Evaluation of Multiversion Concurrency Control Algorithms

[1]Sonal Kanungo, [2]Rustom. D. Morena
[1]Smt. Z. S. Patel College of Computer Application, Veer Narmad South Gujarat University, Surat
[2]Veer Narmad South Gujarat University,Surat

Abstract- This paper presents a simulation study of the execution of different multiversion concurrency control algorithms in standard environment. Several Multiversion concurrency control schemes has been proposed till date. In this paper, we are discussing comparison of three Multiversion schemes; Multiversion Timestamp Ordering, Multiversion 2PL Mixed Method and Multiversion 2PL with Certify Lock (2V2PL) on the premise of their performance, locking mechanism and deadlocks through simulations. Analysis of algorithms is important because, it discovers the characteristics for evaluation and compare it with each other for the same application. The analysis of an algorithm can help us understand it better and can also propose new improvements.

Keyword- *Multiversion; Concurrency control; Serialization; Rollback; Deadlock; Abort; Update; Readonly*

I. INTRODUCTION

The Multiversion concurrency control provides flexible methods which permit Read-only transaction to Read slightly old, but still consistent version of the data. Read operations that would be rejected in other techniques can still be accepted by Reading an *older version* of the item. Multiversion protocol never overwrites old values and these old values or versions are always available to tardy Reads [5]. Multiple versions item helps the scheduler to avoid rejecting operations that arrive too late. A Read normally rejects because the value it was supposed to read has already been overwritten [4]. This rejection of Read can be avoided by keeping old versions; a late Read can be given an old value of a data item, even though it was "overwritten" [3].

The Multiversion concurrency control algorithm produces a new copy or version of data item with each Write on a data item. A list of versions of data item with the history of values is kept. The existence of multiple versions is only visible to the scheduler, not to user transactions [3].

A. Multiversion Technique Based on Timestamp Ordering

Reed's Multiversion timestamp ordering scheme solves problem of deadlocks with 2PL, by ordering transactions and aborting transactions that access data out of order. This increases the concurrency of the system by never making an operation block [2].

The Timestamp Ordering maintain several versions of each data item; each version keeps the value of version and the following two timestamps; Read Timestamp and Write Timestamp. Read timestamp is the largest of all the timestamps of transactions that have successfully read the version and Write timestamp is the timestamp of the transaction that wrote the value of version. Read operation reads the version with the largest timestamp. The timestamp of the reading transaction is added to the item. Write operation creates a new version of data items. The content field of this version holds the value written by transaction.

The transaction can commit only when, its timestamp is greater than last Read and Write timestamp of data else this transaction will abort and rollback. Rollback and abort also happen when the transaction is attempted to write a version data that should have been Read by another transaction. It will abort and restart in basic timestamp ordering and this rollback will cause cascading rollbacks [9]. Old values are never overwritten therefore Reads are always available. Writes do not overwrite each other so Reads can read any version which gives flexibility to Multiversion concurrency control [3].

B. Multiversion Locking

Multiversion locking protocol combines the advantages of Multiversion concurrency control with two-phase locking [3]. When a transaction is written on a data item, it always creates a new version of data items. This Write will not overwrite the old value of the item but creates a new version and keeps both versions. This transaction sets an exclusive lock on a data item that prevents other transactions from reading or writing on the same data item but allows other transactions to Read the previous committed version of this data item [14]. This gives flexibility to other transactions to Read with the option of supplying it to either version; whichever will serve best serializability [7]. The second version is created when a transaction acquires a Write lock on the item. Keeping two versions of each item; one version must always have been written by some committed transaction. Since Writes do not overwrite each other, Reads can read any version, which provide more flexibility in controlling the order of Reads and Writes. This gives a "late" Read operation the chance to Read a value which would have been erased in a single-version system [14].

1)Multiversion Two Phase Locking with Certify Lock (MV2PL)

This Multiversion supports two versions. When a transaction issue a Write; this Write does not overwrite the old value of the item but create a new one and keep both versions. If subsequently another transaction wants to read data, we have the option of supplying to it either version [7].

