IJARCCE

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

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 1, January 2016

Study and Comparative Analysis of Basic Pessimistic and Optimistic Concurrency Control Methods for Database Management System

Jaypalsinh A. Gohil[1], Dr. Prashant M. Dolia[2]

Research Scholar, Dept of Computer Science, MK Bhavnagar University, Bhavnagar, India[1]

Associate Professor& Research Guide, Dept of Computer Science, MK Bhavnagar University, Bhavnagar, India[2]

Abstract: The concurrency in database system is a common phenomenon in multiuser environment where more than one transactions concurrently accessing the common data. Concurrency can lead to an adverse effect on database if it is not efficiently controlled. In the past many people have addressed the problem of concurrency and also suggested and proposed various concurrency control methods. The scope of this study involves study and comparative analysis of basic concurrency control methods that can be classified as either pessimistic or optimistic. Different protocols exhibit good performance on different situations, some methods prefers locking approach while others are based on time stamp. In this work we have studied basic concurrency control methods of both pessimistic and optimistic approaches, highlighting their pros and cons. At last study shows the comparative analysis of basic concurrency control methods.

Keywords: CC, PCC, OCC, 2PL, S2PL, TIMESTAMP, TO, BOCC.

I. INTRODUCTION

In the past many researchers have made valuable contributions in development of efficient concurrency control algorithm based locking approach, but in recent years there is a need of efficient concurrency control algorithm which is suitable for fast and high performance database systems [1,2]. The classic Kung & Robinson time-stamp based concurrency control algorithm proposed initially [1]. The algorithm is based primarily on two innovative techniques: query killing notes and weak serializability of transactions. In particular, it prefers long transactions over short queries and thus reduces considerably the number of transaction rollbacks required.

Traditional concurrency control algorithms can be broadly classified as either pessimistic or optimistic. Pessimistic Concurrency Control (PCC) algorithms [4, 5] avoid any concurrent execution of transactions as soon as potential conflicts between these transact ions are detected. Alternately, Optimistic Concurrency Control (OCC) algorithms [1] allow such transactions to proceed at the risk of having to restart them in case these suspected conflicts materialize.

The main aim of concurrency control method is to preserve the consistency of database without any overhead. This can be achieved through serializabillity and serial execution of transactions. An execution is serializable if it is computationally equivalent to a serial execution. A serial execution of two or more transactions means that all operations of one transaction are executed before any operation from another transaction can execute. Since serial executions preserve consistency by definition and every serializable execution is equivalent to a serial one, every serializable execution also preserves consistency [6]. The optimistic concurrency control method differs since; detection of conflicts and their resolution are deferred until committed. The underlying assumption here is that such conflicts are rare.

II. CONCURRENCY AND CONCURRENCY CONTROL

A. Concurrency

Concurrency is conflicting situation where more than one user or transaction tires to access the same database resource at the same time. In such an environment each user must be given the equal priority to perform their operation. We must avoid the situation in which one user is updating an object in the database, while another user is reading it [7].

Concurrency control is the problem of synchronizing concurrent transactions such that the following two properties are achieved:

- The consistency of the transaction and database is maintained.
- The maximum degree of concurrency of operations is achieved.

Obviously, the serial execution of a set of transaction achieves consistency, if each single transaction is consistent.

B. Concurrency Control

The efficient concurrency control mechanism should ensure the consistency of the database when transactions are executed concurrently. Concurrency Control is an integral part of database system.

(i) Conflict detection:

Detecting the conflict: We can detect the conflict between more than one transactions in the following two ways. In pessimistic method the conflicts are detected before the access of the data object. [8]. In pessimistic method whenever a transaction tries to access some data item, the concurrency control manager (CC Manager) determines the request and will decide whether to grant the permission or not. In contract the optimistic method identifies the conflicting transactions after it accessed the conflicting data items, when transactions are operating concurrently [1].

