ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 3, March 2015

Analysis and Comparison of Concurrency Control Techniques

Sonal Kanungo[1], Morena Rustom. D[2]

Smt.Z.S.Patel College Of Computer, Application,Jakat Naka, Surat[1]

Department Of Computer Science, Veer Narmad South Gujarat University, Surat.[2]

Abstract: In a shared database system when several transactions are executed simultaneously, the consistency of database should be maintained. The techniques to ensure this consistency are concurrency control techniques. All concurrency-control schemes are based on the serializability property. The serializability properties requires that the data is accessed in a mutually exclusive manner; that means, while one transaction is accessing a data item no other transaction can modify that data item.
In this paper we had discussed various concurrency techniques, their advantages and disadvantages and making comparison of optimistic, pessimistic and multiversion techniques. We have simulated the current environment and have analysis the performance of each of these methods.

Keywords: Concurrency, Locking, Serializability

1. INTRODUCTION

When a transaction takes place the database state is changed. In any individual transaction, which is running in isolation, is assumed to be correct. While in shared database several transactions are executes concurrently in the database, the isolation property may no longer be preserved. To ensure that the system must control the interaction among the concurrent transactions; this control is achieved through one of a variety of mechanisms called **concurrency-control schemes**.[1]

1.1 Concurrency control Techniques

The serializable transactions are executed one at a time, or serially, rather than concurrently. [4] All schemes we are going to discuss here are serializable. Serializability, or isolation, is the standard for ensuring atomicity. [1] In this paper we intent to compare the techniques

1.1.1 Lock-Based Protocols
1.1.2 Timestamp-Based Protocols
1.1.3 Validation – Based Protocols
1.1.4 Multiversion Schemes

1.1.1 Lock-Based Protocols

In Lock Based Protocols the Lock mechanism is used for concurrent access to a data item. Permission is given to access a data item only if it is currently holding a lock on that item. Data items can be locked in two modes; either exclusive (X) mode or shared mode (S). [1] For transactions that can both read and write from the data item X, **exclusive-mode lock** is given. For transactions that can read, but cannot write on item S, **shared-mode lock** is given to data item. Transaction can proceed only after request is granted. [11]

A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by the other transactions. N number of transactions can hold shared locks (S) on an item. But if any transaction holds an exclusive lock (X) on the item, no other transaction may hold any lock on that item. In this condition, a lock cannot be granted and the requesting transaction has to wait until all incompatible locks held by other transactions are released. The lock is then granted. [1]

1.1.2 The Two-Phase Locking Protocol

Transaction can always commit by not violating the serializability property. If obtaining and releasing locks are done improperly, it will leads to inconsistency and deadlocks can occur. For transactions to be serial, all access to data must be serialized with respect to access by other transactions.

To ensure that the conflicting operations of the multiple transactions are executed in the same order, a restriction is imposed. Any transaction is not allowed to obtain new locks till it had released a lock. This restriction is called **Two Phase Locking**. The first phase is known as the growing phase, in which a transaction acquires all the locks it needs. The second phase is known as the shrinking phase, where the process releases the locks. [1]

If a process fails to acquire all the locks during the first phase, then it is obligated to release all of them, wait, and then start over. [12] This protocol ensures conflict-serializable schedules. [1] The optimality of two-phase locking implies that, in absence of any information about the transactions or the database, all locking protocols must be two-phase. [14]

Further the Two Phase Locking can be of two types:

1.1.2.1 Strict two-phase locking: It is necessary to hold write locks until after a transaction commits or aborts to ensure serializiblity. As per two-phase locking (2PL) rules, to ensure serializability the read locks may be released earlier. This implies that the read locks can be released when the transaction terminates (i.e., when the scheduler receives the transaction's commit or abort), but write locks must be held until after the transaction commits or aborts. [6] Transaction must hold all its exclusive locks till it commits or aborts and no cascading rollback takes place.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 3, March 2015

1.1.2.2 Rigorous two-phase locking: It is even stricter; all locks (S or X) are held until commit/ abort takes place and no cascading rollback happens. Transactions can be serialized in the order in which they commit. Deadlocks and Starvations are main drawbacks of these protocols.

