

# Concurrency Control Technique for Transaction Processing In Distributed Database System Using Hybridizedmodel

Gabriel, Bariyira Christopher<sup>1</sup>; Nathaniel Ojekudo; Gabriel, Meelubari Napoleon;<sup>3</sup>

<sup>1</sup>Computer Science Department, Ignatius Ajuru University of Education, Port Harcourt, Nigeria

<sup>2</sup>Computer Science Department Ignatius Ajuru University of Education, Port Harcourt, Nigeria

Submitted: 25-07-2021

Revised: 04-08-2021

Accepted: 06-08-2021

**ABSTRACT**— Concurrency control has been actively investigated for the past several years and two-phase locking used as a standard solution for transaction processing in a database management system but the system is saturated with conflicts due to frequent rollbacks, long time waiting and blocking, high rate of aborted transactions that leads to deadlock. Hence to solve these problems, this paper presented a hybridized concurrency control technique which combines two phase locking and timestamp ordering. This technique will enhance the performance of the concurrency control techniques for transaction processing in a distributed database management system. This work is developed in ASP.Net using C# programming language as the front end and Microsoft SQL Server 2008 as the back end that is the database. Our result shows that two-phase locking and timestamp ordering is slow, time consuming during execution while our hybridized approach optimizes faster and consumed less time. This approach will give confidence to database administrators when handling concurrency transactions and make them deliver their work timely.

**Keywords**— Concurrency Control, Distributed database, Transaction processing, Distributed Processing, Timestamp ordering, Two phase locking

## I.INTRODUCTION

In this decade, the world is increasingly relying on information systems. This increase in reliance means increased use of client/server applications and distributed databases. A distributed database is a single logical database that is dispersed across numerous computers that are connected via communication channels[1]. A distributed database can be thought of as a virtual database with its component pieces stored in

various locations physically. In a multi-user database management system (DBMS), concurrency control is simply coordinating the concurrent visits to the database. Concurrency control is a circumstance where users of a database can access a database on a dedicated system, provided each user is operating on their own system[2]. When a user connects to a database, they are said to be executing a transaction. The transaction management keeps the transaction running smoothly in the database management system. Unlike transactions in other database systems, database transactions are database processes like sequential data read/write activities on database object(s) maintained in the database syste[3].

Decades of study have produced a large body of knowledge on concurrency control, and non-distributed Data Base Management Systems face no significant challenges. Two-phase locking has been established as a conventional solution to the problem[2].

A number of conflicts arise when transaction processing is done in a distributed database management system with two-phase locking. Due to transaction execution time waiting, frequent rollbacks of transactions, and aborted transaction rates, several issues exist. Due to stalemate with locking and blocked transaction storage overhead rises, transactions were at a higher risk of becoming stranded. It is difficult to keep track of locks and queue waiting for data access, therefore processing of these costs a lot. With regard to DDBMS (distributed database management systems), the more the number of systems involved, the greater the data fragment and replication. This means that to keep the system up and running, data fragment and replication conflict must be overcome. [4].

### Statement of the Problem

It is not free from deadlock in the distributed database system when employing the two-phase locking technique. It has transaction rollbacks, transaction wait times, and transaction cancelation. Due to deadlock with locking and stopped transactions, storage overhead is raised. There is no waiting, blocking, or locking of transactions using timestamp ordering. Our solution to the aforementioned problem will be to use two-phase locking and time-stamp ordering as a kind of concurrency control, and to use this along with optimized time execution in distributed database systems to reduce transaction processing times.

### Aim and Objectives

The aim of this paper is to develop a hybridized concurrency control techniques for transaction processing using two phase locking and time-stamp ordering system. The specific objectives include;

- i. To design a system that will make an improvement in handling conflicts for concurrent transaction processing in a distributed database management system.
- ii. To develop a software that optimizes time execution for transaction processing in distributed database system.
- iii. To implement the system using C# programming language as the front-end and Microsoft SQL Server 2008 as the back-end.

## II.RELATED WORKS

### A. Two-Phase Locking Technique

In this procedure, transactions are processed sequentially, which is to say that transactions are done one after another. Two-phase locking uses serial execution to implement transaction processing.

For the purpose of this discussion, let E signify an execution of transactions T1, T2, . . . Serial execution means each transaction is finished before the next one begins. Every serial execution is declared correct, as each transaction has attributes that ensure a proper and consistent end result.