However, two or more transactions can conflict in a variety of ways: they can require common resources that must be allocated exclusively, or they can access common data items in incompatible modes. In such a case it will generally be necessary to have some transactions wait, or backup, or restart certain transactions, until the transactions they conflict with have run to completion. If the probability of 'conflict is high, then only a few transactions can run concurrently so that all run to completion. In such a case a limit to increased transaction rates will soon be encountered, and this limit is determined by the nature of the transactions [9].

(ii) Conflicting operations:
Two operations Oi(x) and Oj(x) of transactions Ti and Tj are in conflict if and only if at least one of the operations is a write, i.e.,

- – Oi = read(x) and Oj = write(x).
- – Oi = write(x) and Oj = read(x)
- – Oi = write(x) and Oj = write(x)

The following table shows compatibility matrix of conflicting transactions Ti and Tj.

TABLE I COMPATIBILITY MATRIX FOR TRANSACTIONS TI AND TJ

Compatibility Matrix for Ti and Tj		**Ti**	
		read(x)	**write(x)**
Tj	**read(x)**	□	□
	write(x)	□	□

Generally, a conflict between two operations indicates that their order of execution is important. Read operations do not conflict with each other, hence the ordering of read operations does not matter.

Consider the following two transactions:

T1	T2
Read(x)	Read(x)
x=x+1	x=x+1
Write(x)	Write(x)
Commit	Commit

If conflicts are detected then it can make the adverse effect on the database and leave the database in inconsistent state. To preserve transaction and database consistency, it is important that the read(x) of one transaction is not between read(x) and write(x) of the other transaction.

(iii) Deadlock:
The conflicts can result in dead lock and can be detected through wait-for graph. A set of transactions is in a deadlock situation if several transactions wait for each other. A deadlock requires an outside intervention to take place. Any locking-based concurrency control algorithm may result in a deadlock, since there is mutual exclusive access to data items and transactions may wait for a lock. Some pessimistic algorithms that require the waiting of transactions may also cause deadlocks [9]. Deadlocks are not desirable in concurrent transaction execution environment. The main idea behind developing optimistic concurrency control method is to remove the overhead of locking. By the name itself the method takes into account the assumption that conflicts between the transactions are rare events and very unlikely to happen frequently. They are optimistic in a way by assuming that conflicts will not occur between the concurrent transactions. Generally, in optimistic method locks are not used so it is lock free, which is one of the major disadvantages of pessimistic concurrency control method. In optimistic scheme the concurrency control is postponed the transaction reaches its finish point after that the potential conflicts has to take place and will be resolved. Hereby the conflict resolution mechanism takes into the account the progress done by the transaction and the nature of conflict for a transaction.

(iv)Conflict resolution between concurrent conflicting transactions:
A conflict resolution mechanism is activated by concurrency control manager when there is a conflict between the concurrent transactions tires to access the same data item or object. After identifying the conflicting transactions the concurrency control manager decides a victim from a set of conflicting transactions to penalize it through appropriate action. The following two actions can be imposed on the conflicting transaction which is identified as an victim. (i) Blocking (Waiting) (ii) Abort (Restart after termination). Both the actions can be taken while ensuring the concurrency through pessimistic method i.e. blocking or aborting of transaction [1]. It is not the case with optimistic concurrency control method because here aborting is suitable option since conflict has been detected after the data object is being accessed by a transaction and based on some performance computation. [8]. If we take the timing of both the actions blocking is done immediately after the conflict is being detected but aborting a transaction is either immediate or delayed.

