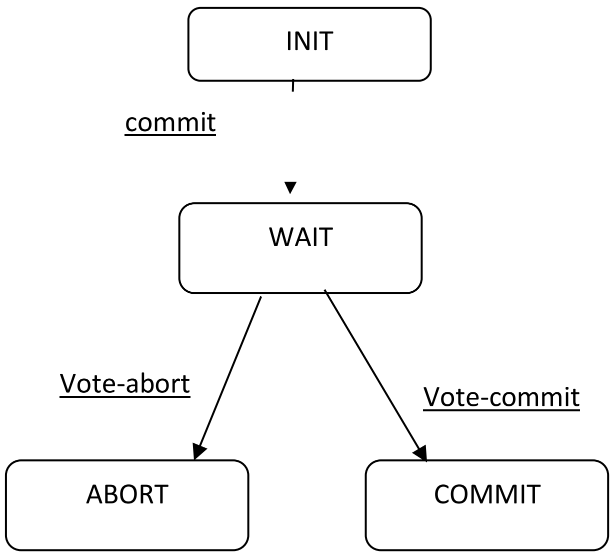


Figure1: finite state machine for the coordinator [4]

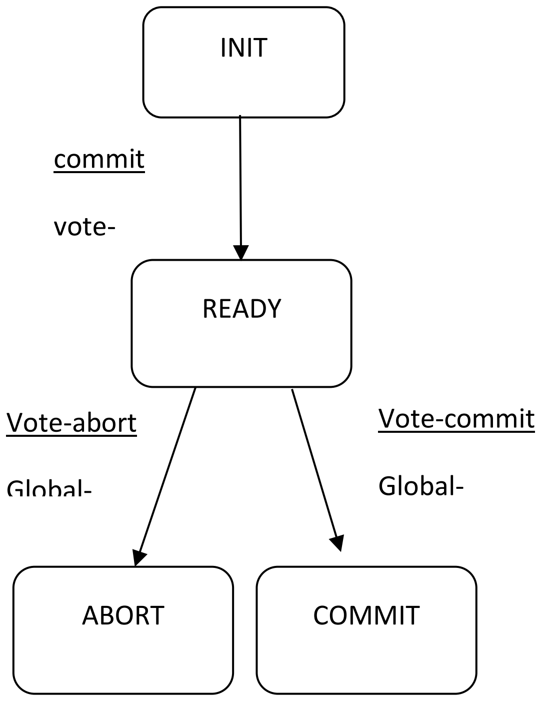


Figure 2: finite state machine for the coordinator [4]

1.2.1. Problems associated with two phase commit protocol

According to the diagram, a blocking can occur in one of the following states

1. Participant P can be in initial state waiting response from coordinator, if that message not received, the participant aborts the transaction.
2. Coordinator can be blocked in waiting for a vote in each of the participant, after sometime, the coordinator aborts the transaction
3. Participant can be blocked in ready state, waiting a global-commit message, it continues blocking till the coordinator recovers [4]

The two phase commit protocol is referred to as blocking protocol when a participant is in a ready state and the coordinator has failed [5]. To overcome this blocking problem another algorithm was proposed, most probably is the three phase commit protocol

2. The three phased commit protocol

The three phase commit protocol was designed to overcome to the blocking problem of the 2pc. It consisted of three phases

Voting phase

1. Coordinator multicasts vote-request message to all participants Each participant receiving the message, if it commits the transaction, it sends “YES”, if it is not committing it sends “NO”

Pre-commit phase

Coordinator waits response from all participant

i) If all participants vote “YES”, prepare-commit is sent to the participant
ii) If at least of the participant voted “NO”, a Global-abort is sent to the participant that replies NO

2. Participants wait for reply from coordinator

i) If participant receive prepare commit message from the coordinator, the participant commits transaction and sends ready commit acknowledge to the coordinator
ii) If Global-abort message is received from the coordinator. Participant abort the transaction.

The three phase commit protocol goes through three states, INI, WAIT and Pre-commit as explained in the diagram below:

