Effective Correctness Criteria for Serializability in Multiversion Concurrency Control Technique

Sonal Kanungo, R.D. Morena

Abstract: ***The Two-Phase Locking of Multi-Version Concurrency Control (MV2PL) avoids conflicts of concurrent transactions by make them wait until conflicts get resolved. These conflicts of transactions may generate starvations and deadlocks in the system. The Transactions suffering from deadlocks are usually aborted and restarted from scratch. In multiuser systems, this "abort and restart" approach degrades the system's performance where higher conflicts for data or long running transactions occur. Restarting from scratch also generates a negative response loop in the system, because the system suffers additional overheads which may result into even more conflicts. In this paper, we are proposing a novel approach for conflict resolution in Multi-Version Concurrency Control for shared databases. Our mechanism quickly detects the conflicts using correctness criteria for serializability and resolves the conflicts between transactions and increases throughput.***

Keyword ***Abort, Consistency, Concurrency, Commit, Multiversion, Locking, Rollback, Serializability***

I. INTRODUCTION

The Multiversion 2PL protocol keeps multiple versions of each data time in the system to support transaction and concurrency control. A single timestamp has been kept for each version of a data item [2]. When one transaction is retrieving a data item, no other transaction can update that data item [16]. The MV2PL is consists of two types of transactions Read-only and Update. Multiversion time stamping has been used to handle Read-only transactions or queries while Multiversion locking is used to control general transactions or updaters [12]. Update transactions will apply read and write locks in the same way as it used in rigorous two-phase locking protocol [13]. To read the data item update transaction will apply the shared lock on the item and reads the latest version of that item and to write on data item, it will apply an exclusive lock on the item and create a new version of that data item [8]. Exclusive lock on the data prevents other transactions from reading or writing a new version. All locks will release only at the end of the transaction's commit [6]. Every successful write on the data item creates the new version of data [17]. During this time Read-only transactions can read the previous committed version of data. Read-only transactions are given a timestamp before they start execution so they can read the latest committed version. Read-only transactions never wait for locks [14]. Since the timestamp associated with a version is the commit timestamp of its writer, therefore a Read-only transaction will read the versions that written by transactions committed before read began running [7].

In this type of conflict resolution techniques where all transactions must check the order of transaction (serialization) as well as conflicts of transactions makes mechanism slower [11]. We are modifying Multiversion Two Phase locking by adding correctness criteria to the concurrency control algorithm [20]. The serialization can improve by providing new correctness criteria. In this paper we are presenting two new criteria for the correctness of serializability; namely ***R\|W 'Read-First operation dependent'*** or ***Delay locking*** or ***D Serializability*** **and** ***W\|R 'Write-First operation dependent'*** or ***Relaxed Locking*** or ***R Serializability***. This serializability resolves the conflicts among concurrent transactions by providing them correct schedule.

II. REVIEW OF LITERATURE

K.P. ESWARAN, J.N. GRAY, R.A. LORIE, AND I.L. TRAIGER has presented two phase locking protocols [1] which hold write locks until the transaction commits or aborts to ensure serializability. If a process fails to acquire all the locks during the first phase, then it is necessary to release all of them, wait, and then start again.

These locking protocols are good for update intensive applications but there is locking overheads as well as they are not free from deadlocks. Another disadvantage is that, there is unnecessary locking for read transactions [18].

DAVID P. REED has presented time stamp protocols [2] where the transactions give better concurrency over two phase locking because transactions do not block each other needlessly. In this algorithm, the timestamp has been given to a transaction when it begins. The timestamp should be unique with respect to the timestamps of other transactions.

Time stamp protocols suffer from many rollbacks when conflicts occur. When transaction aborts, it will restart with a new timestamp. Sometimes this restart will result into a cyclic restart where a transaction can repeatedly restart and abort without ever finishing [9]. These cascading rollbacks degrades concurrency. For maintaining timestamps, two timestamps must be kept for every data object. This has been considered as another disadvantage, because it has storage overhead [19].

H.T. KUNG, JOHN T. ROBINSON has given the optimistic protocol [3] where commit takes place only after validation phase. In frequent-update systems the optimistic methods check timestamps later. In such cases where the conflicts occur between transactions, it may abort more transactions than other concurrency control methods [24].