1.1.3 Timestamp-Based Protocols

For keeping information about the precise order of arrival of execution, requests cannot be taken into account by any locking algorithm. In contrast, algorithms are implemented by queues or timestamps. [14] In this algorithm, the timestamp is given to a transaction when it begins. [14] The timestamp has to be unique with respect to the timestamps of other transactions. Here, W-timestamp is the largest time-stamp of any transaction that executed write successfully and R-timestamp is the largest time-stamp of any transaction that executed read successfully are kept. The protocol manages concurrent execution such that the time-stamps determine the serializability order. [1]

When a process tries to access a data, the data's *read* and *write* timestamps will be older than the current transaction's. If this is not the case, and the ordering is incorrect, this implies that a transaction that started later than the current one accessed the data and committed. In this case the current transaction is too late and has to abort. The rule here is that the lower numbered transaction always goes first *read* it and which committed transaction last *wrote* it. [12]

The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. Timestamp protocol ensures freedom from deadlock as no transaction ever waits. But the schedule may not be cascade-free, and lead to non-recoverable situation. [1]

1.1.4 Validation-Based Protocols

This is also called as **optimistic concurrency control** since transaction executes fully in the hope that all will go well during validation. [1] The methods used are "optimistic" in the sense that they rely mainly on transaction backup as a control mechanism, "hoping" that conflicts between transactions will not occur [6]. These methods are "optimistic" in the sense that they rely for efficiency on the hope that conflicts between transactions will not occur. [5]

Validation based protocols works under assumption that the read and write conflicts among transactions occurs rarely. This allows uncontrolled access to shared data objects during transaction processing. Before a transaction commits, the DBMS has to validate that no conflict had occurred. Conflict resolution mainly leads to transaction abort. [5] Where a majority of transactions are read-only transactions, the rate of conflicts among transactions may be low.

This concurrency-control scheme imposes overhead of code execution and possible delay of transactions.

The execution of transaction is done in three phases. These phases depend on whether it is a read-only or an update transaction. The phases are as follows:

In **Read and execution phase,** the transaction writes only to temporary local variables. It performs all write operations on temporary local variables, without update.

In **Validation phase,** the transaction performs a "validation test" to determine whether local variables are written without violating serializability.

In **Write phase,** in case the transactions are validated, the updates are applied to the database; otherwise the transaction is rolled back. [1]

Each transaction goes through these three phases and in that order. In Optimistic scheme, we do not lock the records and therefore no deadlocks occur. [19]

1.1.4 Multi version Schemes

There are two ways to ensure serializability, either by delaying an operation or aborting the transaction that issued the operation. For example, a read operation may be delayed because the appropriate value has not been written yet; or it may be rejected because the value that it was supposed to read has already been overwritten. These difficulties could be avoided if old copies of each data item were kept in a system. [1]

In a multiversion scheme, each write on any data item, say X, produces a new copy (or version) of X. For each read on X, it selects one of the versions of X to be read. Since writes do not overwrite each other and since reads can read any version, it has more flexibility in controlling the order of *reads* and *writes*. This approach maintains a number of versions of a data item and allocates the right version to a read operation of a transaction. [21]

In a Multiversion scheme, a read operation is never rejected. However, this scheme needs significantly more storage (RAM and disk) for maintaining multiple versions. In order to check unlimited growth of versions, a cleanup action is run when some criteria is satisfied. [11]

Multiversion Scheme is often used along with Time stamping and Two-phase locking.

1.1.4 .1 Multiversion Timestamp Ordering

The timestamps are used to label the versions. When a **read** operation is issued, an appropriate version of data based on the timestamp of the transaction is selected, and the value of the selected version is returned. **Reads** never have to wait as an appropriate version is returned immediately. [1]

When a transaction issues a write step on some entity X, we might choose not to overwrite the old value of X by the new one, but to keep both versions. If subsequently another transaction reads X, we have the option of supplying to it either version, whichever serves serializability best, as that is the final accepted action. [18] In this scheme, each data item X has a sequence of versions <X1, X_2,...., X_m>.