### B. Timestamp Ordering Technique

Timestamp ordering technique uses the concept of Shortest Job First (SJF) scheduling method in handling concurrent transaction, with each transaction  $T_i$  in the system, we associate a unique fixed timestamp, denoted by  $TS(T_i)$ . This timestamp is assigned by the database system before the transaction  $T_i$  starts execution. If a transaction  $T_i$  has been assigned timestamp  $TS(T_i)$ , and a new transaction  $T_j$  enters the system,

then  $TS(T_i) < TS(T_j)$ . There are two simple methods for implementing this scheme:

1. Use the value of the system clock as the timestamp; that is, a transaction's timestamp is equal to the value of the clock when the transaction enters the system.
2. Use a logical counter that is incremented after a new timestamp has been assigned; that is a transaction's timestamp is equal to the value of the counter when the transaction enters the system.

R-timestamp and W-timestamp values is associated with each data item Q:

Where:

- i. W-timestamp (Q) denotes the largest timestamp of any transaction that executed write (Q) successfully.
- ii. R-timestamp (Q) denotes the largest timestamp of any transaction that executed read (Q) successfully.

For instance; suppose that transaction  $T_i$  issue read (Q).

- a. If  $TS(T_i) < W\text{-timestamp}(Q)$ , then  $T_i$  need to read a value of Q that was already overwritten. Hence,

the read operation is rejected, and  $T_i$  is rolled back.

- b. If  $TS(T_i) \geq W\text{-timestamp}(Q)$ , then the read operation is executed, and R-timestamp (Q) is set to the

maximum of R-timestamp (Q) and  $TS(T_i)$ . Also suppose that transaction  $T_i$  issue write (Q).

- a. If  $TS(T_i) < R\text{-timestamp}(Q)$ , then the value of Q that  $T_i$  is producing was needed previously and the

system q that value would never be produced.

Hence, the system rejects the write operation and rolls  $T_i$  back.

- b. If  $TS(T_i) < W\text{-timestamp}(Q)$ , then  $T_i$  is attempting to write an obsolete value of Q. Hence, the system rejects this write operation and rolls  $T_i$  back.

### c. Distributed Database Systems

Centralized data base system (CDBS) One copy of the data is stored on a designated node. Data may be replicated over a network via fragmentation, which could be horizontal, vertical, or a mixture of the two. In terms of project and join assessments, these functions in Structured Query Language(SQL) are referred to as fragmentation, projection, and selection. User concurrent access control, deadlock detection and resolution are major issues for both types of database. An increasingly popular type of database management system (DBMS) must deal with multiple problems.

This kind of circumstance can be described as a distributed database when we have a single database that is capable of being expanded across client machines in various places that are connected by a computer network (DDB). In a distributed database, storage devices are not all connected to a single processing unit, such as the C.P.U. “Many computers within the same physical area may contain the information, or the

information may be distributed across a network of interconnected computers” [5].

To control data entry to different sites, a software known as a Database Management System (DBMS) is needed. Computers and other computer devices connect with one another through a communication network[6]. Data distribution relies on the aid of the processor to handle the retrieval of data in various nodes of a distributed database.