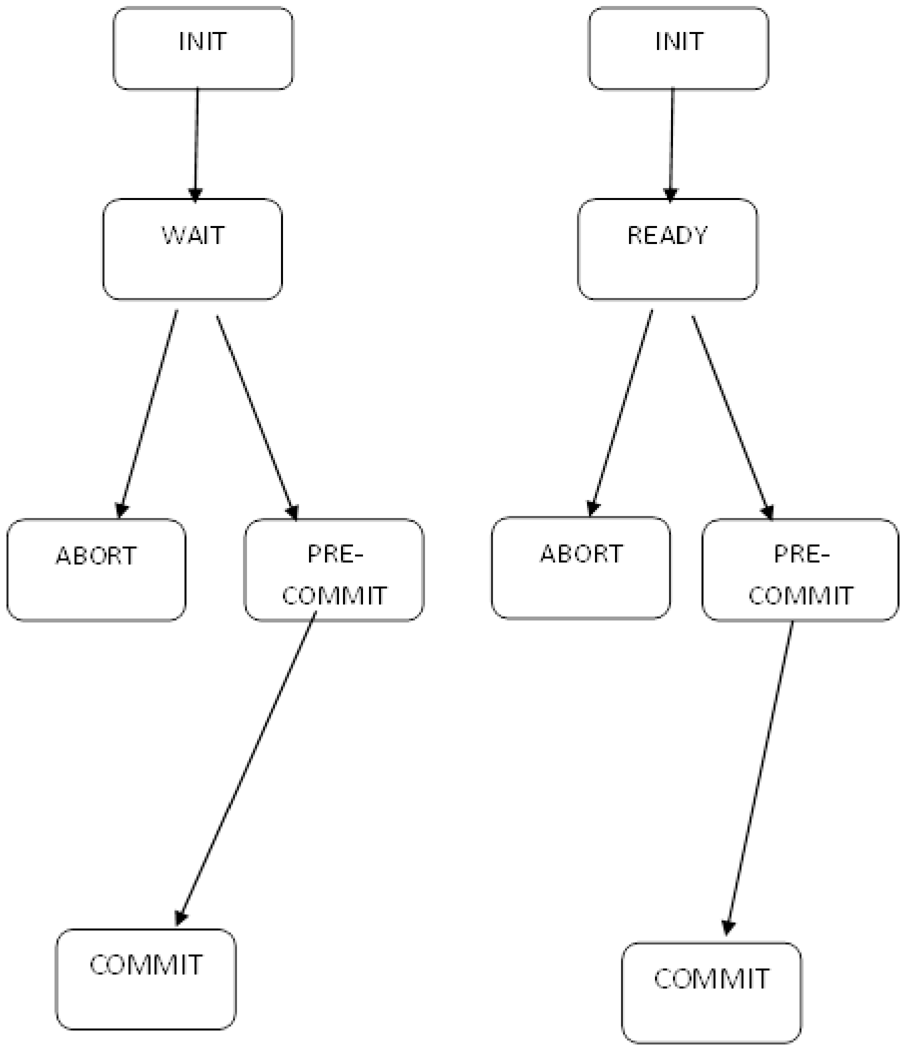


Figure 3: three phase commit protocol states

2.1. Problems associated with the three phase commit protocol

The biggest problem to the algorithm is when there are multiple site failures. To understand it, let's illustrate this case. A coordinator is in pre-commit state and fails before or after sending commit message, and participant fails before or after sending receiving message. In this condition, the coordinator moves to the committed state without acknowledgement from participant, and the participant moves to the aborted without acknowledgement [6]. Although, it is widely referred in literature, it is not used in practice and real distributed systems because the blocking problem of the 2pc is rarely encountered. On the other hand, initially 3PC was introduced as an extension of 2PC, however it brought overhead problems [7]. A number of messages are being transferred hence causing network overhead and congestion when participating sites increase.

3. Review of 2PC Proposed Algorithm

To overcome the 2PC blocking protocol a number of algorithms relating 2PC were proposed. Manikandan and his colleagues [8] proposed an effective non-blocking 2PC protocol. Their algorithm was transaction base middleware in which the server components manage the workload. The algorithm added backup coordinator and middleware to the 2PC protocol architecture to ensure recovery and minimize blocking problem. This algorithm used statistical method to prove the probability that the backup coordinator and the coordinator fail. However, the algorithm was not simulated and blocking is still there when both the coordinator and backup coordinator fail. The most widely known non-blocking protocol is the 3PC protocol which we discussed it above.

Kumar and his colleagues [9] made an enhancement of 3PC protocol algorithm worked on 2PC protocol and runs 3PC upon failure. They made modifications on 2PC protocol algorithm and an extra phase of pre-commit worked like the 3PC in the times of failure. The extra phase is performed at sending time of sender-side. It sends pre-commit message to all receiver sites and was stored in multiple site. However, using a lot message transference is hindrance to the efficiency of the network since each sending site sends message to all receiving site. This exceeded the communication problem that was criticized to the 3PC protocol. Also the algorithm might not have backup coordinator that could minimize the possibility of encountering any blocking problem. This algorithm is an enhancement of 2PC protocol added with pre-commit message thus suggesting problems associated with 3PC such as multiple site failures are still existing.

Teresa [10] proposed an improved failure recovery algorithm for overcoming 2PC blocking problem. Though the main trigger of 2PC blocking is the failure of a coordinator and one of the participating sites is in ready state for committing the transaction, Teresa used a backup coordinator with reduced communication overhead. The algorithm was simulated in MYSQL server as back end server and JCRETOR as front end. The algorithm revealed it consisted of only operation according to Teresa. It had sub transactions partially for inserting, updating and others for error-handling. These sub transactions are executed at the same time. Hence they fail as a group and rollback as group. However, what if the backup coordinator itself failed? This is a question not answered yet and it is not known its consequence on the proposed algorithm.

Teresa [10] proposed another algorithm named a clustering algorithm in 2PC protocol for overcoming distributed transaction failures. Teresa made a simulation to investigate how the algorithm handles site failures using Kisii, Nairobi and head office as replicated sites. The table consisted of five columns; customer ID, Customer Name, Address, City and AccountBalance. The Nairobi of the sites was intentionally modified one of the its columns form AccountBalance to AccountBalance1 and the algorithm was re-run. The transaction was marked as roll back, the two sites did not vote a TRANSACTION_COMMIT. The algorithm ensured the atomicity requirement of the transaction that should be committed as cluster otherwise rolled back.

However, it is unclear how the algorithm handles coordinator failure. Though site failure was simulated and the algorithm handled it successfully, no one knows how it will overcome the blocking problem related to the 2PC protocol when the coordinator fails. Also algorithm redesign is needed so that explicit transaction commit message sent by the other participants, knowing they haven’t voted and the coordinator had rolled back the transaction.

Having mentioned that 2PC has blocking problem, a number of algorithms were proposed. The 3PC protocol has introduced and added an extra phase to the 2PC protocol to overcome the blocking problem. This brought communication problems and not dealt well with the multiple site failure. A clustering 2PC protocol algorithm was designed that transaction should be committed or rolled back as clusters. This algorithm minimized site failure but not the coordinator failure. Teresa proposed a backup coordinator as means of recovering failure. Kumar and his colleagues [8] proposed an algorithm that is working in 2PC protocol and runs 3PC protocol during failure. In spite of different methods proposed to overcoming 2PC blocking problem, there is a gap in the literature yet to be filled. There is hybrid method, in other words, combined method for overcoming 2PC protocol. In our paper, we are proposing this hybrid. We are combining Kumar algorithm and Teresa's algorithm to overcome the problem in a combined form rather a single method. Therefore, this makes it necessary to fill the gap left by the current existing algorithms. That is why we are proposing an alternative model for overcoming 2PC blocking problem by redesigning the algorithm.