Each version data contains three data fields: **Content** is the value of version X_k.**W-timestamp** (X_k) is timestamp of the transaction that created (wrote) version X_k, **R-timestamp**(X_k) is largest timestamp of a transaction that successfully read version $X_{k[18]}$

Transaction reads the most recent version that comes before it in time. If the transaction attempts to write a

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 3, March 2015

version that some other transaction would have already *read*, then that write cannot succeed. [1]

1.1.4.2 Multiversion Two-Phase Locking

The multiversion two-phase locking protocol attempts to combine the advantages of multiversion concurrency control with the advantages of two-phase locking. This protocol differentiates between read-only transactions and updated transactions.[18]

Update transactions acquire read and write locks and hold all locks up to the end of the transaction; that is, the update transactions follow rigorous two-phase locking. [15] In this locking mechanism, two versions for each item X are kept; one version must always have been written by some committed transaction. The second version X is created when a transaction acquires a write lock on the item. Other transactions can continue to read the committed version of X while the transaction is holding the write lock. Transaction can write the value of X as needed without affecting the value of the committed version X. However, once a transaction is ready to commit, before it commits, it must obtain a *certify* lock on all items that it currently holds write locks on. The certify lock is not compatible with read locks. Hence the transaction may have to delay it's commit until all it's write-locked items are released by any reading transactions in order to obtain the certify locks. [8]

The Update transactions perform rigorous two-phase locking and they hold all locks until the end of the transaction. Therefore according to their commit order they can be serialized.

2. RELATED WORK

Database concurrency control is an active area of research and has resulted in the development of many protocols for achieving serializability. The basic mechanisms used by the protocols are of locking, timestamps, and multiple versions. [5, 7, 16]

Korth[1] This work discuss various concurrency-control schemes. All these schemes follow serializability. Conflicts are handled either by delaying or aborting the transactions. The most common schemes are of locking protocols, timestamp, validation techniques, and multiversion schemes.

Bharat Bhargava[2] (Concurrency Control in Database Systems) This work presented several classes of concurrency control approaches and presented a short survey of ideas that have been used for designing flexible concurrency control algorithms.

H.T. Kung and John T. Robinson [5] (**On Optimistic Methods for Concurrency Control**) In this paper, two families of non-locking concurrency controls are presented. The methods used are "optimistic" in the sense that they rely mainly on transaction backup as a control mechanism, "hoping" that conflicts between transactions will not occur. Most important outcome of this paper is; 'locking may be necessary only in the worst case'.

ALEXANDER THOMASIAN [7] (Concurrency Control: Methods, Performance, and Analysis) In this paper, the performance of the locking model is analyzed. This article is to provide ideas of factors leading to performance degradation. It also summarized the conclusions of previous simulation and analytic studies regarding the relative performance of concurrency control methods and survey methods applicable to the analysis of standard locking, restart-oriented locking methods, and optimistic concurrency control.

BERNSTEIN, P. A., AND GOODMAN [22] (Multiversion Concurrency Control-Theory and Algorithms) In this paper they extended concurrency control theory for the translation aspect of multiversion databases. The main idea is one-copy serializability. Any execution of transactions in a multiversion database is one-copy serializable. They applied the theory to three multiversion concurrency control algorithms wherein one algorithm uses timestamps, one uses locking, and one combines locking with timestamps.

CHRISTOS H. PAPADIMITRIOU, PARIS C. KANELLAKIS [19] (**On Concurrency Control by Multiple Versions, A Theorem in Database Concurrency Control).** This paper examined the problem of concurrency control when the database management system supports multiple versions of the data. They characterized the limit of the parallelism achievable by the multiversion approach and demonstrated the resulting space-parallelism trade-off.

3. METHODOLOGY

We simulated concurrency control environment using C++. A simple data structure was used for storing data randomly.We have generated transactions randomly. Every transaction had any of two operations, either Read or Write, which are also generated randomly. When a transaction was entered in system it had applied read or write on some data, concurrently other transactions are also running in system that also wants to apply read/ write on some data. The methodology for these protocols works as is described below.

3.1. Two phase Locking

When a Transaction's operation get executed it first checks whether it has a lock or no-lock on data. In case no-lock is found on data, a lock is applied to data. This lock can be shared (read S) or exclusive (write X). If both the operations gets lock (S,S) or (S,X) or (X,X) or (X,S), the transaction goes to process, where reading or writing on data takes place and unlocking is performed. Else, the operation goes to wait which means any of the two or both operations have found lock on that data.