Some algorithm resolves the conflicts among concurrent transactions by only partially aborting and restarting them [25].

KRITHI RAMAMRITHAM, PANOS K. CHRYSANTHIS gives taxonomy of serializability which implements concurrency control techniques or to impose some structure or ordering on the

set of data items in the database, we need to have additional information about the transactions. If such kind of information is not available, then two-phase locking is necessary for conflict serializability [10].

CHRISTOS H. PAPADIMITRIOU, PARIS C. KANELLAKIS [5] has presented Multiversion concurrency control protocols. These protocols maintain several versions of a data item and assigns the right version to a read operation of the transaction.

Thus, unlike other techniques, a read operation in this mechanism is never applied lock and therefore never rejected. The value read was supposed to read is already overwritten therefore this read is normally rejected. Here reading old copies of each data item can avoid rejections [6].

RICHARD E. STEARNS AND DANIEL J. ROSENKRANTZ presented theory of "before" or old value, and the "after" or new value which creates after commit of Write operation. The concurrency can increase by allowing other transactions to Read the before values of a given transaction. Some systems have permanent copy of the before value for recovery purposes [4].

CHRISTOS H. PAPADIMITRIOU had given many Multiversion algorithms where concurrency control techniques follow serializability and it is also possible to read the versions with their updated values therefore multi-version is best among all techniques [8].

PHILIP A. BERNSTEIN, NATHAN GOODMAN had extended concurrency control theory to account for the translation aspect of Multiversion databases by using the idea of one-copy serializability [6]. Multiversion schedulers are enhancing the performance of the concurrency control of a database.

The user may update each version of the design independently [17]. Therefore, query and Update never block each other. An obvious drawback of Multiversion techniques is that more storage required to maintain multiple versions of the items [21].

III. MODIFIED MULTIVERSION TWO PHASE LOCKING

The main idea behind Modified Multiversion Two Phase Locking (MMV2PL) is to improve throughput of system. In MV2PL the operations read the most recent (latest) committed versions of the data to ensure serializability of the execution. This protocol checks conflicts in same way as other Multiversion protocols do. We are also adding correctness criteria for some cases where these protocols fail. Our correctness criteria for transaction scheduling helps to identify the immediate the dependencies among different operations. Using this information, we find the waits and deadlocks generated by operations of transaction affected by the conflicts and then quickly solve this by using correctness criteria. The dependent operations information solved by employing correctness criteria that is R serializability and D serializability.

These conflicts happen when a transaction reads or write some data objects from the database that is having read lock or write lock by same transaction. When the transaction enters in system, they scheduled in serial and then by checking whether it is requesting same data item more than one time by same transaction locks have decided. The serializability of these locks has specified by correctness criteria which makes it possible to efficiently solve conflicting transactions. MMV2PL provides an efficient conflict resolution mechanism for multi-version databases, which resolve conflicts instead of aborting transactions, with minimal execution overhead. This protocol deals with write-write, read-write, write then read and read then write of same transaction conflicts. This mechanism can boost the serialized process for transactions that have long running queries.

A. New Correctness Criteria and Serialization

The basic serializability criteria of MMV2PL are given below.

- The basic correctness criteria that scheduler must treat write message of a transaction as first read then write message on given object. Such applications which universally imposes correctness criteria have very rare loss of consistency.
- Multiversion two phase locking protocol allows update transactions to use versions with assuring serializability to all transactions. This protocol restricted the transaction to use read for reading purpose only and never upgrade to write while the write operation restricted to read then write.

a) Read Rule

T_r, a transaction may lock on latest version of object from the database. It can execute if any of the given conditions satisfies on the object

1. If no lock set on object.
2. A Write lock set on the object by another transaction T_h.
3. Write lock set by same transaction.
4. Read lock set by another transaction;

If condition 1or 3 has satisfied, then Read lock set on object by transaction T_r. For condition 2 read lock must wait until incompatible lock will release. For condition 4, lock given in most algorithm as read locks will compatible with each other. We are modifying condition 3 by allowing read lock of transaction which has already implemented write lock on same object.

When same object locked by same transaction's write, read lock granted in proposed technique, other algorithm treats this lock as incompatible.

- **R|W 'Read first operation dependent' or 'Delay locking' or 'D-Serializability'**

CASE 1

$H = R_1(X), W_1(X)$

R1 has read lock then immediate write lock by the same transaction then lock will grant.