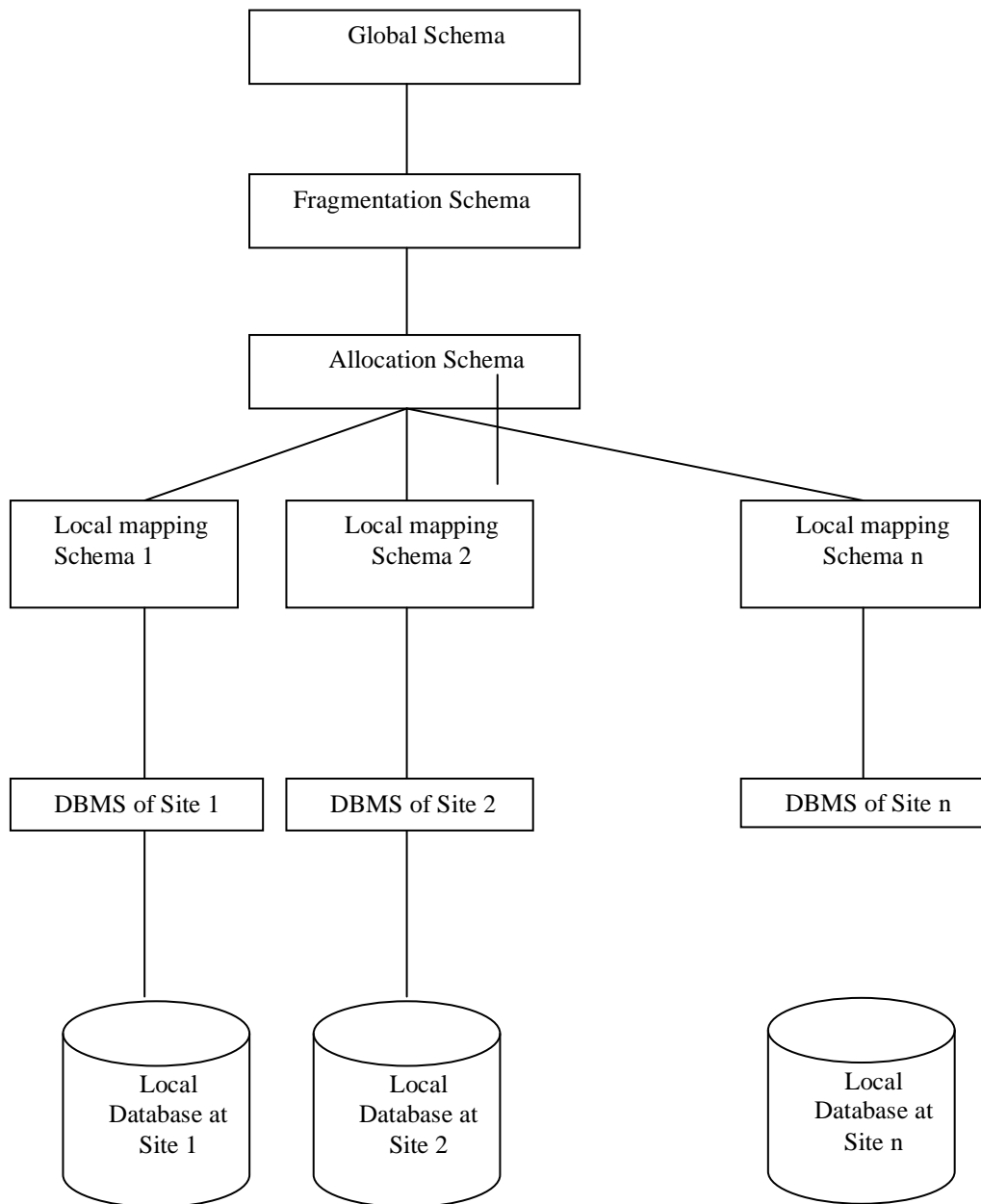


Figure 1.0: Distributed Database System Architecture

Source: Swati, Kuntal, & Bhawna(2011)

### III.METHODOLOGY

We used C# in an environment where we emulated concurrency control approaches. An MSSQL server 2008 database was set up where data are stored. Using a timer scheduler to connect to a database, we construct a simple bank application. The procedures for two consumers who want to check account balance, deposit, and make a withdrawal were shown using six (6) separate transaction processes. In the diagram above, the research system is represented as a hybridization of two formerly separate entities.

The suggested hybridized system integrates the two-phase locking mechanism with

the Thomas Write Rule, which is the present methodology for timestamp ordering. The system is being designed so that WW never causes transactions to be delayed or restarted, as is the case with the old system, where a lot of transactions are being restarted or delayed.

Write transaction may be done concurrently on the same data item by many transactions. Every saved data item must have a lock time stamp, L-ts (x). Every transaction has a timestamp called global timestamp assigned to it before execution begins. If a deadlock develops, the system swaps in another transaction to complete inside the allotted time period.