4. Tools and methods

We conducted simulation of the alternative model to measure the performance and efficient of the proposed algorithm. We used MYSQL AND Eclipse IDE to provide us the simulation environment. We used JTA (Java Transaction API) as coordinator of transactions in distributed environment with the help of the open source Bitronix transaction manager as the implementation of JTA. The experiment also includes JDBC driver for establishing connections with MYSQL server.

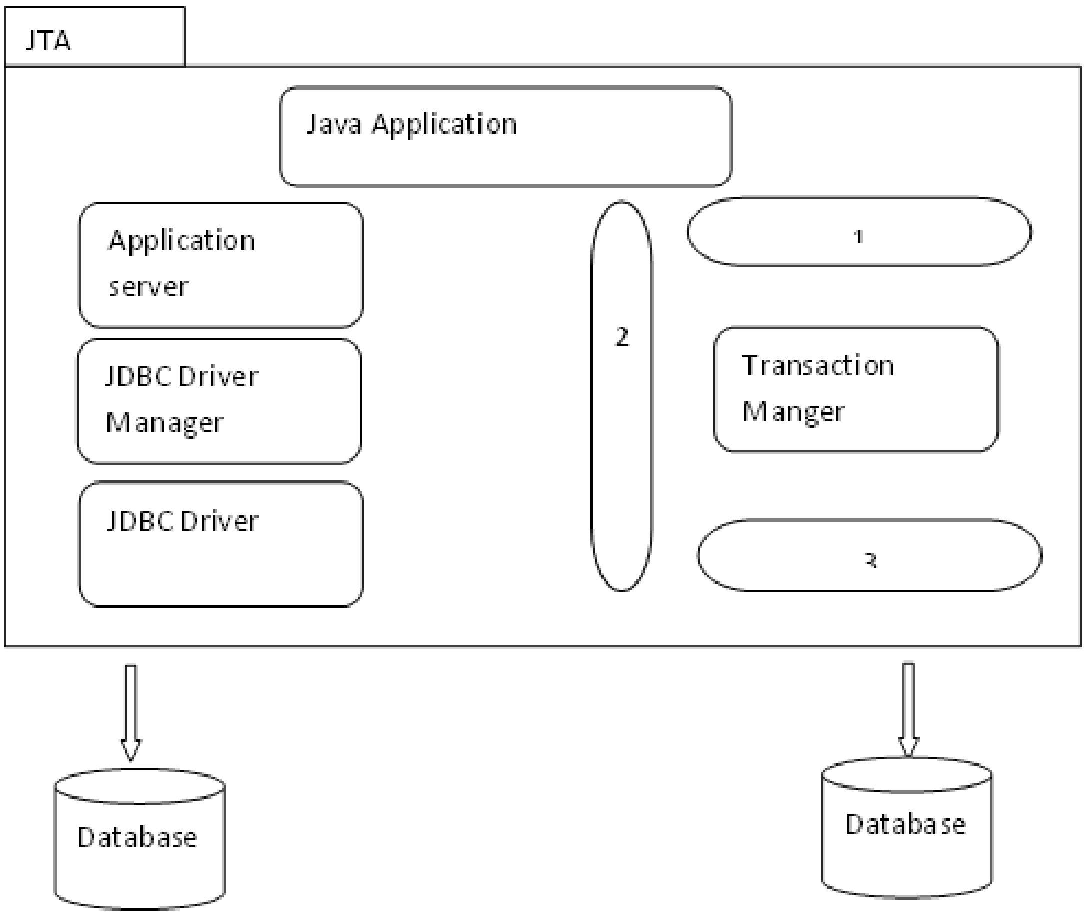


Figure 4: JTA architecture:"(understanding JTA")[11]