III. BASIC CONCURRENCY CONTROL METHODS

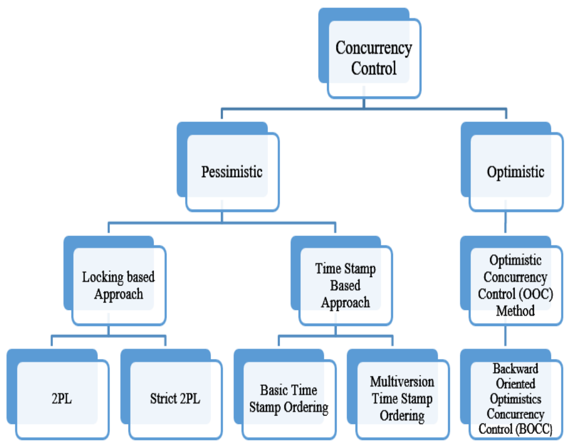

Concurrency Control
Pessimistic
Optimistic
Locking based Approach
Time Stamp Based Approach
Optimistic Concurrency Control (OOC) Method
2PL
Strict 2PL
Basic Time Stamp Ordering
Multiversion Time Stamp Ordering
Backward Oriented Optimistics Concurrency Control (BOCC)

Fig1Basic taxonomy of concurrency control methods

As per the assumptions of pessimistic concurrency control method more number of transaction will conflict in concurrent transaction execution environment, so the concurrent execution of transaction is synchronized and decided early in the transaction execution cycle. The following are the most common pessimistic concurrency control schemes.

1. Two-Phase Locking (2PL)
2. Timestamp Ordering (TO)
2.1 Basic Timestamp Ordering
2.2 Multiversion Timestamp Ordering

Optimistic concurrency control method differ from the pessimistic method in a way that here in contrast to pessimistic concurrency control approach we have to assume that very few transactions will conflict in normal operation, so there is no prerequisite sequence, synchronization and execution of transaction until transaction terminates. Basically Backward Oriented Optimistic Concurrency Control (BOCC) is popularly used.

1. Backward Oriented Optimistic Concurrency Control (BOCC)
A. Pessimistic Concurrency Control Methods

(i) Lock-based approach:
Lock based concurrency control protocol works on simple lock mechanism to control the concurrent access to the data item. If lock is acquired by the transaction then and then only permission is given to access the data item.

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 write lock (w) – also called exclusive lock which is denoted by (X) or read lock (r) – also called shared lock which is denoted by (S) [7]. The transaction which performs both read and write from the data item X, exclusive-mode lock is given. The transaction which is only reading the data item, but cannot write on data item, shared-mode lock is given to data item. Transaction can continue its operation only after request is granted [10].

Locking-based concurrency algorithms ensure that data items shared by conflicting operations are accessed in a mutually exclusive way. This is accomplished by associating a “lock” with each such data item.

TABLE II SHARED &EXCLUSIVE LOCK CONDITIONS

	Write Lock - X	**Read Lock -S**	-
Write Lock –X	N	N	Y
Read Lock - S	N	Y	Y
-	Y	Y	Y

- General locking algorithm:

1. Before using a data item x, transaction requests lock for x from the lock manager.
2. If x is already locked and the existing lock is incompatible with the requested lock, the Transaction is delayed and waits for lock to be released.
3. Otherwise, the lock is granted.

Consider the following two transactions:

T1	T2
Read(x)	Read(x)
x = x + 1	x = x ∗ 2
Write(x)	Write(x)
Read(y)	Read(y)
y = y − 1	y = y ∗ 2
Write(y)	Write(y)

The following schedule S which is a valid locking-based schedule (lock(x) indicates the acquisition of lock and unlock(x) indicates the release of a lock on x):

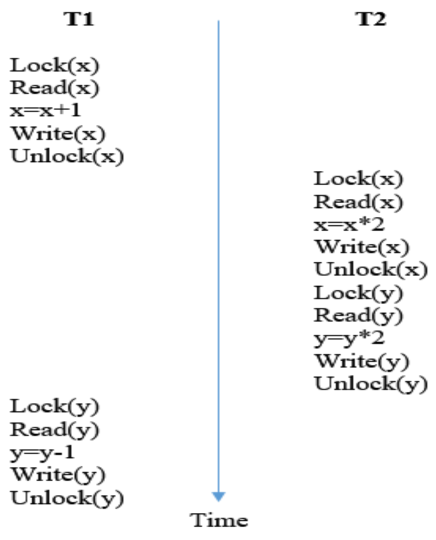


Fig. 2 Schedule S1

However, S1 is not serializable because S cannot be transformed into a serial schedule by using only non-conflicting swaps. Ultimately the result is different from the result of any serial execution.