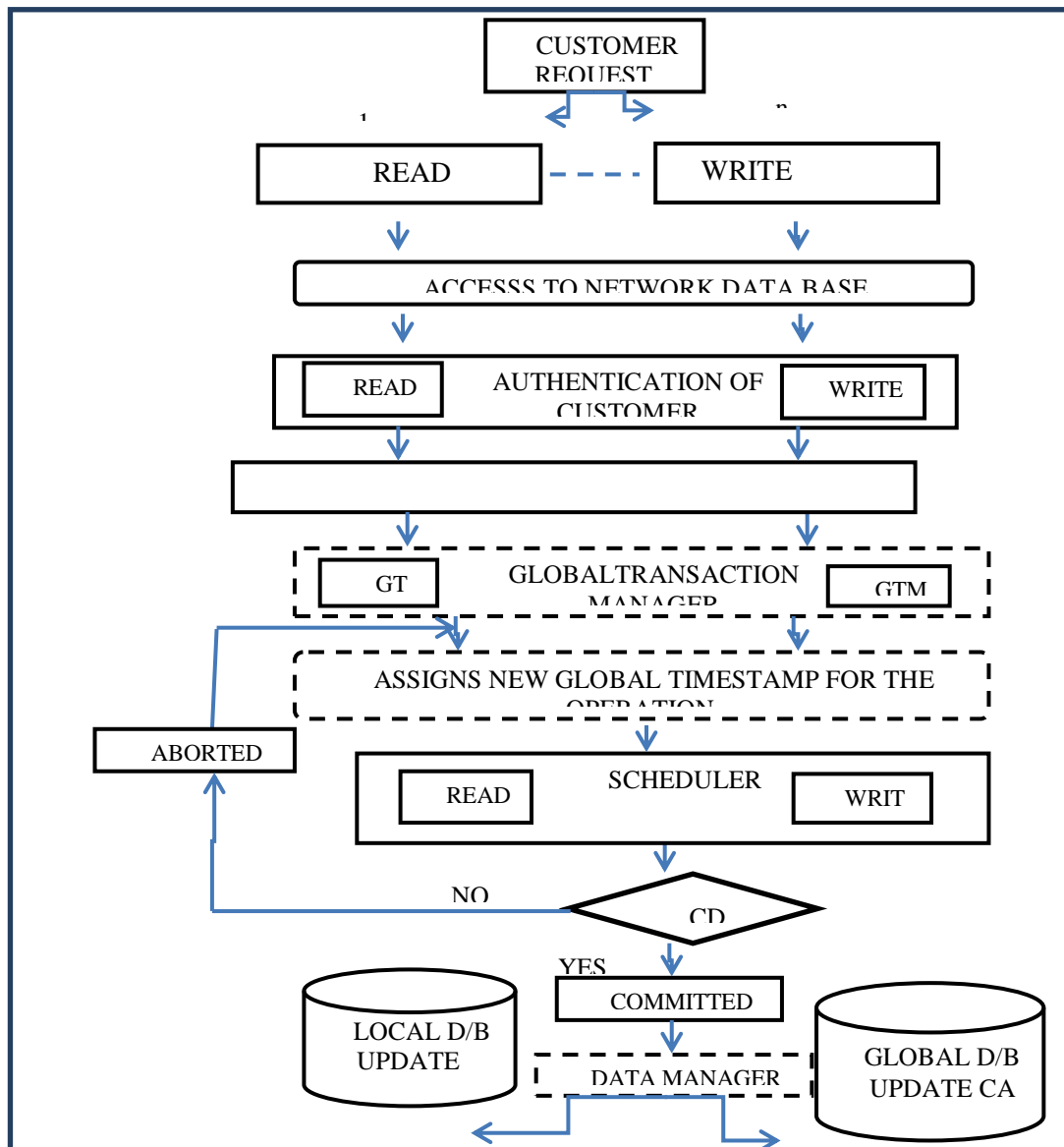


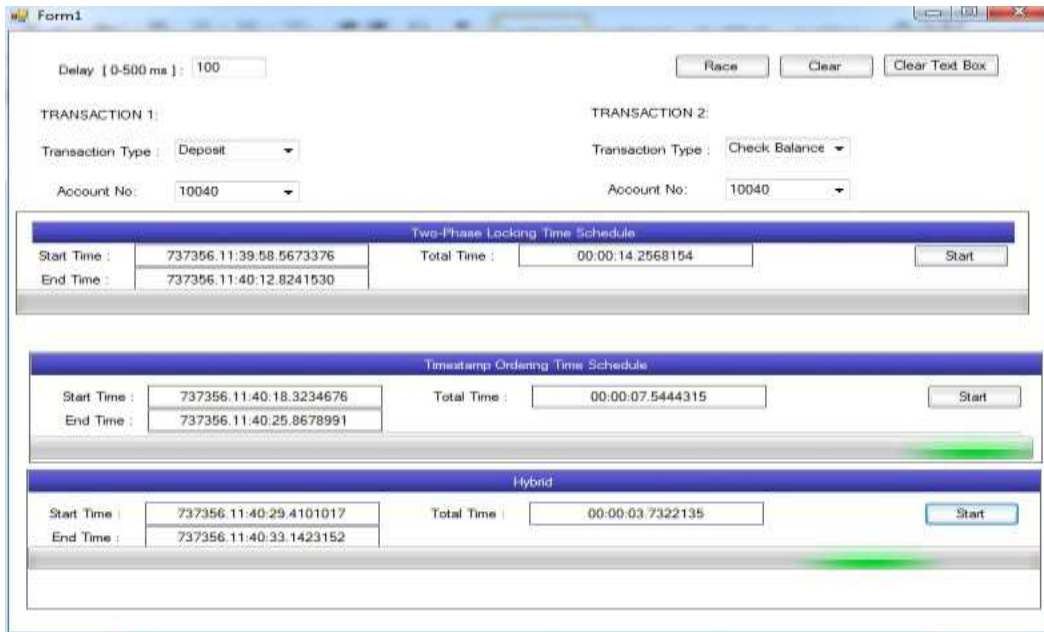
Figure2: Hybridized System

#### IV. RESULTS AND DISCUSSION

Transaction testing has been done by implementing various transaction techniques, start time, end time, total time, and the results were displayed showing that two-phase locking is very slow and time-consuming, and timestamp ordering

is superior to two-phase locking in terms of execution time.