In Process (Unlock) if timeout takes place it goes to rollback. In few cases the cascading Rollback is also found. This implies that after rollbacked by one transaction, the other transaction that apply operation on the same data will also be rollbacked or else transaction will commit.

We ran these transactions 100 times where there are 10 operations in each run ,We found that only 180 transactions are committed while others are either in wait or in rollback. We found that few transactions were committed and more number of waits was generated. This protocol is free from conflict serializiblity. However, we found overheads of lock.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 3, March 2015

3.2. Time Stamping

We used System clock for two timestamps; ReadTimestamp and WriteTimestamp for each data. When transaction is entered in system timestamp is given to transaction. When transaction wants to read some data the transactiontimestamp must be greater than last Write's timestamp, then only read will success and R=T(last Readtime stamp will assign to Transaction's timestamp), else rollback takes place implying that other operation is assigned to that data.

When transaction wants to write on some data, transactions timestamp should be greater than last Read and Writestamp then only write will succeed and W=T(last write timestamp is assign to transaction's timestamp) Else, rollback takes place implying that other operation is assigned to that data. If the Transaction's timestamp is less than last Write time stamp, transaction is ignored. (Thoms' write).

We ran these transactions 100 times where there are 10 operations in each run ,We found that only 288 transactions are committed while others are rollbacked. We did not find any wait here, but the rollbacks took place in a large number. If T is aborted and rolled back, any transaction T_1 that may have used a value written by T must also be rolled back. Similarly, any transaction T_2 that may have used a value written by T_1 must also be rolled back, and so on. This effect is known as **cascading rollback**. This protocol is free from conflict serializiblity, but lot of cascading rollbacks are generated. Overheads of ReadTimestamp and WriteTimestamp are also found.

3.3 Validation Based (Optimistic)

The techniques are called "optimistic" because they assume that little interference will occur and hence that there is no need to do checking during transaction execution. This Protocol is best when there are Read only transactions and when conflicts are not found. Here we allow all transactions to perform locally than we check whether conflicts are there (thru Validation). In case the conflicts are found, we took time stamping for both operations. If the first operation is its end Timestamp it will succeed. Else it will be roll-backed. In case the second operation is its *Start timestamp* and the first operation is its *End timestamp*, then it will succeed or else rollback.

We ran these transactions 100 times where there are 10 operations in each run . We found that only 333 transactions are committed which are mostly Read only. We also found that if there are no conflicts, commits are more. However if conflicts occurs, it generates a lot of rollbacks. In the optimistic concurrency control we do all the checks at once. Hence, we allow the transactions to execute with a minimum of overhead until the validation phase is reached. If there is a little interference among transactions, most will be validated successfully. However, if there are several interferences, many transactions that execute to completion will have their results discarded and must be restarted later. Under these circumstances, optimistic techniques do not work well.

3.4. Multiversion

In multiversion we generate new data with every successful write operation. Here we examined that for write operation if timestamp of given write operation is greater that timestamp of last write, we create new data and if it is equal to last write, we overwrite data. That means with every successful write we generated new version of data. Read operation is always success, because it always found data.

We ran these transactions 100 times where there are 10 operations in each run , we found 666 transactions are committed while others are in Rollback, We found more commit here and less Rollback. On the flip side, generation of new data with every successful write needs more space. Conflicts between transactions are resolved through rollbacks, rather than waits which are be expensive.

This emerges as the best protocol for large database.

4. COMPARISON

4.1. Performance Comparison

Locking protocols are good for update-intensive applications while for read only optimistic protocols are good. This is because there are no unnecessary overheads of locking of read-only transactions and will give good results. The performance is degraded with standard locking because blocking is done if transactions are not compatible with each other, whereas transaction restarts to resolve deadlocks have a secondary effect on performance which further leads to thrashing. [7] Timestamps are used to decide the older-younger relationships. Timestamp can give better results if some available information about the transactions or the database can be used for increasing concurrency. [11]

In a locking approach, having them wait at certain points, while in an optimistic approach backing them up controls the transactions. In multiversion scheme a read operation is never rejected, while large parts of the database reside on secondary storage. The overhead of keeping multiversion of data needs large storage space. For large database multiversion is considered to be best.