CASE 2

$H = R_1(X), R_2(X), W_1(X)$

We are proposing delay locking for this ***'Read First operation dependent'***, or ***'Delay locking'*** or ***'D-Serializability'*** does not block the write operation of the same transaction.

If other transaction comes between read and write of the same transaction, in above case read lock has applied by both transaction and write lock will have to wait and locking delays till read of this different transaction releases the lock.

Multiversion two-phase locking will block this write lock. We are modifying Multiversion mixed method by delaying the locking of the

same transaction until another read operation released, then only write lock can grant to object.

b) Write Rule

Write operation of transaction can lock latest version of the object. It can execute if any of the given conditions satisfies on the object

1. If no lock set on object.
2. A Write lock set on the object by another transaction T_h.
3. A Read lock set by another transaction.
4. Read lock set by same transaction.

If condition 1 or 4 will satisfy, then Write lock will set on object by transaction T_{rw}. For conditions 2 and 3 Write lock must wait until incompatible lock will released.

We are modifying execution rule for condition 4. When write lock requested by same transaction which has already granted read lock on the same object, as read and write are incompatible. In other cases, this write must have to wait; while we are allowing this write.

When same object locked by same transaction's read, write lock granted in proposed technique. Other algorithm treats this lock as incompatible.

- **W|R 'First write operation dependent' or 'Relaxed Locking' or 'R-Serializability'**

CASE1

$H = W_1(X), R_1(X)$

Write lock is exclusive for all operations whether it is read or write. The read request of other transaction must wait, but in our proposed technique, we are giving flexibility, and this write lock on the object will compatible with read lock of the same transaction.

This proposed ***'Write First operation dependent'*** or ***'Relaxed locking'*** or ***'R-Serializability'*** this read after the write will either read the latest committed value of X which will be an inconstant value as this read comes after write of the same transaction, or this should read changed value by the write request of the same transaction which is not committed yet.

We are proposing correctness that relaxes the rule of execution for lock. This read after write will ignore as such read will lead to abort in other techniques. By ignoring these reads will allow more transactions to commit and this will increase consistency and over all system's performance.

This technique will give more flexibility and will improve concurrency control technique. Nevertheless, if one protocol built on W|R-serializability, then it can add more concurrency and smooth performance. Further, starvation of transactions mitigated by this proposed protocol.

c) Commit Rule

When all operation requests of any transaction granted locks and update takes place at secondary storage then the commit happens. Locks held by an object released only when the transaction commits. The new version of the object will create. A new timestamp assigned to latest version. For reliability reasons all locks are held until the transaction terminates. The commit is the guarantee of termination of the transaction.

IV. METHODOLOGY FOR EXPERIMENTAL ANALYSIS

We have simulated this new technique in standard environment. We have designed our own simulation model for concurrency control using visual C++ [22,23, 24]. We have implemented both MV2PL and new serialized MMV2PL which includes test of 10 KLOC. These algorithms are using nested transaction. The serial schedule consists of sequence of operations from various transactions where the operations belonging to each transaction appear together in that schedule.

Transaction Generator

Operations of transactions arranged in queue and served as they come to system, with timestamp given when transaction created. We have a *transaction generator* that is responsible for starting of transactions. Transactions contain operations read, write and read-only. The transactions generator issues FCFS for the retrieval of transactions. The Read-only transaction has assigned a proper timestamp which clearly corresponds to its actual start time and it always refer to a right version. Each version is having maximum timestamps. The transaction identifier sequence generator assigns a unique identifier to each transaction.

Transaction Manager

This simulation examines the behavior of the algorithms under a mix of transactions for which our methodology designed. The mix used here consists of Update transactions and Read-only transactions. Requesting transaction's timestamp is always less than version's timestamp. When different versions created each of them assigned with the write-stamp which corresponds to its commit time.

Database

It primarily stores n data items which shared among all transactions, namely Update transactions and Read-only transactions. The transactions can vary from small to very large fraction applied on standard data set. All data manipulation operations result in one or more versions. The references to these versions added to the version chain of the manipulated data giving timestamp for each version. The data stored in secondary storage classified into three groups: small, medium and large, according to their respective size. For instance, small size denotes easiness and simplicity.