The diagram in Figure 3 depicts the transaction for the Deposit and Checking Balance, whereas the diagram in Figure 4 depicts the transaction for the Deposit and Withdraw.

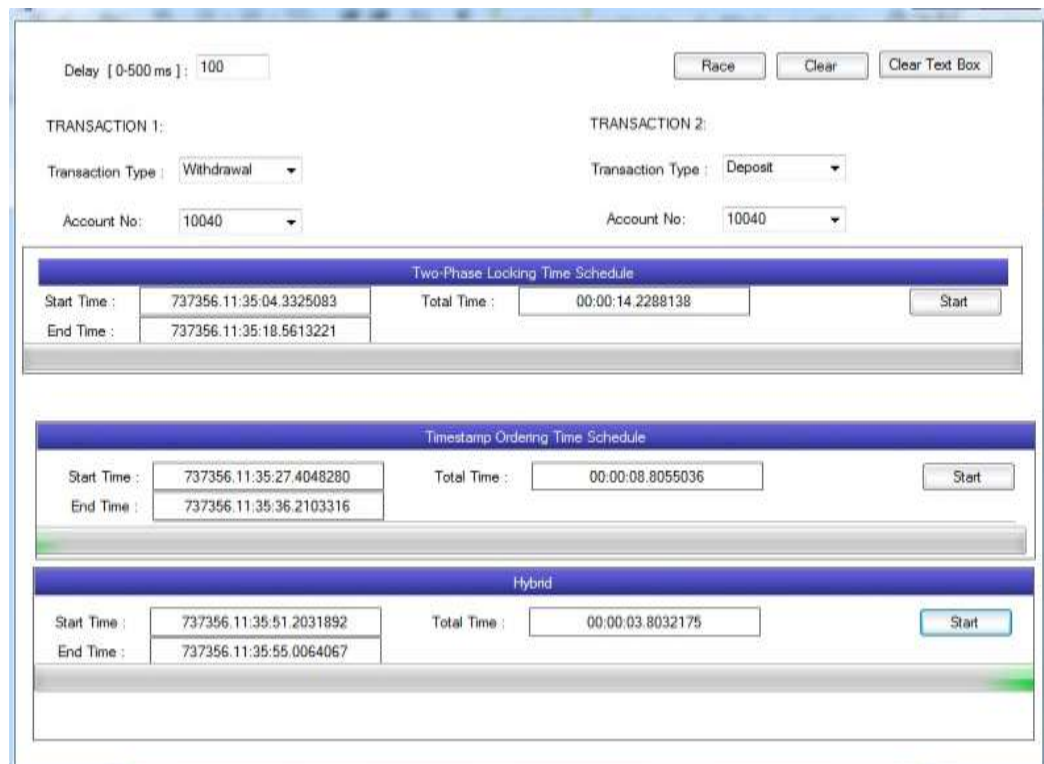


Two-Phase Locking Time Schedule		
Start Time :	737356.11:39:58.5673376	Total Time : 00:00:14.2568154
End Time :	737356.11:40:12.8241530	

Timestamp Ordering Time Schedule		
Start Time :	737356.11:40:18.3234676	Total Time : 00:00:07.5444315
End Time :	737356.11:40:25.8678991	

Hybrid		
Start Time :	737356.11:40:29.4101017	Total Time : 00:00:03.7322135
End Time :	737356.11:40:33.1423152	

Figure3: Transaction for Deposit and Checking Balance



Two-Phase Locking Time Schedule		
Start Time :	737356.11:35:04.3325083	Total Time : 00:00:14.2288138
End Time :	737356.11:35:18.5613221	

Timestamp Ordering Time Schedule		
Start Time :	737356.11:35:27.4048280	Total Time : 00:00:08.8055036
End Time :	737356.11:35:36.2103316	

Hybrid		
Start Time :	737356.11:35:51.2031892	Total Time : 00:00:03.8032175
End Time :	737356.11:35:55.0064067	

Figure 4: Transaction for Deposit and Withdraw

A timestamp has been applied to each transaction that is simultaneously accessing the database. The timestamp classifies the date into two distinct categories: local and global. For a transaction made locally, the transaction time is how long it takes for the user's server to process the transaction. Meanwhile, for a transaction initiated on the main server, the transaction time is the time required to execute the transaction and is calculated as the start time and end time. In the example, the

"Two-phase locking" consists of two components: one called "Committed" which tells whether a transaction was committed or aborted, and the other, "Transacted", which shows whether a transaction has already been conducted. An important part of the successful transaction is recorded with each and every successful transaction; these records represent the number of transactions that have been finished while others are still unfinished.