4.1. flowchart of proposed algorithm

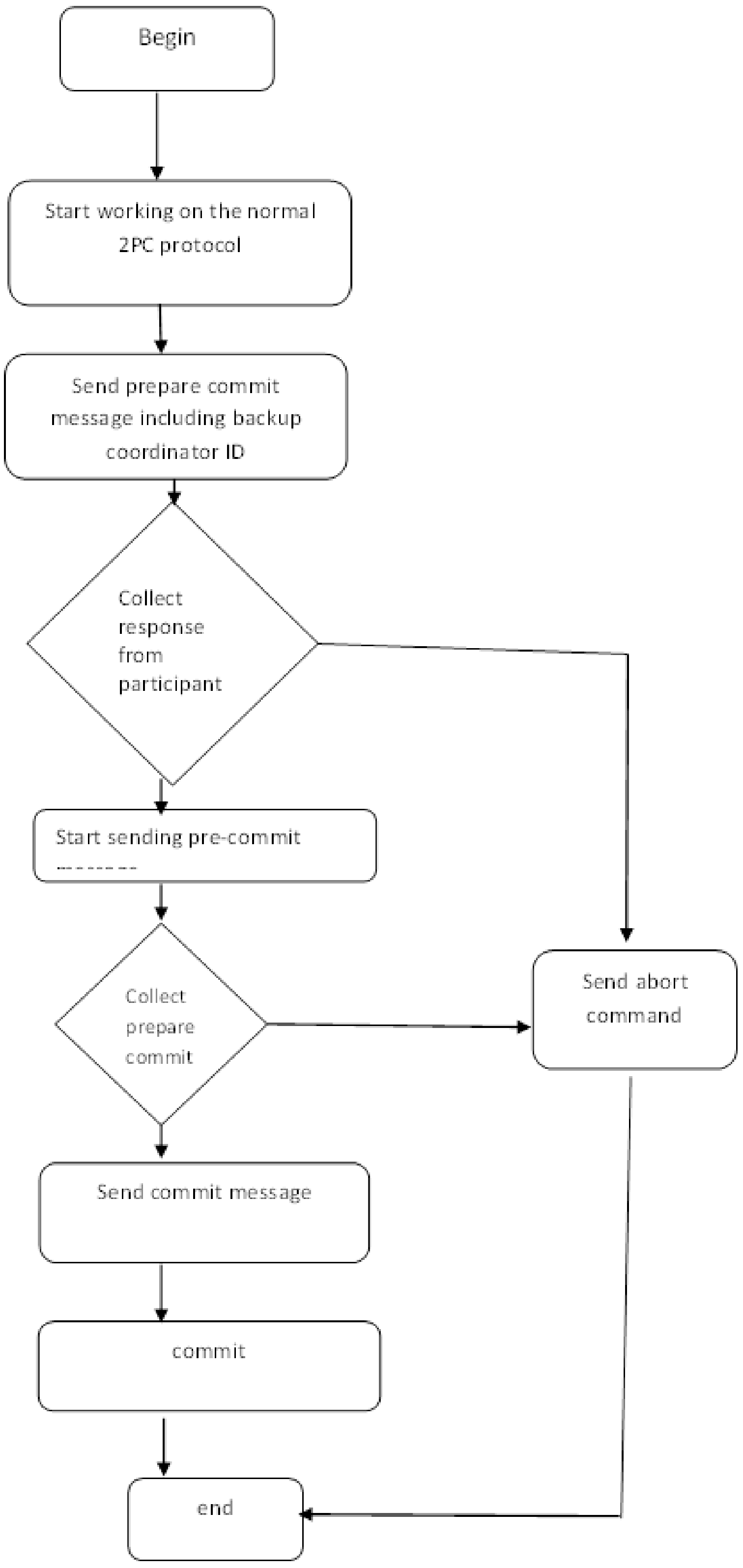

Begin
Start working on the normal 2PC protocol
Send prepare commit message including backup coordinator ID
Collect response from participant
Start sending pre-commit --------
Collect prepare commit
Send abort command
Send commit message
commit
end

Figure 4: Flow chart of the proposed algorithm (an alternative model)

4.2. Simulation Component

The simulation we have conducted in this experiment comprises of the following components:

- Resource manager: each database has its own resource manager for executing transactions locally as coordinated by the transaction manager i.e., Bitronix transaction manager
- Participants: there are site MYSQL SERVERS namely, MYSQL SERVER 1 in site1, MYSQLSERVER 2 in site 2, and MYSQL SERVER 3 in site 3 as distributed database.
- Bitronix transaction manager is the open source transaction manager for managing transaction from different resource manager in each MYSQL SERVER. The version of Bitronix transaction manager btm.2.1.3.jar.
- Resource adaopter: mysql-java-connector is the JDBC driver for connecting to MYSQL SERVERS.
- Database connection do the role of establishing connections to each of MYSQL SERVER connection 1, connection2, connection3 respectively
- Maven for managing dependencies such as MYSQL Connector jar and BTM jar.

Connection con1 =DriverManager.*getConnection*("jdbc:mysql://localhost:3306/site1-db","root","root");

Connection con2 =DriverManager.*getConnection*("jdbc:mysql://localhost:3306/site2-db","root","root");

Connection con2=DriverManager.*getConnection*("jdbc:mysql://localhost:3306/site3-db","root","root");