Concurrency Control Manager

Operations of transactions synchronized by mutual exclusion. A transaction must acquire a lock before accessing a data item from the database. Transactions approved if the lock mode is compatible with the existing transaction. In case of incompatibility, new transaction will have to wait until incompatible transaction will commit or abort, only after termination of transaction lock will release. Conflicts of transaction solved by compatibility matrix of lock manager. Conflicts between update transactions and Read-only transactions have been quite likely to occur for the larger transaction sizes. In the case of standard locking, the lock conflicts are in the form of transaction blocking, that means transaction must wait until incompatible transaction will release.

Besides the start timestamp and the transaction ID, the transaction object keeps track of the compatibility matrix. The compatibility matrix implemented as a list of locks implemented on data item. All operations that interact with the database for reading or

writing data requires the compatibility matrix, which used as a parameter to these operations.
The system may lead to a thrashing behavior because of these conflicts where most
transactions in the system blocked. These locks conflict which blocked are the main cause of deadlocks. The aim of this simulation is to observe increasing concurrency by having multiple versions to eliminate these conflicts.
Deadlock Manager
The deadlock occurs where two transactions blocked indefinitely awaiting the release of each other's locks. Deadlocks removed by periodically checking for cycles. Deadlocks resolved by deadlock handling method and this will rollback one of the transactions in a deadlock cycle according to one of the victim selection policies.
Deadlock will be resolved by aborting the victim transaction. Aborted transaction will be restarted by the system later without user intervention.

V. MODIFIED MV2PL ALGORITHM

Proposed concurrency control technique differentiates between Read-only transactions (R_o) and Update transactions. Update transaction can have two operations read T_r and read then write T_{rw}. Every data version assigned with single timestamp. The value of timestamp takes from a counter Ts - counter and it increments during commit processing.

A. Update Transaction

MMV2PL never Upgrade Read Tr to Write Tw, if transaction wants to access object for read and then write in the later stage this must apply Trw. Update transactions follow rigorous two-phase locking.

- ***When an update transaction wants to read (Tr) on data Q***

When an UPDATE transaction wants to *Read (Tr) on* Data Q
If (lock=NO_LOCK ||lock=LOCK_RRA)
Set LOCK_RRA1(Q) // It obtains a shared lock on it and reads the latest version.
else
If lock=LOCK_RMA2 (Q)
Wait; // Exclusive lock is there this read must Wait
else
If lock=LOCK_ RMA1(Q) //If Exclusive lock of same transaction is there
Ignore; // The read operation will be ignored. *(R serializability)*

- ***When an update transaction wants to write (Trw) on data Q***

When an UPDATE transaction wants to *write (Trw)* on Data Q,
If lock=NO_LOCK
Set LOCK_ RMA1(Q) // It obtains Exclusive lock on; it then creates a new version of the item and sets this version's timestamp to ∞ (infinity).
Elseif
Lock=LOCK_ RRA2(Q) || LOCK_ RMA2(Q)
Wait;// If Shared lock is there; Exclusive will Wait.
Else
If lock= LOCK_ RRA1(Q) // If Shared lock of same transaction is there Exclusive lock granted.
elseif (lock=LOCK_ RRA1(Q) && lock=LOCK_ RRA2(Q))
Delay Wait; // When an update transaction Ti completes, commit processing occurs:
Ti sets timestamp on the versions it has created to Ts-counter + 1
Ti increments Ts-counter by 1

B. Read-only Retrieval Transaction (Ro)

Read-only will use for reading purpose only, this may read old but consistent value. Read-only follows Multiversion timestamp-ordering protocol for performing reads.

- ***When transaction wants to read (Ro) the data***

If Tro>Ts Read-only transactions that start after T_i increments ts-counter will see the values updated by T_i.
Read Q
Else
Tro<Ts Read-only transactions that start before T_i increments the ts-counter will see the value before the updates by T_i.
Read Q