**Table1: Two-phase locking**

Transaction	Start Time (ST) in second(s)	End Time (ET) in second(s)	Total Execution Time (TT) in second(s)	Committed
T1	41	51	13	Yes
T2	36	46	12	Yes
T3	45	57	12	Yes
T4	25	38	13	Yes
T5	23	36	13	Yes
T6	22	35	12	Yes

Where;

T1= check balance, T2= Make deposit, T3=Make withdraw, T4= Deposit and withdraw, T5= Check balance and deposit, T6= check balance and withdraw.

T1 in our case is a read operation because it is a transaction that has two clients who wish to view the same account balance at the same time. When more than one consumer attempts to deposit money into a single account number at the same time, that is known as a T2 transaction. Doing an account update is called "doing an account update." T3 is a transaction that is initiated when multiple customers want to take out money from a single account number, all while a withdrawal is in progress. Updating your account's status is what that is. T4 is a transaction, which happens when multiple customers attempt to deposit and withdraw money from the same account at the same time. That is updating the financial accounts. T5 is a transaction, the first of which is performed

when multiple customers attempt to deposit money from a single account number simultaneously. Read and write operations are performed. It is a transaction where there are multiple customers all trying to withdraw and check their balance all at the same time, regardless of how many accounts are used. To accomplish these tasks, you must do the read and write operations. If you specify a start time and end time, then you may calculate the overall execution time.

The amount of transactions, or Y, in figure 3 is shown; while the total time required for executing with two-phase locking, or X, is also shown. We charted the value of time spent on process execution, X, with the value of Y in figure 2. It illustrates that the time involved in implementing the 2PL strategy is significant because of the first-come, first-served client-side mechanism used. Total execution time for transactions utilizing 2PL is shown in Figure 2.

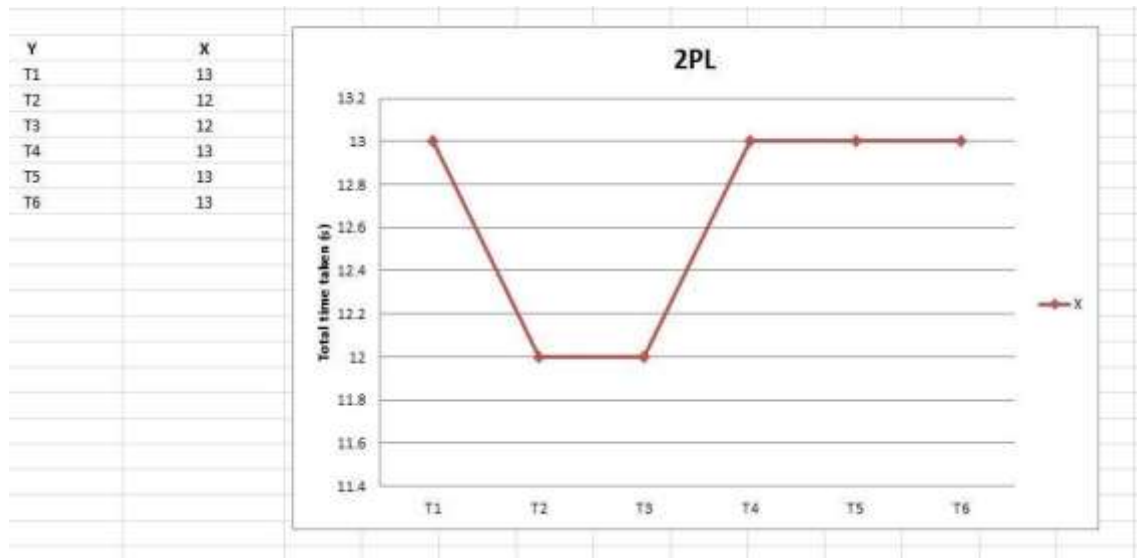


Figure3: Total execution time for transactions using 2pl

Table2 shows the Timestamp ordering with the following columns; transaction(T), StartTime(ST), EndTime(ET), TotalExecutionTime(TT) which is measured in seconds and a committed column. Table2 records