4.2. Serializability

Locking ensures serializability for any types of transactions whether it is Read only or Update-intensive which could operate concurrently with a given transaction. It is good for update-intensive applications because it is safe [6]. The timestamp-ordering protocol ensures conflict serializability. This is because conflicting operations are processed in timestamp order. [1] Transaction can read the same item at different times, conflict-free. [6]

The Optimistic Concurrency Control works on assumption that conflicts between transactions are rare. It does not require locking. Transaction is executed only after the validation. That is because the serializability order is not pre-decided and relatively less transactions will have to be rolled back if there are mostly read only transaction.

The multiversion two-phase locking protocol attempts to combine the advantages of multiversion concurrency control with the advantages of two-phase locking thereby providing serializable schedules. [22]

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 3, March 2015

4.3. Rollback and Deadlock handling

4.3.1 Locking Protocols

The deadlock prevention or detection in 2PL and other locking techniques is much more complex and costly. Storage overhead is increased because of deadlock [6] with locking and the blocked transactions. Processing of these overheads is high as keeping track of locks and the queue waiting for data access is difficult.

The deadlocks are found in most locking protocols. **Starvation** is also possible if concurrency control manager is badly designed. For example: A transaction may be waiting for an X-lock on an item, while a sequence of other transactions requests and are granted an S-lock on the same item. [1] The same transaction is repeatedly rolled back due to deadlocks. [9]

This protocol is inefficient because of locking overhead, possibility of deadlock and waits for locked data. [5] Two-phase locking does not ensure freedom from deadlocks. Cascading roll-back is possible under two-phase locking. [15] To allow a transaction to abort itself when mistakes occur, locks cannot be released until the end of the transaction. This may again significantly lower concurrency. [4]

Locking is done even for read-only transactions, which does not affect the integrity of the data. [5] If the locking protocol is not deadlock-free, deadlock detection must be considered to be part of lock maintenance overhead. There are no general-purpose deadlock-free locking protocols for databases that always provide high concurrency. [5]

4.3.2 Timestamp protocol

Timestamp protocol ensures freedom from deadlocks, as no transaction has to wait for other. However, there is a possibility of starvation of long transactions if a sequence of conflicting short transactions causes repeated restarting of the long transaction. In such cases cascading rollbacks are unavoidable. [17] However, this protocol enhances concurrency over two phased locking because the transactions do not block each other needlessly. It is different from locking, because the blocked transaction aborts rather than waits for access.

4.3.3 Optimistic Protocol

Optimistic protocol is different from locking, because they abort blocked transaction rather than sending them for waits. [5] The performance degradation occurs with standard optimistic approach due to rollback when a conflict happens. In an optimistic approach, the major difficulty is starvation. The validation scheme automatically guards against cascading rollbacks, since the actual writes take place only after the transaction issuing the write has committed.

4.3.4 Multiversion

In multiversion two phase locking, to detect deadlocks, the algorithm can use a directed blocking graph whose nodes are the transactions, and there is a deadlock if the graph has a cycle. To resolve deadlocks caused by certify-locks, the system should force one or more transactions to give up enough of their certify-locks to break the deadlock; these transactions can try later to get these locks back. To break deadlocks the system must abort one or more transactions, cascading aborts are also possible if the algorithm allows transactions to read uncertified versions. [22]

5. RESULT AND ANALYSIS

Transactions are generated randomly on random data where Read and write operations are also randomly performed on data . 1000 transactions are generated on 100 individual runs where each run where each run have 10 transactions, on which we have calculated results for total number of committed transaction, rollback transitions, and wait transaction for 2pl, Timestamp, Optimistic and Multiversion.