Each Write on a data item produces a new copy of the data and Read on data item selects one of the versions of data item. The transaction, which is written on a data item, but it is not committed yet, then the two versions of data are present; one is 'before image' and the other is 'after image'. As soon as the transaction commits, the before image can be dropped. Since the new version of the data is now stable and old versions are no longer needed and there is no need to maintain it [1].

MV2PL supports multiple-mode locking scheme which needs three locking modes for an item; Read, Write, and Certify [14]. MV2PL allows other transactions to Read the old committed version of the data with Write lock. The transaction can write the value of data, without affecting the value of the committed version data.

When this Write will be ready to commit, it must obtain a Certify lock which is not compatible with Read locks, that means the next read operation is not granted to this data. The transaction has to delay its commit until all Reading transactions released the write lock item then obtain the Certify locks [8]. Once the Certify locks which are exclusive locks are acquired, the committed version of the data item is set to the value of "new" version data and "old" version is discarded then only Certify locks will release. Therefore, certify lock is not compatible with Read locks [15]. Certify lock is used to delay the commit of a transaction, if there is still any active reader for data items which are about to be overwritten.

Certify locks only conflict with Read locks and other Certify locks [15]. Concurrency can be improved by Certify locks. When a transaction Writes an entity and then successfully terminates. The "after" value of the entity replaced the "before" value as the "official" value in the database. Therefore "before" value must require terminating successfully and "after" values are still retained in the database. After a transaction terminates, the concurrency control need no longer keep track of information about the transaction [1].

2)Multiversion Mixed Method

In traditional 2PL, Read-Write conflicts block each other. Write lock on a data item always prevents transactions from obtaining Read locks on the same data item. In the standard locking scheme, the Read lock is shared and Write lock is an exclusive lock. Once a transaction obtains a Write lock on an item, no other transactions can access that item. This algorithm uses Multiversion Timestamping to process Read-only transactions (queries) and Multiversion locking to process general transactions (updates). Querying (Reading) data don't conflict with locks acquired for writing data and so Query never blocks Updaters and Updaters never blocks Query [6].

Multiversion 2PL protocol differentiates between Read-only transactions and Update transactions. The Update transactions follow a rigorous two-phase locking where all locks are released only in the end of the transaction at commit time [3]. A single timestamp is kept for each version of a data item. When an Update transaction, Reads or Writes a data item it locks the item just as it would in two-phase locking and it Reads or Writes the most recent version of the item. An update transaction wants to Reads an item it gets shared lock on the item and Reads the latest version of that item and when an Update transaction wants to Write an item, it first gets an exclusive lock on the item and then creates a new version of the data item. With each Writes on a data item, a new version is created. It sets a lock on data and prevents other transactions from Reading or writing a new version. At the same time, other transactions are allowed to read the previous version of the data. Thus, Reads are never delayed [10]. Since the timestamp associated with a version is the commit timestamp of its Writer, a Read-only transaction is thus made to only Read versions that were written by transactions that committed before Read even began running [7].

The "before" or old value, and the "after" or new value which creates after commit of Write operation. Concurrency can be increased by allowing other transactions to Read the before values of a given transaction. Some systems have a permanent copy of the before value for recovery purposes [1]. When Read-only transactions start execution, they assign a timestamp by reading the current value. Multiversion timestamp-ordering protocol is used for Reads [2]. Read-only transactions never wait for locks. Multiversion two-phase locking also ensures that schedules are recoverable and cascade less [14]. Queries never delay or abort updaters, and updaters never abort queries.

II. RELATED WORK

CHRISTOS H. PAPADIMITRIOU, PARIS C. KANELLAKIS, NATHAN GOODMAN DESCRIBED VARIOUS CONCURRENCY CONTROL METHODS USING MULTIPLE VERSIONS [2,3,5] Improved Multiversion concurrency control is given with effective necessary and sufficient conditions for an execution to be l-SR concurrency control and extended concurrency control theory. This condition uses the concept of version order. They gave a graph structure, Multiversion serialization graphs (MVSGs), that helps to check these conditions and applied the theory to three Multiversion concurrency control algorithms. One algorithm uses time stamps, one uses locking, and one combines locking with timestamps.