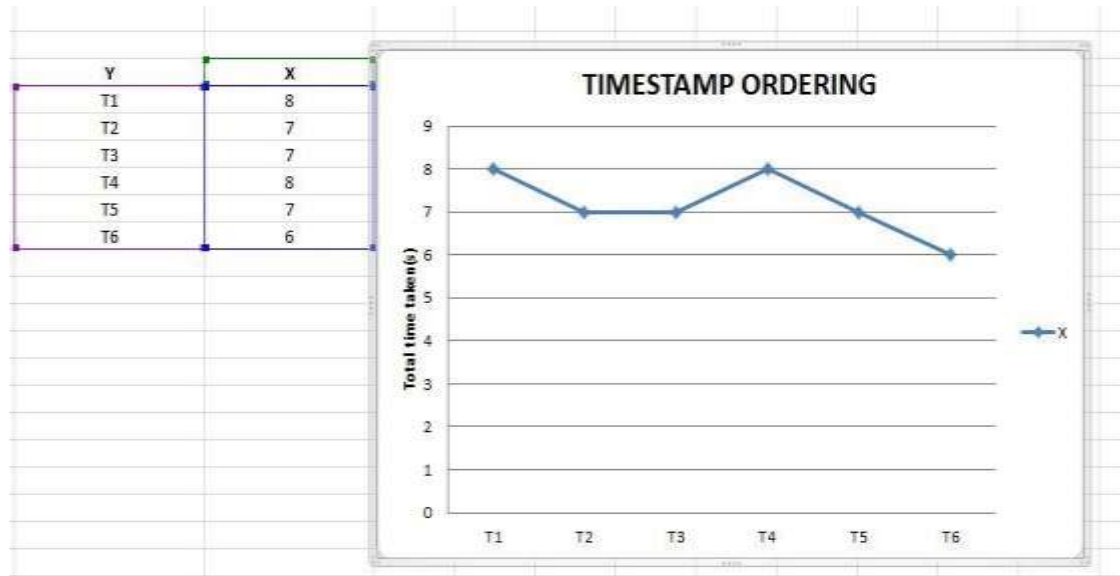
all the transaction perform by customers for read and write operation using a timestamp ordering technique. From the start time and the end time, the total time taken for execution is determined.

**Table2: Timestamp ordering**

Transaction	Start Time (ST) in second(s)	End Time (ET) in second(s)	Total Execution Time (TT) in second(s)	Committed
T1	13	21	8	Yes
T2	12	19	7	Yes
T3	14	21	7	yes
T4	34	42	8	yes
T5	42	49	7	yes
T6	55	61	6	yes

In figure4, Y is the number of transactions in a database; X is the total time taken for execution using Timestamp Ordering(T/O) which uses moderate time compare to two phase locking.

In figure3 we plotted the value of time spent on process execution against Y. It shows that Timestamp Ordering(T/O) technique is moderately time consuming unlike the 2PL.



**Figure4:** Total execution time for Timestamp Ordering(T/O) technique

Table 3 illustrates the Hybrid, which comprises columns such as transaction (T), start time (ST), end time (ET), and total execution time (TT), which is measured in seconds. In the tables below, you will find information about all of the

transactions conducted by customers for read and write operations using a hybrid approach. It is possible to calculate the total time required to execute, as long as you know the start and end times.

**Table 1.2: Hybrid**

Transaction	Start Time (ST) in second(s)	End Time (ET) in second(s)	Total Execution Time (TT) in second(s)	Committed
T1	57	63	6	Yes
T2	39	44	5	Yes
T3	38	44	6	Yes
T4	43	48	5	Yes
T5	49	56	7	Yes
T6	13	18	5	Yes

In figure 5, Y represents the number of transactions within a database, and X denotes the overall time required to carry out the execution using the Hybrid Technique, which is quicker than utilizing two-phase locking and timestamp

ordering. See the graph in figure 4, where the X-axis represents the time spent on process execution, and the Y-axis represents Y. By visualizing the performance of 2PL and T/O, we can see that Hybrid method is faster.