- Two-phase locking protocol (2PL):
It can be possible for a transaction to always commit itself by not violating the serializability property. If proper care is not taken while acquiring and realizing the locks, it will result in inconsistency and can translated into deadlock. Transaction execution must always be serialized in concurrent execution environment and result of serialization must always be same as if transactions were performed in serial manner to ensure the transaction and database consistency. So it is up most important that the conflicting operations of the multiple transactions are executed in the same order, a restriction is imposed. It ensures that any new transaction is not allowed to aquire a new lock until the old transaction completes its execution and releases the lock. This phenomenon is called Two Phase Locking (2PL) [11].

In two phase locking protocol (2PL) each transaction is executed in two phases namely,
- Growing phase: the transaction obtains the necessary locks
- Shrinking phase: the transaction releases the unwanted locks

IJARCCE

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

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 1, January 2016

The lock point is the moment when transitioning from the growing phase to the shrinking phase.

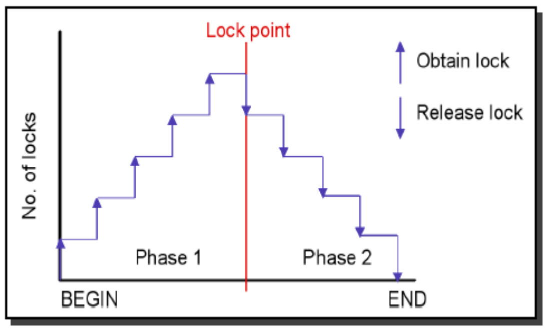


Fig.3Two-phase locking protocol (2PL)

- Properties of the 2PL:
Generates conflict-serializable schedules, but schedules may cause cascading aborts. If a transaction aborts after it releases a lock, it may cause other transactions that have accessed the unlocked data item to abort as well. Sometimes this situation is not desirable which is shown in represented in the below schedule so strict 2PL comes for rescue. The following schedule S2 is not valid in the 2PL protocol:

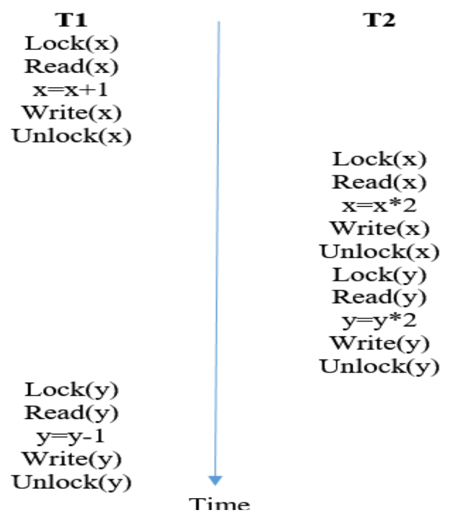


Fig.4Schedule S2 for Two-phase locking protocol (2PL)

- Strict 2PL locking
In strict 2PL the transaction holds the locks till the end of the transaction. So, ultimately cascading aborts are avoided [12].

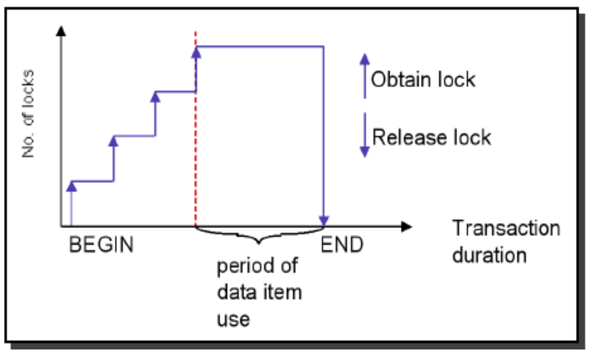


Fig. 5 Strict Two-phase locking

In below schedule after Read(x) transaction T1 cannot request the lock write(y). So following schedule S3 is valid schedule in the strict 2PL protocol.