PHILIP A. BERNSTEIN and NATHAN GOODMAN [3] This paper had presented a theory for analyzing the correctness of concurrency control algorithms for

Multiversion database. They presented some new Multiversion algorithms analyzed these new algorithms and several previously published ones. This paper gives reviews concurrency control theory for nonmultiversion databases and extends the theory to Multiversion databases.

RICHARD E. STEARNS AND DANIEL J. ROSENKRANTZ [1] This paper deals with the Write of the database. Write of a database entity is both the "before" or old value, and the "after" or new value. Two schemes for producing such controls are given, first scheme works in a system where processes are committed on termination, and the other for systems where commitment happened later. They introduce some basic terminology and model of a concurrent, distributed database system. This paper developed a series of design principles which applied to all designs. Two concurrency control schemas are introduced, one is for the case COMMIT=CLOSE and the other for COMMIT=TERMINATE.

D. AGRAWAL, V. KRISHNASWAMY [6] This paper states that in database systems consisting of abstract data objects where blind-Write operations are dominant, for non-interfering execution of Write-only transaction is useful. Transaction ‘s operations do not observe the states of the objects; instead they “blindly” modify the object states. The transactions should be classified into three categories: Read-only, Read-Write, and Write-only transactions. Multiversion data can also be used to eliminate or minimize the interference between the Read-Write and Write-only transactions. A separate treatment of Write-only transactions is given. In this paper, a version control mechanism is given that minimizes the interference between the Read-Write and Write-only transactions.

MICHAEL J. CAREY and WALEED A. MUHANNA [9] This paper examined the performance and storage overheads of three Multiversion concurrency control algorithms, Reed’s Multiversion timestamp ordering algorithm, the CCA Multiversion locking algorithm, and a Multiversion variant of Kung and Robinson’s serial validation algorithm. Authors compared the performance of the algorithms to their single-version counterparts like timestamp ordering, two-phase locking, and serial validation, respectively. The study of these algorithms was based on a detailed simulation model of a centralized (i.e., single-site) database management system.

III. METHODOLOGY FOR EXPERIMENTAL ANALYSIS

Concurrency control algorithms are described for simulation worked with the same set of data and transaction’s operation is applied to each algorithm. There is transaction generator which generate Readonly and Update transactions. Update transaction has read-set and a write-set. These determine the data item that the transaction will read and write during its execution.

A. Multiversion Timestamp Ordering

We used System clock for two timestamps; Read Timestamp and Write Timestamp for each data. When the transaction is entered in system timestamp is given to the transaction. When transaction wants to Read some data, this Read timestamp will select time stamp equal to Write’s timestamp of latest committed version of data else it will Read the old version. This Read transaction’s timestamp will have assigned as Read timestamp. Read never fails in this methodology.
When transaction wants to Write on some data, transactions timestamp should be greater from last Read and Write, stamp of given versions’ timestamp and new version is created, it will not overwrite the old version. This Write will succeed and Write transaction’s timestamp will become new Write timestamp and Read stamp of this new version.
In our experiments, we observed that nearly one fourth of transactions are committed while others are roll backed. We did not find any wait here. Large number of Restarts are generated because of Rollbacks took place in a large number. If Write is performed on some data and other requests to update that data, not only the operation wills Rollback whole transaction will Rollback. This effect is known as cascading Rollback. This protocol is free from conflict serializability, but a lot of cascading rollbacks are generated.

B. Multiversion Two-Phase Locking Using Certify Locks Version 2PL

This protocol works same as traditional two phase Multiversion protocol the only difference is a delay in unlocking. When a Transaction’s operation enters, it first checks whether it has a lock or no-lock on the data. In case no-lock is found in the data, a lock is applied to the data. This lock can be shared (Read S) or exclusive (Write X). If all the operations got locks new version is created (not committed) and the old version is still available for Read. When the transaction is ready to commit; X lock Converts into C lock, this is incompatible to Read locks. All Read-only transactions are committed first, then only Update transactions can commit. After that commit takes place, unlocking is performed and the old version is converted into a new version. When Write operations got an X lock on the data, this goes to wait. We ran these transactions, we found that only very less transactions are committed while others stuck largely in deadlock and Rollbacked.
As the X lock is converted in to Certify lock and Certify lock is not compatible with Read lock, Read operations will send to wait, this will create more deadlocks. We found that few transactions were committed and number of waits was generated. This protocol is free from conflict serializability.