VI. EXPERIMENT RESULTS AND DISCUSSION

We consider a database with D objects that can apply lock in exclusive and shared mode. The performance analysis of concurrency control methodology is discussing how the objects accessed and locks obtained by a transaction. In standard locking, locks have been usually requested on demand, which referred as *dynamic locking* (DL), that means locks required for assessing objects of database to execute transactions. These requests for locks are unknown to priori and requested before the transaction begins its execution.
. This algorithm dealing with the lock contention between long-running Read-only queries and short Update transactions.
To achieve higher transaction throughputs, many transactions should run in parallel which results into high probability of lock conflicts. The performance of such systems reduces because of these lock conflicts. We are using wide range of Read-only and Update transactions with varied degree of conflicts among update transactions and different versions of object.
Experiments and Discussion
Example
We are taking simple example with 10 data item and 5 transactions each transaction is having 3 operations.
These operations are generated randomly, and they can be either read or write. When all operations of same transactions applied lock on data item commit happens, if incompatible lock is there other requesting operation will wait until this exiting lock will release.
Results of MV2PL
Operation no. 1 (Read Operation) of Transaction no. 4 has acquired a shared lock on DATA_ID = 3
Operation no. 1 (Read Operation) of Transaction no. 1 has acquired a shared lock on DATA_ID = 9
Operation no. 1 (Read Operation) of Transaction no. 2 has acquired a shared lock on
DATA_ID = 3

Operation no. 1 (Read Operation) of Transaction no. 5 has acquired a shared lock on DATA_ID = 1
Operation no. 2 (Read Operation) of Transaction no. 1 has acquired a shared lock on DATA_ID = 7
Operation no. 2 (Read Operation) of Transaction no. 5 has acquired a shared lock on DATA_ID = 5
Operation no. 3 (Read Operation) of Transaction no. 5 has acquired a shared lock on DATA_ID = 10
Operation no. 1 (Write Operation) of Transaction no. 3 has acquired an exclusive lock on DATA_ID = 2
Operation no. 2 (Write Operation) of Transaction no. 4 has acquired an exclusive lock on DATA_ID = 7
***** Transaction no. 5 is committed. ****
Transaction no. 5 has released all the locks.
Operation no. 3 (Read Operation) of Transaction no. 1 has acquired a shared lock on DATA_ID = 8
Operation no. 2 (Read Operation) of Transaction no. 2 has acquired a shared lock on DATA_ID = 6
Operation no. 2 (Write Operation) of Transaction no. 3 has acquired an exclusive lock on DATA_ID = 9
Transaction no. 4 is aborting because it has already executed a write operation on DATA_ID = 7 because Rewriting is not allowed
***** Transaction no. 1 is committed. ****
Transaction no. 1 has released all the locks.
Operation no. 3 (Read Operation) of Transaction no. 2 has acquired a shared lock on DATA_ID = 10
***** Transaction no. 2 is committed. ****
Transaction no. 2 has released all the locks.
Operation no. 3 (Read Operation) of Transaction no. 3 has acquired a shared lock on DATA_ID = 10
***** Transaction no. 3 is committed. ****
Transaction no. 3 has released all the locks.
****
****
No. of transactions committed = 4
No. of times transactions were sent to wait state = 0
No. of time transactions were aborted = 1
No. of times transactions were rolled back 2
Transaction no. 1, 2, 3, 5, were committed.
Transaction no. 4, were aborted
Transaction no. 2, 5, were rolled back
Results of Modified MV2PL
Operation no. 1 (Read Operation) of Transaction no. 4 has acquired a shared lock on DATA_ID = 3
Operation no. 1 (Read Operation) of Transaction no. 1 has acquired a shared lock on DATA_ID = 9
Operation no. 1 (Read Operation) of Transaction no. 2 has acquired a shared lock on DATA_ID = 3
Operation no. 1 (Read Operation) of Transaction no. 5 has acquired a shared lock on DATA_ID = 1
Operation no. 2 (Read Operation) of Transaction no. 1 has acquired a shared lock on DATA_ID = 7
Operation no. 2 (Read Operation) of Transaction no. 5 has acquired a shared lock on DATA_ID = 5
Operation no. 3 (Read Operation) of Transaction no. 5 has acquired a shared lock on DATA_ID = 10
Operation no. 1 (Write Operation) of Transaction no. 3 has acquired an exclusive lock on DATA_ID = 2
Operation no. 2 (Write Operation) of Transaction no. 4 has acquired an exclusive lock on DATA_ID = 7
***** Transaction no. 5 is committed. ****
Transaction no. 5 has released all the locks.
Operation no. 3 (Read Operation) of Transaction no. 1 has acquired a shared lock on DATA_ID = 8
Operation no. 2 (Read Operation) of Transaction no. 2 has acquired a shared lock on DATA_ID = 6
Operation no. 2 (Write Operation) of Transaction no. 3 has acquired an exclusive lock on DATA_ID = 9
Transaction no. 4 is Write Lock ignoring lock because it has already executed a write operation on DATA_ID = 7 because writing is not allowed
***** Transaction no. 4 is committed. ****
***** Transaction no. 1 is committed. ****
Transaction no. 1 has released all the locks.
Operation no. 3 (Read Operation) of Transaction no. 2 has acquired a shared lock on DATA_ID = 10
***** Transaction no. 2 is committed. ****
Transaction no. 2 has released all the locks.
Operation no. 3 (Read Operation) of Transaction no. 3 has acquired a shared lock on DATA_ID = 10
***** Transaction no. 3 is committed. ****
Transaction no. 3 has released all the locks.
****
****
No. of transactions committed = 4
No. of times transactions were sent to wait state = 0
No. of time transactions were aborted = 0
No. of times transactions were rolled back 2
Transaction no. 1, 2, 3, 4, 5, were committed.