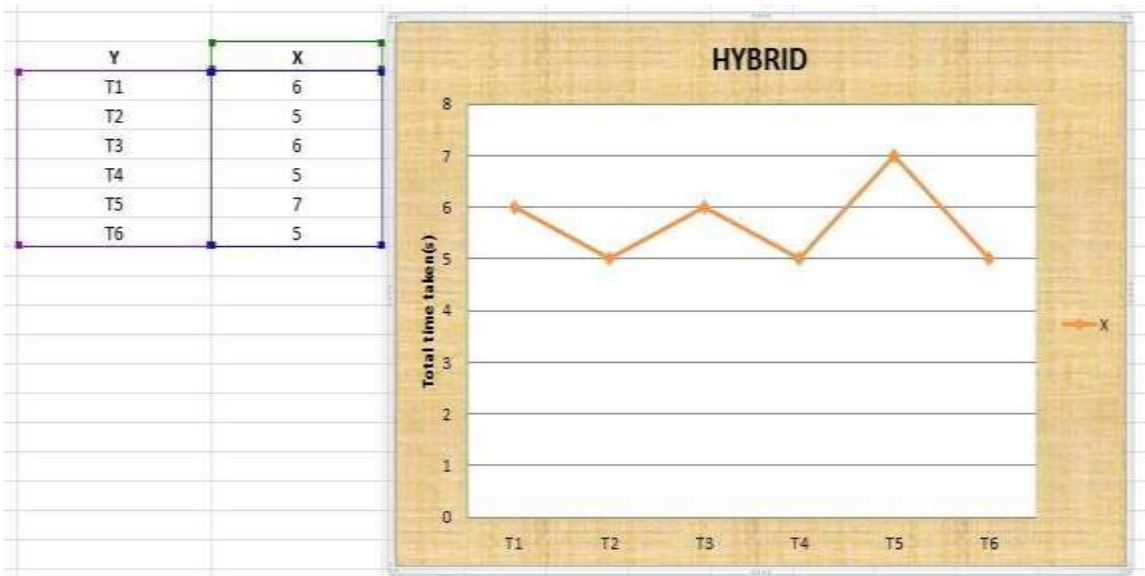


Figure5: Total execution time for transactions using hybridized technique

In table4, we describe the three (3) different technique used and then find the average execution time of each of the techniques when apply separately will be given as follows;

$$\text{Average Execution Time} = \frac{\text{Total number of execution time}}{\text{Total number of transaction}}$$

For two-phase locking:

$$\text{AET} = \frac{72}{6} = 12s$$

For timestamp ordering:

$$\text{AET} = \frac{43}{6} = 7.16$$

For hybridized:

$$\text{AET} = \frac{30}{6} = 5s$$

**Table4: Comparison of the three Techniques**

	2PL	T/O	HYBRID
Transaction	Total Execution Time (TT) in second(s)	Total Execution Time (TT) in second(s)	Total Execution Time (TT) in second(s)
T1	13	8	6
T2	12	7	5
T3	12	7	6
T4	13	8	5
T5	13	7	7
T6	13	6	7
Total = 6	76	43	34

We see that the total execution time for 2PL is lengthy, which indicates it is time-consuming for all transactions that are processed compared to timestamp ordering and hybrid, which demonstrates it is less time-consuming for transactions that use hybrid. In this case, hybrid is the most appropriate methodology for a distributed

database system that must deal with simultaneous transactions execution.

Table 5 illustrates the overall average lock/timestamp and hybridization execution time. We may conclude that the methodology that has been combined is low in complexity, while Timestamp is medium, and Two-phase is the most complex overall.

**Table5: Average Time**

Technique	Average Execution Time(AET)	Status
Two-phase locking	13.5	High
Timestamp ordering	7.16	Medium
Hybridized	6	Low

We depict X in Figure 6 as the number of transactions in a database, while Y represents the total time for two-phase locking, which is the same as X, because it's the same amount of time; as well, Y is the same as X1 because it is the same amount

of time, as well as X2, which is hybridized, which takes a lesser amount of time.

In the first graph, we show the time spent on process execution, which is represented as X, X1, and X2, versus the value of that time, represented as Y.



Figure6: shows the Total Execution Time for Transactions

### V.CONCLUSIONS

We've tackled the problem of transaction processing in distributed databases, where clients and servers on separate workstations take part. Concurrency control in distributed database is the activity of coordinating concurrent accesses to a database in a multi-user Database Management System (DBMS). The implementation of concurrency control enables programs to access a database from different programs as long as each application is executing on its own dedicated system. To speed up database design and development, we adopted an object-oriented approach and used C# for the front-end of the system, with Microsoft SQL Server 2008 serving as the relational database management system in the back end. This suggested project, "A

Hybridized Methodology for Concurrency Control in a Distributed Database System," uses a technique that can be used to improve the speed of transactions and to avoid deadlocks. The app was designed utilizing the recommended procedures in order to avoid transaction deadlock.

### REFERENCES

- [1]. M. Kaur and H. Kaur, H., "Concurrency Control in Distributed Database System". International Journal of Advanced Research in Computer Science and Software Engineering. 3(7), 2015.
- [2]. K. Arun and A. Agarwal, "A Distributed Architecture for Transactions Synchronization in Distributed Database Systems". International Journal on

- Computer Science and Engineering (IJCSE).  
2(6)1984-1991, 2010.
- [3]. A. Larson, S. Blanas, C. Diaconu, C. Freedman, M. Patel, and M. Zwillig, "High-performance concurrency control mechanisms for main-memory databases". Proceedings of the VLDB Endowment, 5(4) 298-309, 2011.
- [4]. Z. Svetlana and P. Aleksandar, "Models of 2PL Algorithms with Timestamp Ordering for Distributed Transactions Concurrency Control". International Journal of Soft Computing and Engineering (IJSCE). 3(4), 2013.
- [5]. S. Gupta, K. Saroba and K. Bhawna. (2011), "Fundamental Research of Distributed Database", International Journal of Computer Science and Management Studies, (11) 138-146, 2011.
- [6]. Y. Clement and S. Meng, (1998). Principles of Database Query Processing for Advanced Application, 1998. [7]. D. Bell and J. Grimson, "Distributed Database Systems", Reading, MA: Addison-Wesley, 1992.