C. Multiversion Mixed Method

Read-only and Update transactions are predefined. Update transaction operations are a combination of Read and Write**.** An update transaction follows rigorous 2PL. When an Update Transaction's operation enters, it first checks whether it has a lock or no-lock on the data. In case no-lock is found in the data, a lock can apply to the data. This lock can be shared (Read S) or exclusive (Write X). When a transaction is committed, unlocking is performed and new version is created and transaction counter will increase by one. Else, the operation goes to wait which means any of the two or both operations have found lock on that data.

Read-only transactions that start after *Update*, it will Read the values updated by a transaction. Read-only transactions that start before *Update*, it will Read the value before the update starts.

We ran these transactions, compared to certify lock more transactions are committed while others are either in abort and stuck into deadlock. We found that few transactions were committed and number of waits was generated. This protocol is free from conflict serializability. However, we found overheads of the lock.

Read-only will Read old value as the new value of Update's write is not committed but Read of Update will have to wait as Writing is done by Update transaction. Read-only doesn't mind Reading old value.

IV. DISCUSSION

In this section we present detail study of experiment, which describe the behavior of all the concurrency control algorithms. On the basis of above experiments we are giving a comparison of Multiversion concurrency control for all 3 methods.

A. Performance

1) Multiversion Timestamp Ordering (MVTO) scheduled in first-in-first-out (FIFO) fashion. Transactions are conflict-free therefore they can Read the same item at different times. Transactions do not block each other which enhance concurrency. Multiversion Timestamp ordering minimizes the number of restarts as it allows Reading old version of data that is written by another transaction and not committed yet. The case where Reading is a more frequent operation than writing, this protocol gives better result. Blocked transaction rollback rather than waits for access. While there is some drawback with Multiversion Timestamp Ordering Protocol, two potential disk accesses require for Reading of a data item and updating the timestamp field. A sequence of conflicting of short transactions causes repeated restarting of the long transaction therefore there is possibility of starvation of long transactions, cascading rollbacks are also unavoidable.

The starvation problem occurs where a transaction might get started and aborted many times before it finishes. When a transaction restarts it will receives, a new startup time stamp each time, so the timestamp ordering of such a pair reverses every time one of them restarts. A pair of update transactions wishing to concurrently Read and then Write a common object can restart each other over and over again, and this "restart loop" can persist indefinitely.

2) 2V2PL Transaction can read an item while another transaction holds a Write lock on it. Keeping two versions of each item; one version must always have been written by some committed transaction. The second version is created when a transaction acquires a Write lock on the item. Since Writes do not overwrite each other, Reads can Read any version. Write locks will convert into Certify locks before Commit. Certify locks conflict with Read locks. On those data items where such Read locks exist, the lock conversion is delayed until all Read locks will release. Thus, Transaction could not commit until there are no active Readers of data items it is about to overwrite. It requires more storage to maintain multiple versions of data item.

These lock conversions can lead to deadlock just as in standard 2PL. When a transaction has a Read lock on data and other transaction has a write lock on it. If the first transaction tries to convert its Read lock to a Write lock and other transaction tries to convert its Write lock to a Certify lock, then the transactions are deadlocked. Read will also wait in one condition, when there is a Certify-lock on data, and transaction is greater than the time of that Certify-lock, then this Read must wait until certify will release.

3) MV2PLMixed Method This protocol distinguished Read and Updates and their effect on the database. An update transaction locks the item Reads or Writes just as it does in two-phase locking, and it Reads or Writes the most recent version of the item. Transactions block when they cannot obtain a lock, and deadlock must be dealt with in one of the usual ways. A new version is created when an item is written. The transaction will commit and create version, this y version of an item is stamped with the commit timestamp of its creator. A Read-only transaction wishes to access an item, without locking it. The transaction, simply Reads the most recent version of the item Since the timestamp related with a version is the commit timestamp of its Writer, a Read-only transaction only Reads versions that were composed by transactions that committed before the transaction even started running. Read-only transactions are always successful. This is good because it is safe for update-intensive applications.

