2017 International Conference on Computer, Communications and Electronics (Comptelix)
Manipal University Jaipur, Malaviya National Institute of Technology Jaipur & IRISWORLD, July 01-02, 2017

Concurrency Control Methods in Distributed Database: A Review and Comparison

Mahnaz Nasseri
Department of Computer Engineering, Malard Branch,
Islamic Azad University, Tehran, Iran.
Irannasserimahnaz@gmail.com

Seyed Mahdi Jameii
Department of Computer Engineering, Shahr-e-Qods
Branch, Islamic Azad University, Tehran, Iran.
Jamei@Qodsiau.ac.ir

Abstract— **In the last years, remarkable improvements have been made in the ability of distributed database systems performance. A distributed database is composed of some sites which are connected to each other through network connections. In this system, if good harmonization isn't made between different transactions, it may result in database incoherence. Nowadays, because of the complexity of many sites and their connection methods, it is difficult to extend different models in distributed database serially. The principle goal of concurrency control in distributed database is to ensure not interfering in accessibility of common database by different sites. Different concurrency control algorithms have been suggested to use in distributed database systems. In this paper, some available methods have been introduced and compared for concurrency control in distributed database.**

Keywords-component; Distributed database, two phase locking protocol, transaction, concurrency

I. INTRODUCTION

From the past years, distributed databases have been very important in the field of researching on database. Distributed data provide chances for improving performance through simultaneous query performance and load balancing in order to expand the availability of data. In today's technological world, effective data processing is a fundamental and vital problem for almost every scientific organization. Extending an efficient distributed database system, it is necessary to propagate security [1].

It is also important to emphasize on every issues related to security such as multi-level access control, confidentiality, reliability, integrity and improving the related problems to distributed database system [2]. In general, concurrency is related to performing more than one simultaneous processing on a common database system. Concurrency controlling involves managing simultaneous operations in a database in order to prevent database access interference by 2 users [3,4,5].

Distributed database systems are the systems which their data are distributed and repeated from different places or separated sites in opposite to the concentrative databases which copy of data have been saved in. but both of them have the same problem of the concurrence access to data [1,6]. Concurrency control is a method to guide concurrency access of transactions to specific type of data in order to maintain the database constancy [5,7,8]. Constancy means that when a transaction arrives to be performed, database is in consistent status and when it leaves the system, database should be in consistent status too and also the result derived from it must be correct [9].

This problem will be complicated in distributed databases because the data are not saved in one place. The user can access the data from every site and control of mechanism might not realize it immediately in other site. In a distributed database system, a transaction might have access to saved data in more than a site [2].

Most of the distributed concurrency control algorithms are derived from 3 existed principle classes [1].

1.locking algorithms

2.timestamping algorithms

3.optimistic algorithms

Here, a look has been taken on distributed databases and how its accessibility and flexibility can be improved when accessing to different kinds of data and also it is discussed about the several concurrency control algorithms.

II. A REVIEW ON DISTRIBUTED DATABASE

At first, a distributed database model is explained in this part.

A. A distributed database model:

Fig.1 shows a common structure of a model. Every site in this model has four parts. A source which produces the transactions and maintains the levels of transaction information for site. A transaction manager who models the executive treatment of transactions. A concurrency control manager who implements the details of a special concurrency control algorithm and a resource manager who models CPU and I/O sources for site. In addition to these per-site components, the model also has a network manager who models the network communication behavior [1,9,10].

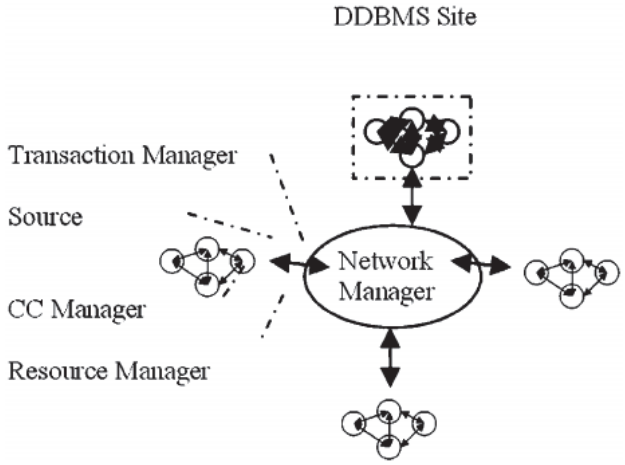

Figure.1. Distributed database model structure[1]

1) transaction manager

Every transaction in workload, will have a master slave process, some of cohort groups and updaters. The master process occupies the site which transaction has been offered in. every batch of cohorts sends reading or writing requests to one or more files which have been saved in this site. A transaction, composed of a collection or group of cohorts, exists in every site that needs to access the data [11].