Average of transactions

	Number of runs for Transactions	Transaction in each run	Committed Transaction	Rollback Transaction	Wait Transaction
2PL	100	10	180	370	550
Timestamp	100	10	288	712	-
Optimistic	100	10	333	677	-
Multiversion	100	10	666	334	-

Table 1
Average number of transaction for different methods of concurrency control

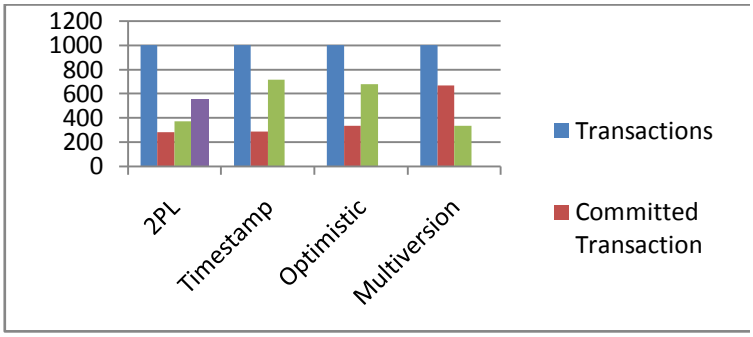

Figure 1 Comparison of all Techniques

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 3, March 2015

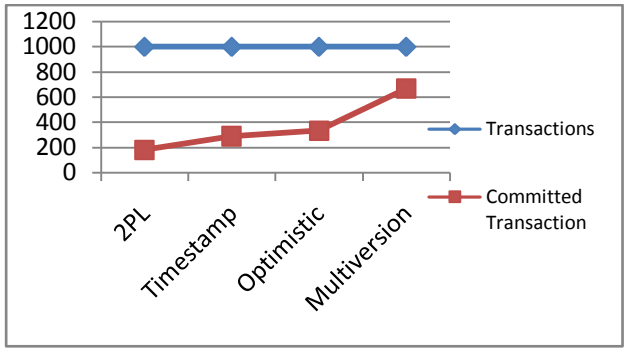

Figure 2 Average number of Commit transactions for different concurrency control methods

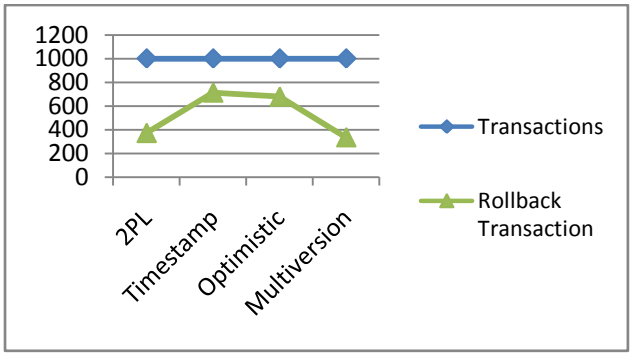

Figure.3.Average number of Rollback transactions for different concurrency control methods

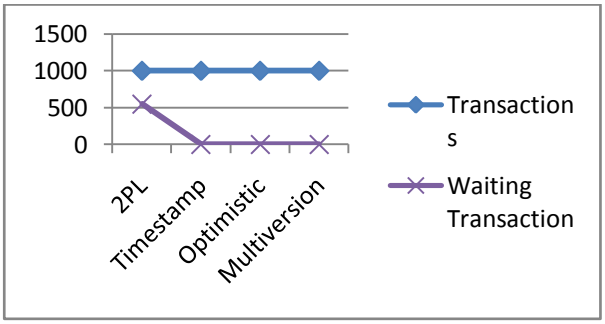

Figure 4.Average number of Wait transactions for different concurrency control methods

5. CONCLUSION

Locking Protocols follow serializability irrespective of type of transactions; read or update-intensive, which could run concurrently. They are good for update-intensive applications but there are locking overhead and they are not free from deadlocks. Also, unnecessary locking for read transactions takes place.

In Time stamp protocols transactions are conflict-free, it gives better concurrency over phased locking because transactions do not block each other needlessly but suffers with large amount of rollbacks. If a transaction is aborted, it is restarted with a new timestamp. This can result in a cyclic restart where a transaction can repeatedly restart and abort without ever completing. Cascading rollback is also degrading concurrency. Another disadvantage is that it has storage overhead for maintaining timestamps as two timestamps must be kept for every data object.

In Optimistic protocol, commit is done only after validation phase because if conflicts occurs between transactions and if not prevented in frequent-update systems it may abort more transactions than either previous method because checks timestamps later.