B. Methodology

1) Multiversion Timestamp Ordering MVTSO does not perform locking on operations and no transaction has

to wait for other, a Multiversion Timestamp protocol ensures freedom from deadlocks. Read request never fails and it is never made to wait. Conflicts between transactions are resolved through rollbacks, rather than through waits, this will result into cycle restarts, which becomes more expensive. It avoids cascading aborts, since transactions are only allowed to Read the version that was written by a committed transaction.

2) 2V2PL Two phase locking with certify lock exclusive lock 'X' become certify 'C' when transaction is Ready to commit. The rigorous locking is performed that means both sharable and exclusive locks will be unlocked after the commit of the transaction. The transaction becomes committed only when all certify locks are released. Certify locks, delayed unlocking and unlocking is done until no Read operation will be left. Certify locks in 2V2PL behave much like Write locks in ordinary 2PL. 2V2PL's certifies locks, delay Reads for less time than 2PL's Write locks delay Reads. Read locks, delay a transaction's certification in 2V2PL, which improved concurrency of Reads but increase the expense of delaying the certification. Read-Write-Certify must still block Read requests when certifying an update. Deadlocks can occur because transactions go in the wait state. An updater can still delay a query under one condition when a query Reads data and updater has a certify-lock on it, and Read transaction is greater than the time of that certify-lock, then Read must wait until the transaction certifies data.

3) Multiversion Mixed Method The transactions that issue Reads but no Writes are called queries, while those that issue Writes (and possibly Reads as well) are called Updaters. Queries uses MVTO and Strict 2PL is used by updaters. Here all locks (shared and exclusive) are held till commit/abort. Therefore, there are fewer chances of Rollback and cascading rollbacks. This will give best results to Query (Read) transactions which doesn't mind Reading old values. While locking overhead, it inhibits concurrent execution. It is inefficient for query-intensive applications because of locking overhead, possibility of deadlock and waits for locked data. In this method queries, may Read out-of-date data. Tagging and interpretation of timestamps on versions may add significant scheduling overhead.

C. *Rollback and Deadlock handling*

*1)*Multiversion Timestamp Ordering Conflicts between transactions are resolved through rollbacks, rather than through waits. This may be an expensive method. Multiversion timestamp-ordering scheme does not ensure recoverability and cascadelessness. Timestamp ordering is free from deadlocks. The timestamp protocol ensures freedom from deadlock as no transaction ever waits.

*2)*2V2PL 2PL with Certify lock a transaction may lead to deadlock while trying to convert its Write locks and may be aborted during this activity. These locks could not release till transaction become commit. 2PL and Certify lock (2V2PL) have less cascading rollback as compared to Multiversion Timestamp Ordering.

Lock conversions can lead to deadlock just as in standard 2PL. A transaction has a Read lock on data and other transaction has a write lock on this data, if the first transaction tries to convert its Read lock to a write lock and another tries to convert its write lock to a certify lock, then the transactions are deadlocked.

*3)*Multiversion Mixed Method Queries and updaters never delay each other. A query can always read the data it wants without delay. Although updates may delay each other, queries set no locks and therefore never delay updates. This is in sharp contrast to 2V2PL, where a query may set many locks. Deadlocks are found, but they are less as compare to 2V2PL.

V. RESULT AND ANALYSIS

The results of our experiments designed to examine the performance of the multiversion concurrency control algorithms. The algorithms are a mix of small update transactions and larger read-only transactions. The relative performance of the three multiversion algorithms behaves over a range of update conflict probabilities with on number of commit, abort and rollback.

	No of Transactions	**No of Commit**	**No of Abort**	**No of Rollback**	**No of Wait**	**No of Deadlock**
MV2PL	100	28	72	32	56	44
MVTSO	100	37	45	88	0	0
MVMM	100	45	40	22	33	31

Table-1 Comparison of MV2PL MTSO MVMM

Above three methods performances have been compared through analytic and simulation studies.