Cohort groups connect to their updaters when they get permission to access required writing for repeated data and their updaters. A transaction can be executed sequential or parallel which is related to transaction class algorithm [1].

2) Resource manager

Resource manager can be assumed as a model of operating system for a site which directs the physical resources of that site consisting CPU and discs. Resource manager provides CPU and I/O services for transaction manager and concurrency control manager and also provides message delivery services that use CPU sources [1,4].

Transaction manager uses CPU and I/O resources to read and write a disc and also for sending messages. concurrency control manager uses the CPU resources too for processing the requests and sending the messages [10].

3) Network manager

The network manager encloses the network connection model. Network model is very simple. For routing messages from a site to another site, it acts only on one switch. Network properties is isolated in this model [1,7].

4) Concurrency control manager

Concurrency control manager, involves the meaning of concurrency control and it is a model which only should change from an algorithm to another algorithm and is responsible for supporting concurrency control requests made by transaction manager composed of writing and reading access requests. Requests include permission to commit a transaction [1,6,7].

In fig.2 a more precise look has been taken on a model of distributed database [1,5,6].

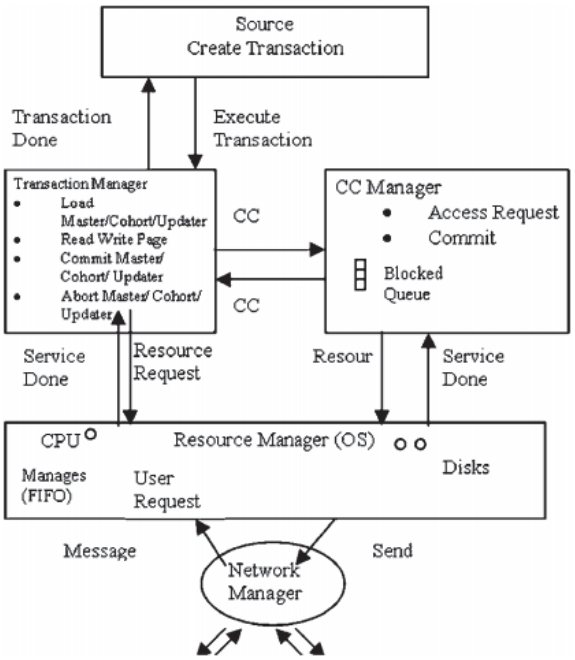

Figure.2. An example of distributed database model [1]

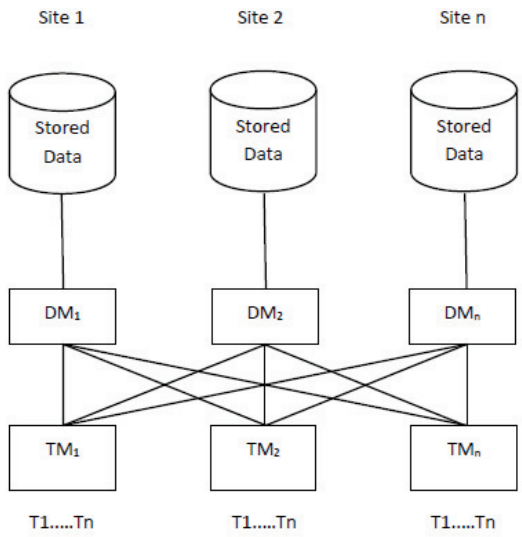

Figure.3. Distributed transaction pcocessing model[2]

B. Distributed transaction processing model:

In order to understand how concurrency control algorithm works, a simple model of distributed database management is shown in fig.3. A distributed database system is a set of sites connecting to each other through a network [2,5,6].

TM is transaction manager and DM is data manager. Here having safe network connection is important. It means if site A sends a message to site B, it should arrive destination without any error [2].

1) Transaction division

A distributed database system scenario is drawn in the following figure4.

C. Concurrency control algorithms in distributed database system

1) Two phase locking distributed protocol

The first two phase locking distributed algorithm means "Read any and write all". Transactions set reading locks on items which are read and transform these locks to writing locks on items which need to be updated. For reading an item, it is enough to set a reading lock on a copy of that item. So the local version is locked. For updating an item, it is necessary to apply writing locks on all versions. Writing locks are being established until all versions of an item are updated. All locks are confirmed until the transaction is finished successfully or gives an error message [1,5,6,7,9,10].