The picture below depicted the overall simulation

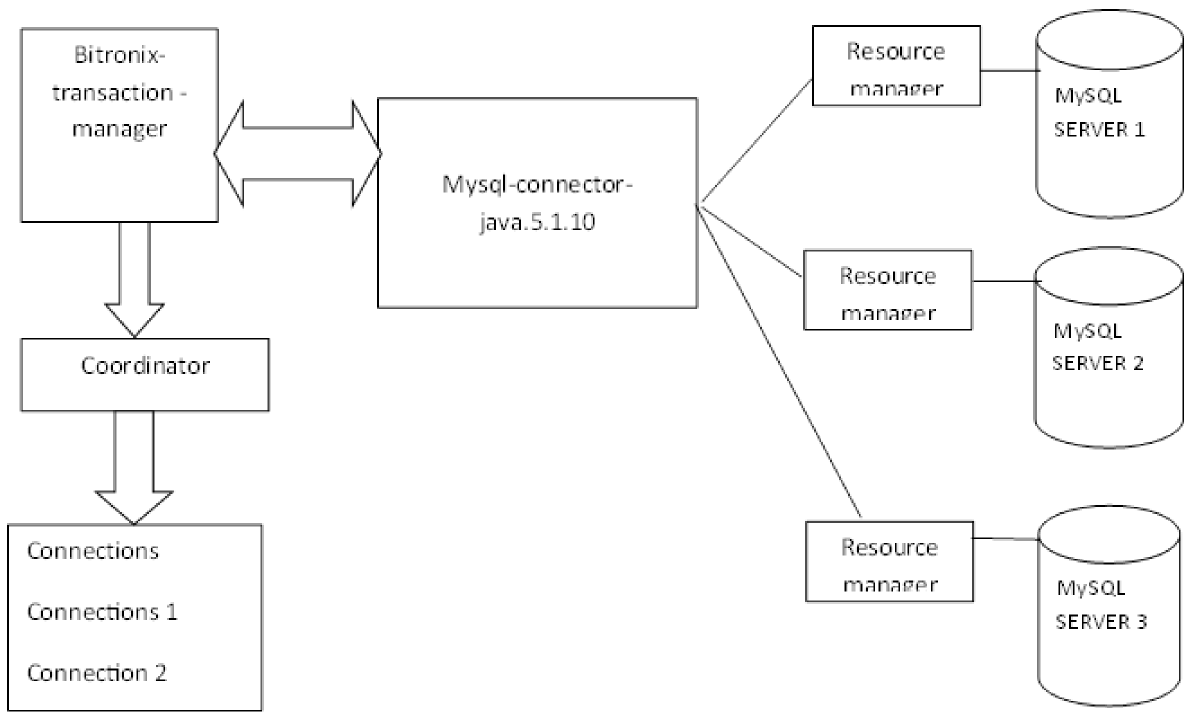


Figure 5: Simulation Architecture

4.3.2. PC Coordinator Failure Simulation

To show 2PC site failure, the figure below shows an output adopted from Teresa's work of clustering algorithm in two-phased commit protocol for optimizing distributed transaction failure. The output clearly shown coordinator failure as the figure below indicated

```
"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.startRe
covery(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.jav
a:138)"
```

Figure 6: 2PC Site Failure. Teresa [10]

The snippet above has given Bitronix Transaction file log details while conducting 2PC coordinator failure. The Bitronix Transaction manager services was initialized and assigned a unique ID with localhost address of the MYSQL Server. TwopcCoordinator Class representing main class of java has failed and It was marked as resource failure in the output of the log file. *bitronix.tm.recovery.RecoveryException was raised*

4.4. Pseudo Code of Algorithms

In this part, we have shown pseudo code of algorithms namely, 2PC, 2PC using back up coordinator, and the alternative model-the proposed algorithm. We clearly pinpointed coordinator failure and the backup coordinator failure and how the modified algorithm overcame failures. The bold text inside pseudo code shows changes made and differences between algorithms

4.4.1. Pseudo code of 2PC without backup coordinator

```
In phase 1: Voting Phase

Begin
```

Coordinator sends prepare message

Participants receive prepare message by sending Ready Message “YES”

Coordinator collects all the RM

If one of the participants replied “NO” or Times out

Coordinator Sends GLOBAL ABORT message by locally aborting the transaction

All participants also abort the transaction

In phase 2: Decision Phase

Coordinator Receive RM “YES” from the participants and send Commit Message CM GLOBAL COMMIT MESSAGE

If one of the participants fail it checks with the coordinator upon recovering from failure

Participants receive CM from coordinator by replying acknowledge message and locally committing transaction

Coordinator commits transaction

If coordinator fails when sending CM message and non-participant receive,

Alive participant cannot make a decision and the algorithm gets in stuck.

End of algorithm

4.4.2. Pseudo code of 2PC with backup coordinator

There is no much difference between the algorithms apart from writing transaction logs in a single coordinator alone, we also use back up coordinator in case of failures.

In phase 1: Voting Phase

Begin

Coordinator sends prepare message with back up coordinator ID

Participants receive prepare message by sending Ready Message “YES”

Coordinator collects all the RM

```
If one of the participants replied "NO" or Times out

Coordinator Sends GLOBAL ABORT message by locally aborting the transaction

All participants also abort the transaction

End of Algorithm

In phase 2: Decision Phase

Coordinator Receive RM "YES" from the participants and send Commit Message CM GLOBAL COMMIT MESSAGE

If one of the participants fail it checks with the coordinator upon recovering from failure

Participants receive CM from coordinator by replying acknowledge message and locally committing transaction

If coordinator and backup coordinator fails when sending CM message and non-participant receive,

Alive participants cannot make a decision and the algorithm gets in stuck

End of Algorithm
```

4.4.3. Pseudo code of the Proposed algorithms using hybridization

In this pseudo code, we have shown the pseudo code of the algorithm clearly stating how this algorithm bypasses blocking problem caused by coordinator failure using hybridization, combination of back up coordinator and 3PC. Both coordinators and participants register locally pre-commit with no broadcast or sending to coordinator or participant as a means of reducing communication overhead associated with 3PC. The bold text in the code modification made in the previous pseudo code. As its predecessor, it consists of two phases namely, phase I and phase II. The pseudo code of the algorithm is shown below.

```
In Phase I: Voting Phase

Start of the algorithm

Coordinator sends prepare message with back up coordinator ID

Participants receive prepare message by sending Ready Message "YES"

Coordinator collects all the RM

If one of the participants replied "NO" or Times out
```

Coordinator Sends GLOBAL ABORT message by locally aborting the transaction

All participants also abort the transaction

In phase 2: Decision Phase

Coordinator Receive RM “YES” from the participants and before send message it locally registers Pre-Commit state and send Commit Message CM GLOBAL COMMIT MESSAGE.

If one of the participants fail it checks with the coordinator upon recovering from failure

Participants register pre-commit state upon receive CM from coordinator by replying acknowledge message and locally committing transaction

If coordinator *and backup coordinator fails* when sending CM message and non-participant receive,

Alive participant start communicating to each other to check for pre-commit state

If any participant has locally registered pre-commit state, they commit the transaction

If non-participant has registered pre-commit state, they abort the transaction.