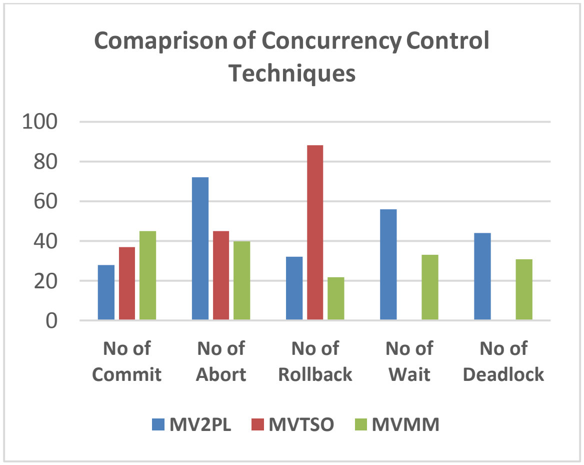

Fig.1: Comparison of Concurrency Control Techniques

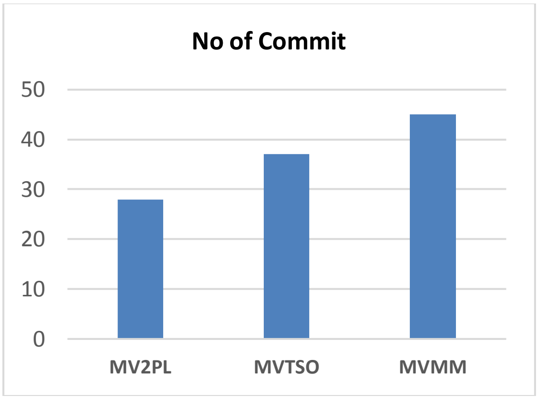

Fig.2: Number of Commit

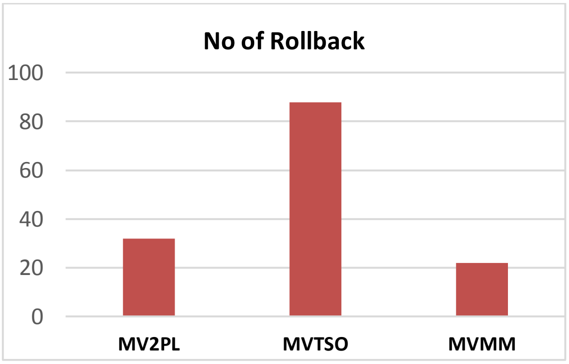

Fig.3: Number of Rollback

These methods are usually based on exclusive/shared access to abstract database objects. The more complex lock request patterns are generated by relational queries, which need to be incorporated in performance studies.

It is assumed that the database size remains fixed as the transaction arrival rate is varied, which may not be necessarily true, for dynamic environment these results may vary.

VI. CONCLUSION

Multiversion concurrency control algorithms can give improvements in performance by allowing large Read-only transactions to access previous or older versions of data items. Multiversion schedulers are enhancing the performance of the concurrency control component of a database. In this paper, our findings based on different Multiversion protocols say that Time stamping Multiversion protocol gives best result with Read only transactions while Mixed Method and 2V2PL are good for update intensive transactions. 2V2PL delays commit as compare to 2PL and give more accurate results as it allows reading of all read operations on certain data, then only Write Lock will release, So Read operations can Read old values and then Write will commit and data is assigned to the new value of a particular data item. Multiversion Mixed method differentiates Readonly and Update and give different treatment and gives best results. The user may update each version of the design independently. Therefore, query and Update never block each other. An obvious drawback of Multiversion techniques is that more storage is required to maintain multiple versions of the items.

VII. REFERENCES

[1]. RICHARD E. STEARNS AND DANIEL J. ROSENKRANTZ, "Distributed Database Concurrency Controls Using Before-Values", Computer Science Department, State University of New York, C l981 ACM 0-89791-040-O/8 0/0400/0074.