The correctness checks associated with read and write operations of transactions. Each executed operation of the transaction uses this information in order to solve conflicts efficiently. For this purpose, the possible read/write, write-write, read then write/read, read then write/write conflicts identified. Serializability defined in algorithm correct them and provide locks accordingly. When all operations of particular transaction granted locks successfully and changes takes place in secondary memory commit happens.

The experiment has taken a window size *N*, which implies having *N* concurrent transactions, *N* transactions picked from the input stream and all of them start, then they execute, and finally they try to lock and commit one after the other. Here, the transactions that wait during the execution will again tried to lock if inconsistent lock will be removed. Transactions that abort acquire a new timestamp immediately and their executions moved to the next window. These executions are in serial.

The transaction does not terminate because indefinite postponement or deadlocks. This postponement avoided by a fair scheduling strategy of new technique.

The performance evaluation of transactions in concurrency control systems calculated by number of commits, abort, wait and deadlocks. Number of waits for resource request has the primary effect on performance, whereas deadlocks and aborts have a secondary effect. Performance degradation takes place due to transaction blocking (wait), restarts, or both. When restarted transaction have lock conflicts with the transaction with which they already had lock conflicts leads to cyclic restarts. This generates repeated deadlocks.

The parameters that impact the effectiveness of both MMV2PL and MV2PL, such as the number and size of concurrent transactions, the percentage of conflicts among concurrent transactions, and the average number of times

that a transaction aborted until it commits.

We have taken total 1000 transactions, where total number of transactions in one run is 100 and total number of runs are 10. The total numbers of operations on object and class in transaction vary in each run. Similarly, Read-only and Update transactions also may vary in each run. These transaction's operations are random, and each operation read or write (read then write) are accessing objects.