It is possible deadlock occurs. Whenever transactions are blocked, local deadlock is investigated and transaction sends ABORT message and is restarted again. Supporting local deadlock is done by a "snoop" processing which sends WAIT request to the information of all sites periodically and then removes local deadlock. This processing is called "SNOOP", its duty is to move between sites. No site suffer public deadlock costs [1,5,6,8,11].

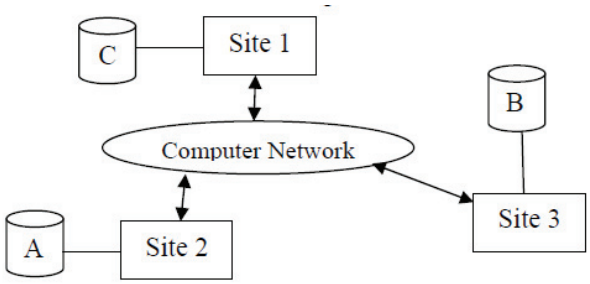

Figure.4. A senario of a distributed database[2]

2) locking algorithm wound-wait

Second locking algorithm has the law "Read every thing, write all" and is different from 2PL in supporting deadlock. Every transaction is numbered based on starting time and delaying older transactions because of newer transaction is avoided. If an older transaction requests for a lock and it is resulted in older transaction waiting for younger transaction, younger transaction is hurt. Younger transaction is restarted again. Younger transactions can wait for older transactions in order to remove the possibility of deadlock [5,6,7,12].

3) Timestamp sequence (BTO, Basic Timestamp Ordering Algorithm)

Third algorithm is the basic timestamp algorithm. Like previous algorithm, it uses timestamps of transaction start, but is used differently. Rather than a locking approach, this algorithm shares timestamps with all data items recently available and access to data through transactions is executed in the sequence of timestamps. Transaction which try to access data non-sequentially are restarted again. When a reading request is received for an item, reading timestamp is allowable to be different from writing timestamp [8,11,12]

In fact, if applicant timestamp is less than item writing timestamp, updating operation is removed simply. For repeated data, read any and write all, the used approach is to send a reading request to all possibly until a writing request is sent and being approved by all versions. collectivity of the algorithm is done by COMMIT cohort. This means that writers keep their updaters in a specific workspace up to commit time [1,4].

4) Distributed certification (OPT,Optimium protocol)

This algorithm is distributed and on the basis of timestamp. Optimum concurrency control algorithm operates on the base of certified information changing. For every data item, a reading timestamp and a writing timestamp is kept. Transaction may read data items or update items for free and save every update in a local workspace until COMMIT time. For every reading, transaction should remember identifier version like (writing timestamp) which is related to the read item. Then, when all transaction cohorts were completed and reported to MASTER, transaction is assigned with a unique timestamp. This timestamp is sent to every transaction with a ready message for COMMIT and transaction approves every reading and writing locally. A reading request is approved if the read version is still the item current version and no writing request with new timestamp has not been approved in local recently. [1,2,4].

A writing request is approved if next reads is not approved and commited and next reads has not been approved in local recently [1,5,6].

D. Algorithm basis on timestamp RCTO(Recoverable Timestamp Ordering)

This algorithm is on the basis of principle operations of timestamp in COMMIT operations. There are two vectors for every data item X: A writing vector wv which registers writing timestamp for every reading or writing operations and a COMMIT vector named cv which registers successful operations for transaction if the timestamp of the arrived COMMIT operation is not equal to the timestamp of first saved item in wv. A rotate function shifts wv elements as the first element in wv shifts to the last and other elements transfer to top according to the following picture [2,11].

This algorithm includes 3 different operations. Reading operation, writing operation and successful transaction message operation. Before defining this operation, we point that COMMIT operation timestamp on a same item for different transactions is as following codes [2].

1.if ts(Ci)=ts(wv[i])

{

2.Execute Ci,

3.Delete the contents of wv[i],

4.Move up the remaining values of wv

}

The above code operates on both vectors wv and cv. Fig.6 explains a schedule including following operations:

C3C4C5 is a group of three COMMIT transactions saving three values t1, t2 and t3 in COMMIT vector. w2(x)w4(x)w1(x) is a group of writing operations on the same object x that has been saved in writing vector for three transactions. When c4 and w4 were saved in the same row, simultaneously are removed from wv and cv. W2, c3, w1 and c5 operations moves up a row

When reading operation Ri arrives, following code is executed.