[2]. CHRISTOS H. PAPADIMITRIOU, PARIS C. KANELLAKIS, "ON Concurrency Control by Multiple Versions", 1982 ACM Publication.

[3]. PHILIP A. BERNSTEIN and NATHAN GOODMAN, "Multiversion Concurrency Control-Theory and Algorithms", ACM Transactions on Database Systems, Vol. 8, No. 4, December 1983, Pages 465-483.

[4]. SHOJIRO MURO, TIKO KAMEDA, TOSHIMI MINOURA, "Multi-version Concurrency Control Scheme for a Database System*", JOURNAL OF COMPUTER AND SYSTEM SCIENCES 29, (1984) Pages 207-224.

[5]. CHRISTOS H. PAPADIMITRIOU, "On Concurrency Control by Multiple Versions", ACM Transactions on Database Systems, Vol. 9, No. 1, March 1984, Pages 89-99.

[6]. D. AGRAWAL AND V. KRISHNASWAMY, "USING Multiversion data for non-interfering execution of Write-only transactions", Proceedings of the 1991 ACM SIGMOD international conference on Management of data ,1991, Pages 98 – 107.

[7]. HENRY F. KORTH, ABRAHAM SILBERCHATZ, S. SUDARSHAN, "Concurrency Control: Database system Concepts (Forth Edition)", Page :591 -617.

[8]. RAMEZ ELMASRI AND SHAMKANT B. NAVATHE, "Multiversion Concurrency Control Techniques:", Pages 585-587.

[9]. MICHAEL J. CAREY and WALEED A. MUHANNA, "The Performance of Multiversion Concurrency Control Algorithms", ACM Transactions on Computer Systems, Vol. 4, No. 4, November 1986, Pages 338-378.

[10]. AZER BESTAVROS, “Multi-version Speculative Concurrency Control with Delayed Commit” Computer Science Department Boston University, Boston, October 27, 1993.

[11]. BAOJING LU, QINGHUA ZOU, WILLIAM PERRIZO, “A Dual Copy Method for Transaction Separation with Multiversion Control for Read-only Transactions”, SA C 2001, Las Vegas, NV, © 2001 ACM 1-58113-287-5/01/02.

[12]. GERHARDWEIKUM, GOTTFRIEDVOSSEN, “TRANSACTIONAL Information Systems Theory, Algorithms, and the Practice of Concurrency Control and Recovery”, A volume in The Morgan Kaufmann Series in Data Management Systems,2002, Pages 185–216.

[13]. CHANJUNG PARK, SEOG PARK, SANG H. SON, “Multiversion Locking Protocol with Freezing for Secure Real-Time Database Systems”, IEEE Transactions on knowledge and data engineering, vol. 14, no. 5, September/October 2002.

[14]. K M PRAKASH LINGAM, “Analysis of Real-Time Multi version Concurrency Control Algorithms using Serializability Graphs”,2010 International Journal of Computer Applications (0975 - 8887) Volume 1 – No. 21.

[15]. TIMOTHY MERRIFIELD, JAKOBER IKSSON, “CONVERSION: Multi-Version Concurrency Control for Main Memory Segments”, Eurosys’13 April 15-17, 2013, Prague, Czech Republic Copyright 2013 ACM 978-1-4503-1994-2/13/04.

[16]. Anand S.Jalal, S. Tanwani, A. K. Ramani: “Lecture Notes in Computer Science:
Optimistic Concurrency Control in Firm Real-Time Databases, Springer-Verlag Berlin Heidelberg 2005, Pages 487 – 492.

[17]. SONAL KANUNGO, R.D. MORENA, “Analysis and Comparison of Concurrency Control Techniques”, International Journal of Advanced Research in Computer and Communication Engineering, Vol. 4, Issue 3, March 2015.

[18]. SONAL KANUNGO, R.D. MORENA, “Comparison of Concurrency Control and Deadlock Handing in Different OODBMS”, International Journal of Engineering Research & Technology, Vol. 5 Issue 05, May-2016.

[19]. Marwa Mohamed, Mohammed B. Badawy, Ayman El-Sayed: “Survey on Concurrency Control Techniques”, CAE, Volume 5,May 2016.