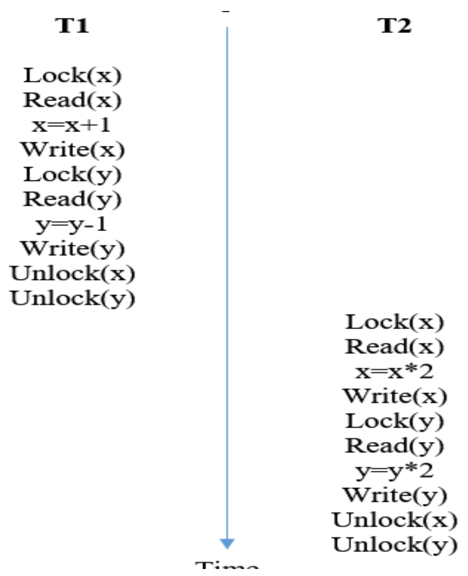


Fig. 6Schedule S3 for strict Two-phase locking protocol

(ii) Timestamp ordering method
- Basic timestamp-ordering protocol
Timestamp ordering eliminates the major bottleneck of deadlock for transactions, as in this environment no transaction is waiting for other. The problem of starvation can surface for long transactions if a sequence of conflicting short transactions causes repeated restarting of the long transaction. In such situation rate of cascading rollbacks are high [12]. This protocol provides edge over locking protocol because transaction do not wait for each other over a long period of time needlessly. It aborts the conflicting transaction instead of putting it in a waiting state.

- Timestamp-ordering rule (TO rules):
Timestamp-ordering based algorithms do not maintain serializability by mutual exclusion, but select (a priori) a serialization order and execute transactions accordingly. Transaction Ti is assigned a globally unique timestamp ts(Ti). Conflicting operations Oi and Oj are resolved by timestamp order, i.e., Oi is executed before Oj if and only if ts(Ti) < ts(Tj) [13]. To allow for the scheduler to check whether operations arrive in correct order, each data item is assigned a write timestamp (wts) and a read timestamp (rts) in the following manner: rts(x): largest timestamp of any read on x. wts(x): largest timestamp of any write on x. Then the scheduler has to perform the following checks:

- Read operation, Ri(x):

∗ If ts(Ti) < wts(x): Ti attempts to read overwritten data; abort Ti.
∗ If ts(Ti) ≥ wts(x): the operation is allowed and rts(x) is updated.

- Write operations,Wi(x):

∗ If ts(Ti) < rts(x): x was needed before by other transaction; abort Ti.

∗ If ts(Ti) < wts(x): Ti writes an obsolete value; abort Ti.
∗ Otherwise, executeWi(x).

Consider the following schedule S4 with three transaction T1, T2 and T3 respectively. The schedule shows their execution order on data items x, y and z.

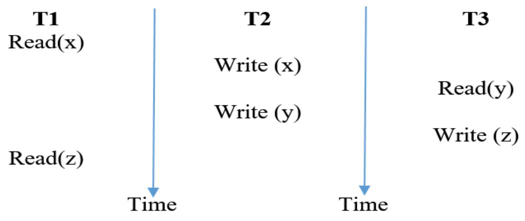


Fig. 7Schedule S4 with three transactions T1, T2 and T3

If we look closely at the execution order and sequence of operations of transactions, the transaction T1 starts its execution earlier to other two transactions and does two read operations Read(x) and Read(z). It is also visible from the sequence of operations represented in a schedule that T1 also performs las read operations Read(z) after all the operations of other two transactions have completed. In the similar manner transaction T2 performs two write operations Write(x) and Write(y) and transaction T2 performs one read and one write operations namely Read(y) and Write(z).