End of the algorithm

4.4.5. Simulation of Coordinator and Backup Coordinator Failure

To demonstrate coordinator and backup coordinator failure, we ran the simulation by making insertion to the student table created in the three database sites namely site1-db, site2-db, and site3-db. The data tried to insert was the following:

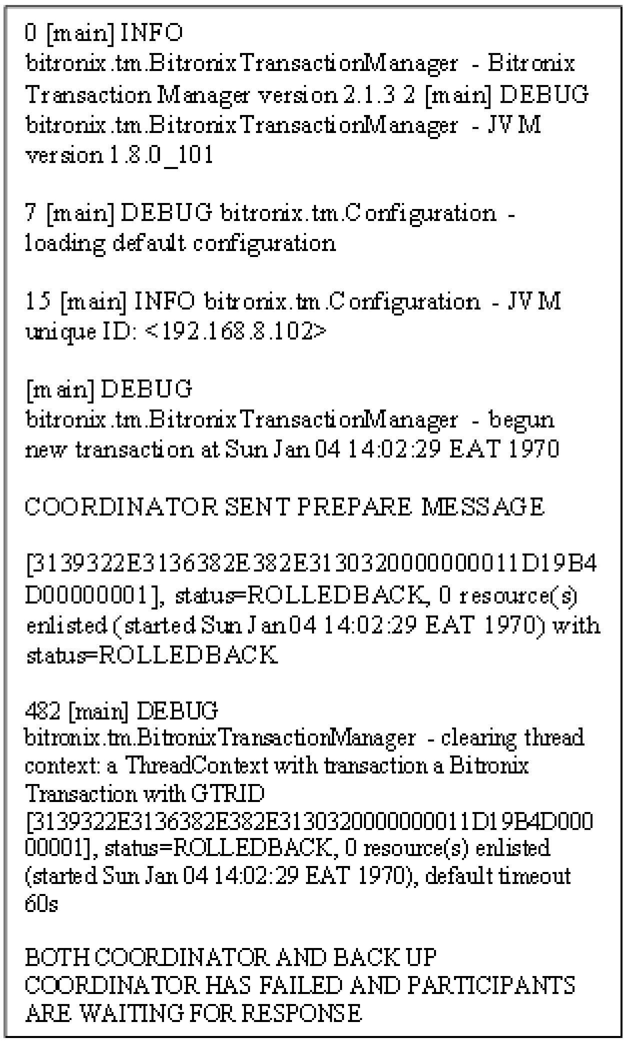
Table 1: Data Inserted In Three Databases

ID	NAME	ADDRESS	GENDER	DOB
1	HASSAN	MOGADISHU	MALE	1988

After running, the Bitronix output in the console window of Eclipse IDE appeared like this

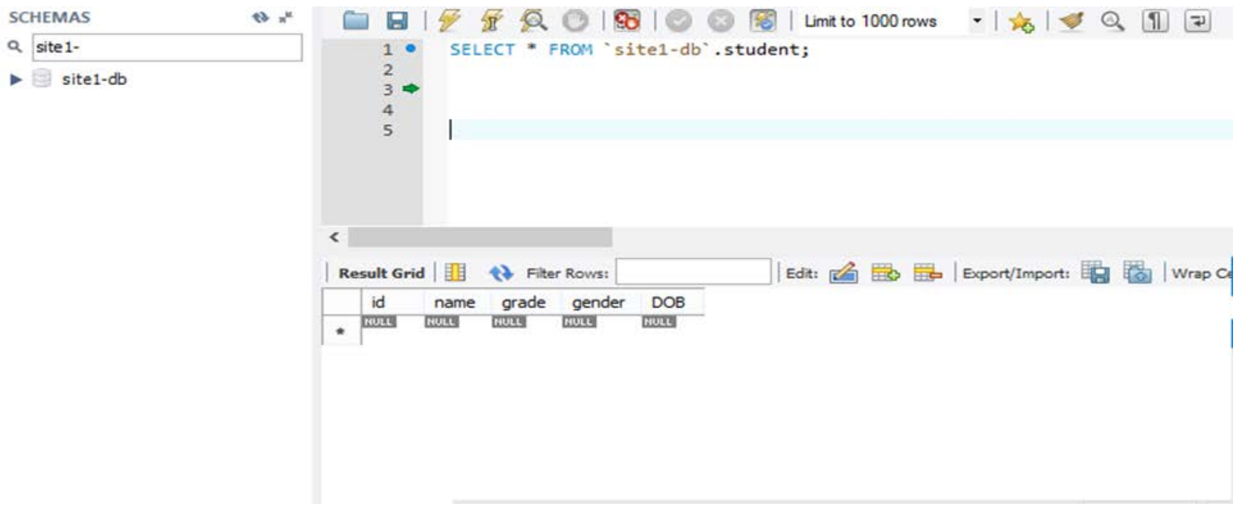
As shown above, Bitronix Transaction manager started beginning, and the coordinator and backup coordinators were coordinating transaction between different sites of the database. There happened a serious communication error where both coordinators have failed. Then, the participants

were forced to wait a response from the coordinators thereby getting in blocking sate as the above output depicted. No rows were affected. The details of the student table in three databases were shown below.



```
0 [main] INFO
bitronix.tm.BitronixTransactionManager - Bitronix
Transaction Manager version 2.1.3 2 [main] DEBUG
bitronix.tm.BitronixTransactionManager - JVM
version 1.8.0_101

7 [main] DEBUG bitronix.tm.Configuration -
loading default configuration

15 [main] INFO bitronix.tm.Configuration - JVM
unique ID: <192.168.8.102>

[main] DEBUG
bitronix.tm.BitronixTransactionManager - begun
new transaction at Sun Jan 04 14:02:29 EAT 1970

COORDINATOR SENT PREPARE MESSAGE

[3139322E3136382E382E3130320000000011D19B4
D000000001], status=ROLLEDBACK, 0 resource(s)
enlisted (started Sun Jan 04 14:02:29 EAT 1970) with
status=ROLLEDBACK

482 [main] DEBUG
bitronix.tm.BitronixTransactionManager - clearing thread
context: a ThreadContext with transaction a Bitronix
Transaction with GTRID
[3139322E3136382E382E3130320000000011D19B4D000
00001], status=ROLLEDBACK, 0 resource(s) enlisted
(started Sun Jan 04 14:02:29 EAT 1970), default timeout
60s

BOTH COORDINATOR AND BACK UP
COORDINATOR HAS FAILED AND PARTICIPANTS
ARE WAITING FOR RESPONSE
```