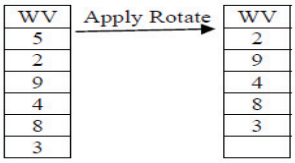

Figure.5. An example of a rotate function[2]

Index	wv	cv
1	2	C3
2	4	C4
3	1	C5

Index	wv	cv
1	2	C3
2	1	C5

Figure.6. Writing vector and commit vector after the conformity of C4 and W4[2]

1.For each received read operation i

{

2.Let z=i*random number

3.If ts(Ri) is already in wv

Then

4.Store z instead of Ri;

Else

5.Store z at the end of wv

}

Assume that the next reading operation is R2(x). This operation causes the existed table transforms to the table on the right of the following picture. Since the timestamp of read operation is already stored in wv, the contents of wv is multiplied by a random value.

The original benefit of this procedure is to realize reading from writing operation that avoids delay in executing reading operation. Fig.7 shows R2 operation arrives and then is multiplied in a random value [2].

When reading operation wi arrives, following code is executed.

1.For each received write operation i

{

2.If ts(Wi) is not stored in wv then

3.Save ts(Wi) in wv.

Else

4.Save i*random number in wv

}

Writing operation saves timestamp in vector wv. Fig.8 shows wv and cv values after doing new operations w3(x)

In fig.9 writing vector and COMMIT vector values have been shown when w3(x) arrives after R3(x)

Index	wv	cv
1	2	C3
2	1	C5

Index	wv	cv
1	2*random no	C3
2	1	C5

Figure.7. R2 operation arrives[2]

Index	wv	cv
1	2	

Index	wv	cv
1	2	
2	3	

Figure.8. Writing and commit vectors after executing w3[2]

Index	wv	cv
1	3	4
2	5	1

index	wv	cv
1	3*Random no	4
2	5	

Figure.9. Write vector and commit vector status after executing W3 when read operation accessing same data is already present.[2]

COMMIT operation is the most original one that compares ts(ci) with the first component in wv in order to keep COMMIT operation in cv. The respective code is as following:

1.Let K is the index of the first empty available position in cv.

2.For each received Commit operation Ci

{

3.If (ts(Ci)=wv[0]) or (ts(Ci)=wv[k})

{

4.Execute Ci;

5.Delete wv[0] or Delete wv[k];

6.Move up all the remaining values in wv and cv simultaneously

}

Else

{

7.record Ci at the first empty cell in cv

}

}

The advantage of this COMMIT operation is to delay every early COMMIT and force the COMMIT operation of transaction during writing to be executed before the COMMIT

operation of transaction during reading. Fig.10 shows the COMMIT operation after arriving c3. Recently w3 value has been saved in wv vector. COMMIT operation has been executed and wv values are removed [2].

III. COMPARISON AND ANALYSIS OF EXISTED METHODS

Here, four methods were used for concurrency control mechanism in distributed database.

In two phase locking algorithm, we have problems like concurrency weakness, sequential rejection and deadlock and starvation happening [1,2,7,9,12] . In algorithm wound-wait, time priority of algorithms is observed [1,2,4,11,12] . In protocole according to timestamp, conflict sequence ability is assured and deadlock does not happen. but sequential harm or starvation is possible to happen and recoverability of execution plan is not assured [1,2,4,5,6,12]. But about optimistic algorithms or approval technic should say that:

- In all previous techniques, system does some inspections before doing an operation on database.
- In approval technic which is known as optimistic technique, no inspection is done before.

Index	wv	cv
1	2	
2	3	

→

Index	wv	cv
1	2	
2		

Figure.10. c3 entrance and w3 remove from wv[2]

- this technique, in comparison with locking technique, provides more concurrency.
- In this technique, writing operation is not done in database directly, unless transaction execution has finished.
- Instead in the middle of execution, all updates are done on the local versions of related data.
- at the end of transaction execution, during an approval stage, it is inspected if any harm has been entered into serializability of execution plan or it was of no effect.
- if serializability has not been observed, transaction is rejected in order to be restarted.
- If no harm has been entered into serializability, stabilized transaction and databases are updated from local versions [4,10].