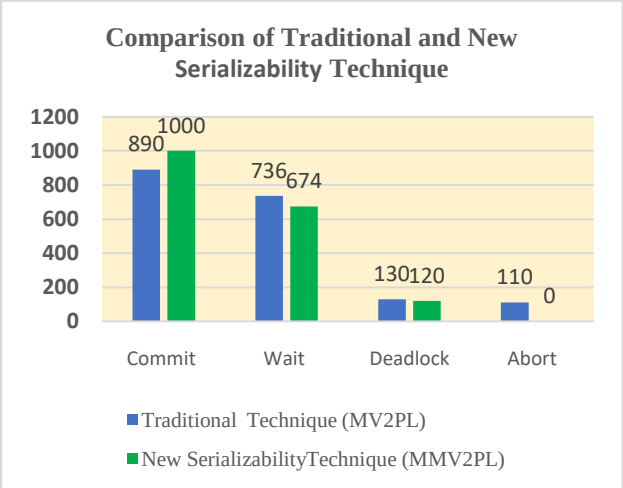

Figure 1

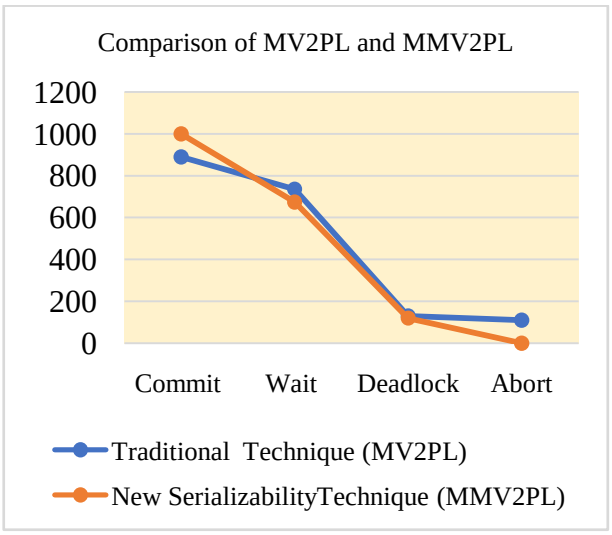

Figure 2

Comparison of Performance by New Technique and Traditional Serializability Techniques

In Figure 1, MMV2PL shows 30% more performance compared to MMV2PL.

New technique is giving better number of commits, a smaller number of waits and deadlocks and almost zero abort. The purpose for the algorithm fulfills user's expectation in terms of high number of commit and better throughput.

Our experimental results show that Modified Multiversion Two Phase Locking MMV2PL achieves more than compared to MV2PL in transaction processing throughput for high-contention scenarios.

VII. CONCLUSION

In this paper we are presenting a new technique MMV2PL, a conflict resolution mechanism which proves its correctness and showed its effectiveness in various scenarios. For any update intensive applications domain, Read-only transactions are giving better performance by reading slightly older versions. In this new technique the latest versions have been given to objects to enhances concurrency control for Read-only transactions. New correctness criteria allow transactions that used to abort by traditional concurrency control techniques. This Exclusive lock allows Shared lock of the same transaction. The new correctness criteria improve the concurrency control within update transactions, wherein, same cases in the traditional algorithms cause unnecessary abort during concurrent execution. The new technique achieves a good transaction completion ratio even in the cases where Updates frequently conflicted with each other because of updating the same data object.

High number of commits in transactions are achieved in this new technique when compared to the older methodology for large number of concurrently executed updates. The new technique is generating smaller number of waits, deadlocks, rollbacks and aborts. Thus, the proposed technique performs better for new correctness criteria by giving a greater number of commits as compared to traditional concurrency control techniques.

REFERENCE

1. K.P. ESWARAN, J.N. GRAY, R.A. LORIE, AND I.L. TRAIGER: "The Notions Of Consistency And Predicate Locks In A Database System", Communications Of The ACM, November 1976 Of Volume 19, Pages 624-633.
2. DAVID P. REED: "Implementing Atomic Actions On Decentralized Data", ACM Transactions On Computer Systems, Vol 1, Feb 1983, Pages 3-23.
3. H.T. KUNG, JOHN T. ROBINSON: "On Optimistic Methods For Concurrency Control", ACM Transactions On Database Systems, Vol. 6, No. 2, June 1981, Pages 213-226.
4. RICHARD E. STEARNS AND DANIEL J. ROSENKRANTZ: "Distributed Database Concurrency Controls Using Before-Values", L981 ACM, Pages 74-83.
5. CHRISTOS H. PAPADIMITRIOU, PARIS C. KANELLAKIS: "On Concurrency Control By Multiple Versions", ACM Transactions On Database Systems, Vol. 9, No. 1, March 1984, Pages 89-99.
6. PHILIP A. BERNSTEIN, NATHAN GOODMAN: "Multiversion Concurrency Control-Theory And Algorithms", ACM Transactions On Database Systems, Vol. 8, No. 4, December 1983, Pages 465-483.
7. HENRY F. KORTH: "Locking Primitives In A Database System", Journal Of The Association For Computing Machinery, Vol 30, No1, January 1983, Pages 55-79.
8. CHRISTOS H. PAPADIMITRIOU: "On Concurrency Control By Multiple Versions", ACM Transactions On Database Systems, Vol. 9, No. 1, March 1984, Pages 89-99.
9. RONG SUN, GOMER THOMAS: "Performance Results On Multiversion Timestamp Concurrency Control With Predeclared Write Sets", 1987 ACM, Pages 177-184.
10. DIVYAKANT AGRAWAL, AMR EL ABBADI, And AMBUJ K. SINGH: "Consistency And Orderability: Semantics-Based Correctness Criteria For Databases", ACM Transactions On Database Systems, Vol 18, No. 3, September 1993, Pages 460-486.
11. MIHALIS YANNAKAKIS: "Issues Of Correctness In Database Concurrency Control By Locking", ACM 1981, Pages 363-367.
12. RICHARD E. STEARNS, DANIEL J. ROSENKRANTZ: "Distributed Database Concurrency Controls Using Before-Values", Computer Science Department, State University Of New York, L981 ACM, Pages 74-83.