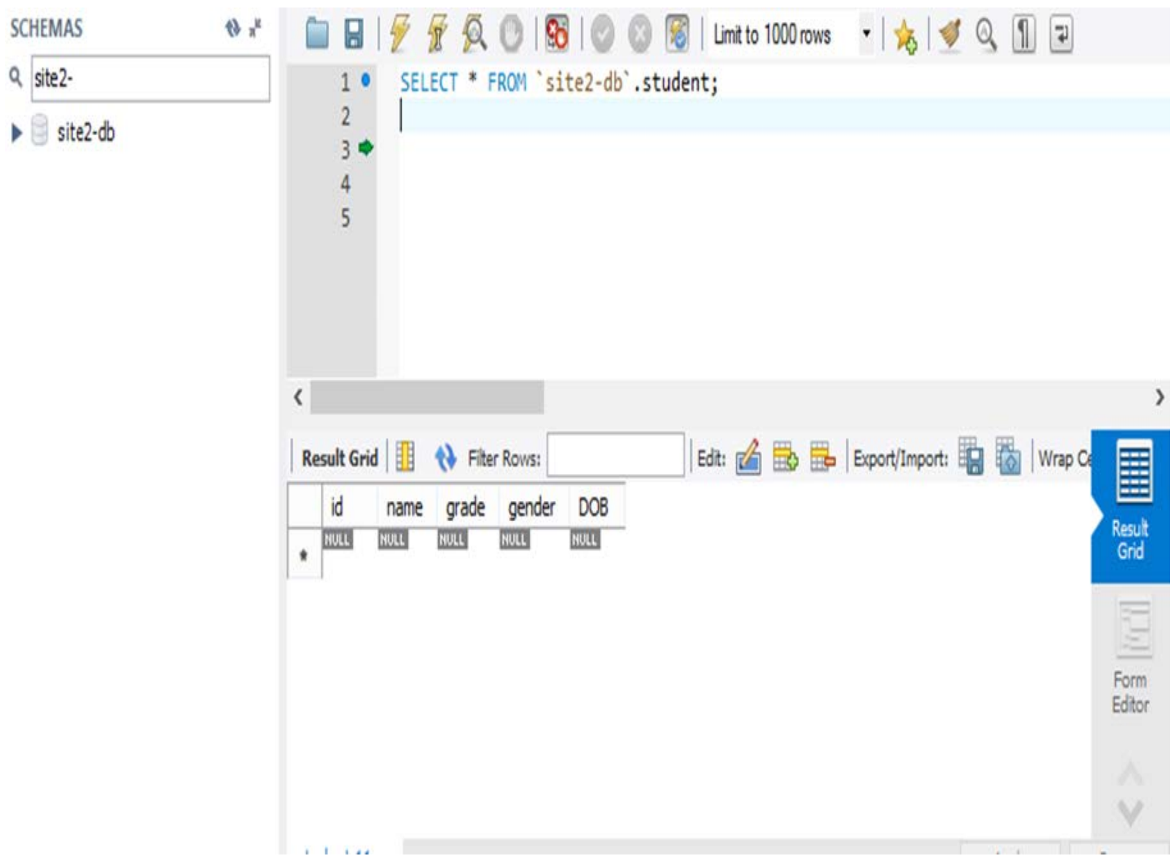
Figure 7: Bitronix output in the console window of the eclipse IDE.



```
SELECT * FROM `site1-db`.student;
```

	id	name	grade	gender	DOB
*	NULL	NULL	NULL	NULL	NULL

Figure 8: Status of site1-db after running the simulation: Console log of MYSQL Workbench



```
SELECT * FROM `site2-db`.student;
```

	id	name	grade	gender	DOB
*	NULL	NULL	NULL	NULL	NULL

Figure 9: Status of site2-db after running the simulation: Console log of MYSQL Workbench

4.4.6. Simulation of the proposed algorithm

To demonstrate how the alternative model overcame blocking problem, we run the simulation using the same insertion scenario we did in the simulation of the coordinator above. After running, the Bitronix Transaction log displayed output below.

As shown in the figure above, the same failure has occurred i.e., both coordinator and backup coordinator has failed. Instead participants get stuck in blocking state, they made communication by accessing each site’s status of pre-commit state as we have in the pseudo code. As shown in the figure above, the same failure has occurred i.e., both coordinator and backup coordinator has failed.

Instead participants get stuck in blocking state, they made communication by accessing each site’s status of pre-commit state as we have in the pseudo code of the proposed algorithm.

It was stating that the algorithm should run to 3PC in case of failure. As evident in the output above, one of the participants has not reached pre-commit before coordinators has failed and the participants unanimously agreed to abort the transaction.

```
1 [main] INFO bitronix.tm.BitronixTransactionManager  -
Bitronix Transaction Manager version 2.1.3

3 [main] DEBUG bitronix.tm.BitronixTransactionManager
- JVM version 1.8.0_101

8 [main] DEBUG bitronix.tm.Configuration  - loading
default configuration

27 [main] INFO bitronix.tm.Configuration  - JVM unique
ID: <192.168.8.102>

514 [main] DEBUG bitronix.tm.BitronixTransaction  -
transaction status is changing from ROLLING_BACK to
ROLLEDBACK - executing 0 listener(s)

514 [main] DEBUG bitronix.tm.BitronixTransaction  -
successfully rolled back a Bitronix Transaction with GTRID
[3139322E3136382E382E313032000000002A4D6C10000
00001], status=ROLLEDBACK, 0 resource(s) enlisted
(started Fri Jan 09 08:08:37 EAT 1970)

514 [main] DEBUG
bitronix.tm.internal.XAResourceManager  - clearing
XAResourceHolder states on 0 resource(s)

515 [main] DEBUG bitronix.tm.BitronixTransaction  - after
completion, 1 synchronization(s) to execute

517 [main] DEBUG bitronix.tm.BitronixTransaction  -
executing synchronization a ClearContextSynchronization
for a Bitronix Transaction with GTRID
[3139322E3136382E382E313032000000002A4D6C10000
00001], status=ROLLEDBACK, 0 resource(s) enlisted
(started Fri Jan 09 08:08:37 EAT 1970) with
status=ROLLEDBACK

BOTH COORDINATOR AND BACK UP COORDINATOR
HAS FAILED

a live participants stated communicating each other

Site 3 has not reached pre-commit and transaction was aborted
by the alive participants
```