IV. SIMULATION AND TEST RESULTS

Several simulating tests were done for different assumptions effect studies about resources effect on performance of three concurrency control algorithm. The efficiency of three algorithms under unlimited resources assumption was surveyed with limited numbers. [4,10]

A. *unlimited resources*

The first test related to resources is to survey performance characters of three strategies for different levels of multi programming with the assumption of unlimited resources. With some unlimited resources, throughput rate in the absence of competition for data should be a non-falling function of multi programming level. Nevertheless for a database with special size, increasing multi programming level conflict probability will increase. for locking, conflict probability increase will be shown in increasing the number of obstructions because of rejecting lock requests and increasing the number of restarting due to deadlock. for strategies based on restarting, more conflict probability will result in more number of restarting. Fig. 11 shows Throughput results for the first test [4].

Delay time is adaptive and is equal to average reply time of executing transactions. Because of this adaptive delay, immediate restart algorithm arrives a point that all inactive transactions upper than that are in restarting delay position or in terminal think position. We get this point when the system has so many active transactions that a new transaction conflicts an active transaction and as a result is restarted immediately and delays. Therefore restarting average delay time increases that result in decreasing the transactions competing for active manner and as a result, conflict probability decreases. When arriving a constant point, there is no waiting transaction in ready array and multi programming level increase has no effect in this point. Fig.12 shows algorithms reply time [4,9,10].

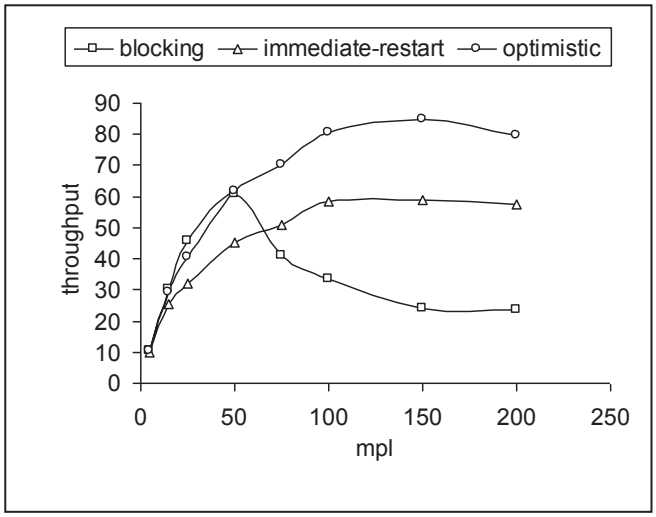

Figure.11. Throughput with assumption of unlimited resources.

B. *Limited resources*

In second test, limited resources effect on performance properties of 3 concurrency control algorithm is investigated. A database with a processor and two discs have been spotted for this test. Fig.13 shows throughput for 3 algorithms in this status. Paying attention to throughput curve for every three algorithms indicates thrashing happening when multi programming level increases. Throughput increases at first, then climaxes and at the end decreases or fixes [4,10].

When multi programming level was increased, at first throughput increased for every 3 algorithms because there were not enough transaction number in multi programming low level for imparting the resources. Fig.14 shows the response time for 3 strategies [4].

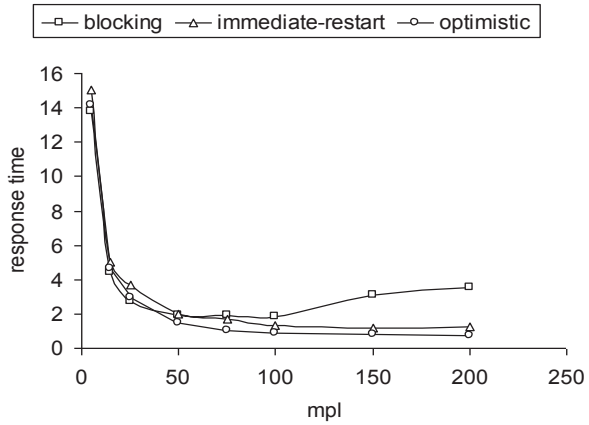

Figure. 12. Reply time with the assumption of unlimited resources.