In some cases we need either to have additional information about the transactions or to impose some structure or ordering on the set of data items in the database. In the absence of such information, two-phase locking is necessary for conflict serializability.

Multiversion follows the approach for maintaining a number of versions of a data item and allocates the right version to a read operation of a transaction. Thus unlike other techniques a read operation in this mechanism is never rejected. Read is normally rejected because the value it was supposed to read is already overwritten. Here reading old copies of each data item can avoid rejections. Read can be given an old value of a data item, even though read is always possible. As Multiversion follows serializability in one hand it is also possible to read all versions that are all updated values therefore multiversion is best among all schemes.

REFERENCES

1. Henry F. Korth, Abraham Silberchatz, S. Sudarshan : Concurrency Control: Database system Concepts (Forth Edition), Page : 591 -617
2. Bharat Bhargava : Concurrency Control in Database Systems : IEEE Transactions on Knowledge and Data Engineering, Vol. 11, NO. 1, January/ February 1999
3. Data Concurrency and Consistency Oracle®DatabaseConcepts 10g Release 2 (10.2)
4. NASER S. BARGHOUTI AND GAIL E. KAISER : Concurrency Control in Advanced Database Applications, ACM Computing Surveys, Vol 23, No 3, September 1991
5. H.T. Kung and John T. Robinson : On Optimistic Methods for Concurrency Control, ACM Transactions on Database Systems, Vol. 6, No. 2, June 1981, Pages 213-226.
6. Patricia Geschwent : A Survey of Traditional and Practical Concurrency Control in Relational DatabaseManagementSystems,TECHNICALREPORT:MU-SEAS-CSA-1994-006,Miami University
7. ALEXANDER THOMASIAN : Concurrency Control : Methods, Performance, and Analysis ACM Computing Surveys, Vol. 30, No. 1, March 1998
8. Ramez Elmasri and Shamkant B. Navathe : Concurrency control techniques, Fundamental of database system ,page 575-596
9. MOHAN, DONALD FUSSELL, ZVI M. KEDEM,AND ABRAHAM SILBERSCHATZ : Lock Conversion in Non-Two-Phase Locking Protocols, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-11, NO. 1, JANUARY 1985

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 3, March 2015

10. Thanasis Hadzilacos' and Christos H. Papadimitriou : CONTROL ALGORITHMIC ASPECTS OF MULTIVERSION CONCURRENCY, ACM Transactions on Database Systems, Vol. 9, No. 1, March 1984, Pages 89-99
11. Joe Hellerstein : Concurrency Control, Locking, Optimistic, Degrees of Consistency Advanced Topics in Computer Systems ,Spring 2008 UC Berkeley
12. Paul Krzyzanowski : Lectures on distributed systems Concurrency Control, Rutgers University – CS 417: Distributed Systems V3.3 ©1999-2009
13. MOHAN, DONALD FUS SELL, ZVI M. KEDEM AND ABRAHAM SILBERSCHATZ : Lock Conversion in Non-Two-Phase Locking Protocols , IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-11, NO. 1, JANUARY 1985
14. CHRISTOS H. PAPADIMITRIOU : A Theorem in Database Concurrency Control, Journal of the .Association for Computing Machinery, Vol. 29, No. 4, October 1982, Page 998-1006
15. MEICHUN HSU and ARVOIA CHAN : Partitioned Two-Phase Locking,ACM Transactions on Database Systems, Vol. 11, No. 4, December 1966, Pages 431-446.
16. PARTHA DASGUPTA , ZVI M. KEDEM : The Five Color Concurrency Control Protocol: Non-Two-Phase Locking in General
17. Databases, ACM Transactions on Database Systems, Vol. 15, No. 2, June 1990, Pages 281-307
18. Pei-Jyun Leu,Bharat Bhargava: MULTIDIMENSIONAL TIMESTAMP PROTOCOLS FOR CONCURRENCY CONTROL l,CSD-TR-521,revised Oct. 1986
19. BERNSTEIN, P. A., HADZILACOS, V., AND GOODMAN : Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading, Mass., 1987
20. 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.
21. HENRY F. KORTH : Locking Primitives in a Database System, Journal of the Association for Computing Machinery, Vol 30, No 1, January 1983, pp 55-79
22. 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.
23. 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