Figure 10: Bitronix output in the console of Eclipse IDE after running the alternative model

5. Conclusion

we conducted a research in the distributed algorithms specifically the commonly used 2PC protocol. The main drawback related to this algorithm was blocking problem that occurs when participants are in the 2nd phase of the algorithm and the coordinator has failed. The participants were forced to wait until the coordinator recovers thereby leading them to get in stuck and blocked. To overcome this, a number of algorithms were proposed among them is the clustering algorithm, 3PC, non-blocking 2PC, and improved failure recovery algorithm.

However, each has its own limitations as explained in the literature review. Therefore, in this study, we proposed an alternative model to overcoming 2PC blocking problem by combining two techniques the use of backup coordinator adapted from TRESA'S work of improved failure recovery algorithm and the technique whereby the algorithm runs to 3PC in case of failure. This part adapted from Kumar and his colleagues [8]. To reduce communication issues, we modified the algorithm thereby not sending pre-commit message at the same time each participant has declared and registered locally it has reached pre-commit state.

The proposed algorithm i.e., the alternative model has achieved its objectives of the study. We proposed an alternative model using hybridization. We also evaluated and studied existing algorithms overcame 2PC blocking problem. Though we were combining two algorithms we reduced communication overhead. Lastly we have assessed the performance of the alternative model by comparing other algorithms in terms of failure resilience and communication cost. The alternative model has outperformed existing 2PC algorithm.

6. Recommendation

Failure was a common problem in 2PC whether it is a site or coordinator. To address this, different algorithms were proposed among them is the alternative to overcoming two phase commit blocking problem. we simulated the algorithm using Eclipse IDE and MYSQL Server. the algorithm combined two algorithms using backup coordinator and technique where it runs to 3PC in case of failure. The algorithm has achieved its objectives using hybridization. However, further redesign of the algorithm is recommended during failure when it runs to 3PC, so that each participating sites' states are already known to every other site instead of communicating between them. We also recommend testing the algorithm in other backend databases such as ORACLE and NoSQL database such as MongoDB.

References

[1]. H. Garcia-Molina, J. D. Ullman, and J. Widom, Database systems: the complete book. Upper Saddle River, New Jersey Pearson Prentice, 2008. pp. 1020-1021.

[2]. S. K. Rahimi and F. S. Haug, Distributed Database Management Systems: A Practical Approach. John Wiley & Sons, 2010. pp.185.

[3]. L. Lou, F. Tang, I. You, M. Guo, Y. Shen, and L. Li, "An Effective Deadlock Prevention Mechanism for Distributed Transaction Management," 2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, 2011.

[4] S. Tanenbaum and M. van. Steen, Distributed Systems: Principles and paradigms, Second edition. Upper

Saddle River, New Jersey: Pearson Prentice Hall, 2007. pp 355-363.

[5]. Lannhult, & B. Lindblom, "Review of Non-Blocking Two-Phase Commit Protocols. Master Program in Computer System Engineering", Emausgatan C, 29. Available: http:// http://www.idt.mdh.se/kurser/ct3340/ht09/ADMINISTRATION/IRCSE09-submissions. [Mar 5, 2016].

[6] M. Atif, "Analysis and verification of Two-Phase Commit & Three-Phase Commit Protocols," in proc. DOI: 10.1109/ICET.2009.5353152. Dec 2009, pp. 326-331. Available: http://ieee.exolpre.org [12 march, 2016].

[7] P. Singh, P. Yadav, A. Shukla, and S. Lohia, "An Extended Three Phase Commit Protocol for Concurrency Control in Distributed Systems," International Journal of Computer Applications, vol. 21, no. 10, pp. 35–39, 2011. Available: http://www.ijcaonline.org/archives/volume21/number10/2596-3608 [Jan 15, 2016]

[8] V. Manikandan, R. Ravichandran, R. Suresh, & F. S. Francis,. "An Efficient Non-Blocking Two Phase Commit Protocol for Distributed Transactions". International Journal of Modern Engineering Research, Vol.2, Issue.3, pp. 788-791 may 2012. Available: http://www.ijmer.com. [Feb 25, 2016].

[9] N. Kumar, L. Sahoo, and A. Kumar, "Design and implementation of Three Phase Commit Protocol (3PC) algorithm," 2014 International Conference on Reliability Optimization and Information Technology (ICROIT), pp. 116–120, 2014.

[10] T. K. Abuya, R. M. Rimiru, & W. K. Cheruiyot, "AN IMPROVED FAILURE RECOVERY ALGORITHM IN TWO-PHASE COMMIT PROTOCOL FOR TRANSACTION ATOMICITY". Journal of Global Research in Computer Science, vol.5, Issue.12, pp 01-11, Dec 2014. Available: http://www.jgrcs.com. [Feb 20, 2016].

[11] T. K. Abuya, R. M. Rimiru, & W. K. Cheruiyot, "A Clustering Algorithm in Two-Phase Commit Protocol for Optimizing Distributed Transaction Failure". International Journal of Computer Science and Mobile Computing, vol.4, Issue.3, pp 97-106, Mar 2015. Available: http://www.ijcsmc.com [Mar 20, 2016].