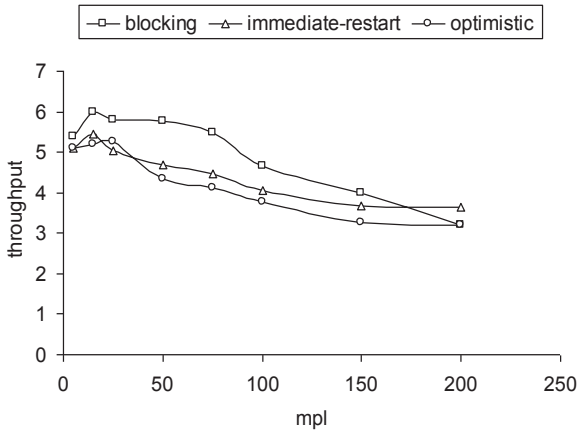

Figure.13. Throughput (a source unit)

CONCLUSION

In this paper we pointed out four concurrency control algorithms in distributed database and introduced an algorithm sample based on timestamping. At the end, after comparing these four algorithms in the last part we conclude that optimistic algorithm and based on certification in low work load has lower abort rate than other algorithms and in low work load, it has the least amount of increase in response time because of repeating and has more concurrency than locking technique , and also no inspection is done on database before.

In general, optimistic algorithm is chosen as the best concurrency control mechanism in distributed database.

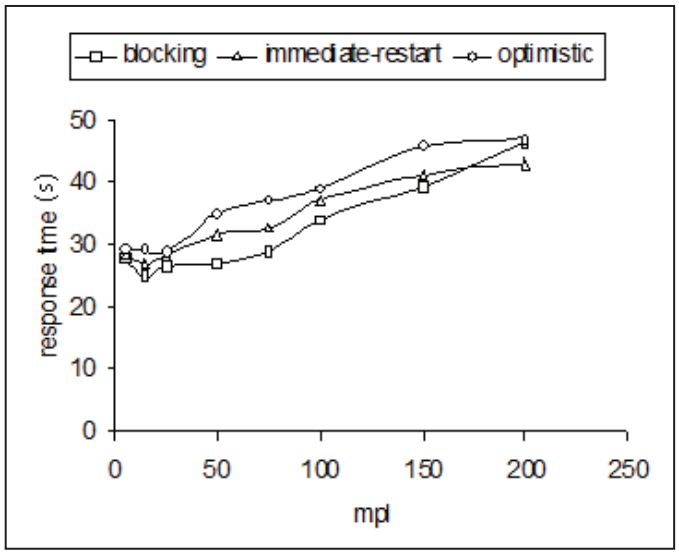

Figure.14. Response time (a source unit)

V. REFRENCES

[1] Yadav, Arun Kumar, and Ajay Agarwal. "An approach for concurrency control in distributed database system." *International Journal of Computer Science & Communication* 1.1 (2010): 137-141.

[2] Chauhan, Rinki, et al. "Recoverable Timestamping Approach For Concurrency Control In Distributed Database." (2011).

[3] Coronel, Carlos, and Steven Morris. *Database systems: design, implementation, & management*. Cengage Learning, 2016.

[4] Agrawal, Rakesh, Michael J. Carey, and Miron Livny. "Concurrency control performance modeling: alternatives and implications." *ACM Transactions on Database Systems (TODS)* 12.4 (1987): 609-654.

[5] Abbas, Qasim, et al. "Concurrency control in distributed database system." *2016 International Conference on Computer Communication and Informatics (ICCCI)*. IEEE, 2016.

[6] Gupta, Akshay M., and Yogesh R. Gore. "Concurrency Control and Security Issue in Distributed Database System." (2016).

[7] Kaur, Mandeep, and Harpreet Kaur. "Concurrency control in distributed database system." *International Journal of Advanced Research in Computer Science and Software Engineering ISSN* 2277 (2013).

[8] Kudo, Tsukasa, et al. "An Implementation of Concurrency Control between Batch Update and Online Entries." *Procedia Computer Science* 35 (2014): 1625-1634.

[9] Burger, Albert, Vijay Kumar, and Mary Lou Hines. "Performance of multiversion and distributed two-phase locking concurrency control mechanisms in distributed database." *Information Sciences* 96.1 (1997): 129-152.

[10] Kumar, Vijay. "Performance comparison of database concurrency control mechanisms based on two-phase locking, timestamping and mixed approach." *Information Sciences* 51.3 (1990): 221-261.

[11] Ramesh, Dharavath, et al. "Hash Based Incremental Optimistic Concurrency Control Algorithm in Distributed Databases." *Intelligent Distributed Computing*. Springer International Publishing, 2015. 139-150.

[12] Nguyen, Loc. "An Advanced Approach of Local Counter Synchronization to Timestamp Ordering Algorithm in Distributed Concurrency Control." *Open Access Library Journal* 2.10 (2015).