13. CHANJUNG PARK, SEOG PARK: "Alternative Correctness Criteria For Multiversion Concurrency Control And A Locking Protocol Via Freezing", 1997 IEEE, Pages 73-81.
14. RYOJI KATAOKA, TETSUJI SATOH, USHIO INOUE: "A Multiversion Concurrency Control Algorithm For Concurrent Execution Of Partial Update And Bulk Retrieval Transactions",1991 IEEE, Page 131-136.
15. DIVYAKANT AGRAWAL, SOUMITRA SENGUPTAT: "Modular Synchronization In Multiversion Databases: Version Control And Concurrency Control", ACM 1989, Pages 508-417.
16. ZVI M. KEDEM, ABRAHAM SILBERSCHATZ: "Locking Protocols: From Exclusive To Shared Locks", Journal Of The Association For Computing Machinery, Vol 30, No 4, October 1983, Pages 787-804.
17. MICHAEL J. CAREY, WALEED A. MUHANNA: "The Performance Of Multiversion Concurrency Control Algorithms", ACM Transactions On Computer Systems, Vol. 4, No. 4, November 1986, Pages 338-378.
18. HENRY F. KORTH, ABRAHAM SILBERCHATZ, S. SUDARSHAN: "Concurrency Control: Database System Concepts (Forth Edition)", Pages 591 -617.
19. RAMEZ ELMASRI, SHAMKANT B. NAVATHE: "Multiversion Concurrency Control Techniques", Pages 585-587.
20. KRITHI RAMAMRITHAM, PANOS K. CHRYSANTHIS: "A Taxonomy Of Correctness Criteria In Database Applications", The VLDB Journal, Springer-Verlag (1996), Pages 85–97.
21. MICHAEL J. CAHILL, UWE RÖHM ROEHM, ALAN D. FEKETE: "Serializable Isolation For Snapshot Databases", SIGMOD'08, June 9–12, 2008, Vancouver, BC, Canada, Pages 729-738.
22. 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, Pages 245-251.
23. 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, Pages 492-498.
24. SONAL KANUNGO, R.D. MORENA, "Evaluation Of Multiversion Concurrency Control Algorithms", International Journal Of Research In Electronics And Computer Engineering A Unit Of I2OR, Vol 6, Issue 3, July-September 2018, Pages 807-813.
25. MOHAMMAD DASHTI, SACHIN BASIL JOHN, AMIR SHAIKHHA, AND CHRISTOPH KOCH, "Transaction Repair For Multi-Version Concurrency Control", SIGMOD'17, May 14-19, 2017, Chicago, Illinois, USA, Pages 235-250

AUTHORS PROFILE

Dr. Sonal Kanungo, Assistant Professor
sonalkanungo@gmail.com
Sonal has an excellent Teaching experience for more than 10 years. Her research area is in Concurrency Control in DBMS and ODBMS. Her qualifications include PhD in Computer Science after completing B.C.A. and M.Sc. (IT). She has more than 6 years of Research experience in the field of Concurrency control.

Dr. R. D. Morena,
Professor rdmorena@rediffmail.com
He is working as a Professor in Department of Computer Science, Veer Narmad South Gujarat University and is associated with the Department since last 25 years. His qualifications include MCA, M.Phil and PhD in Computer Science. He was a member of Board of University Teachers (BUT) and currently is a member of Board of Studies (IT) at VNSG University. He is the co-coordinator of UGC supported Special Assistance Program (SAP) awarded to the Computer Science Department. He was a member of Academic Council and Faculty at various other Universities. He is giving service as an Editor and Reviewer for several National and International Journals. His research area is Data Management and is a PhD and M.Phil supervisor. Three PhD and eight M.Phil degrees have been awarded under his guidance.