If we go by the execution sequence and order of transaction after applying rules of Basic Time Stamp ordering protocol, then we can see from below represented diagram that T1 and T2 will be aborted and T3 will survive and executed in normal manner. If we go by an execution sequence and rules of protocol, in the execution order first of all transaction T1 performs Read(x) operation then, there is turn of transaction T2 which performs Write(x) operation. In next sequence transaction T3 get the contention of resource and performs Read(y) operation. At this point rules are checked for transaction T2 and it is obvious that it violates the rules and according to the first condition of write rule the transaction T2 is aborted. Then there is a trun of transaction T3 which performs Write(z) operation, here rules are checked and since there is no violation of rules the transaction allowed to perform Write(z) operation after that transaction T3 commit. At the last stage of sequence transaction T1 performs Read(z) operations and due to incompetency in following the rules transaction T1 is aborted.

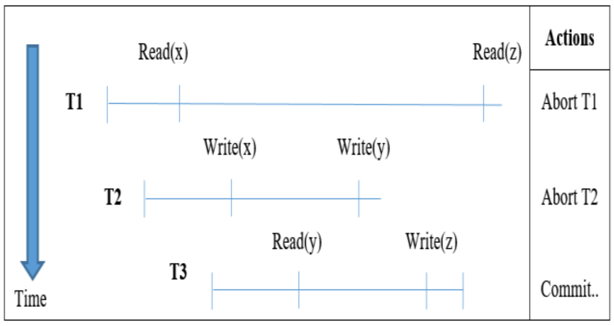


Fig. 8 Execution sequence of transactions T1, T2 and T3 under basic timestamp ordering protocol

So, in this example if we impose the basic time stamp ordering rules and go by the execution sequence then transactions T1 and T2 is aborted and only one transaction T3 successfully commits.

The generation of timestamps (TS) in a distributed environment executed in the following way: TS needs to be locally and globally unique and monotonically increasing. System clock, incremental event counter at each site, or global counter are unsuitable (difficult to maintain). Concatenate local timestamp/counter with a unique site identifier: <local timestamp, site identifier>. The site identifier is in the least significant position in order to distinguish only if the local timestamps are identical [13]. Schedules generated by the basic TO protocol have the following properties:

- Serializable
- Since transactions never wait (but are rejected), the schedules are deadlock-free
- The price to pay for deadlock-free schedules is the potential restart of a transaction several times.

- Multiversion timestamp ordering:
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 [14].
This approach maintains a number of versions of a data item and allocates the right version to a read operation of a transaction. Thus unlike other mechanisms a read operation in this mechanism is never rejected.
To understand the concept assume that x1, x2, …, xn are the versions of a data item x created by a write operation of transactions. With each xi a read_TS (read timestamp) and a write_TS (write timestamp) are associated. The operation read_TS(xi) reads timestamp of xi is the largest of all the timestamps of transactions that have successfully read version xi. On the other hand operation write_TS(xi) writes timestamp of xi that wrote the value of version xi. A new version of xi is created only by a write operation.

- Rules:

The following two rules governs the multiversion time stamp ordering protocol [14].
Rule 1: If transaction T issues write_item(x) and version i of x has the highest write_TS(xi) of all versions of x that is also less than or equal to TS(T), and read _TS(xi) > TS(T), then abort and roll-back T; otherwise create a new version xi and read_TS(x) = write_TS(xj) = TS(T).

Rule 2: If transaction T issues read_item (x), find the version i of x that has the highest write_TS(xi) of all versions of x that is also less than or equal to TS(T), then return the value of xi to T, and set the value of read _TS(xi) to the largest of TS(T) and the current

read_TS(xi). Rule 2 guarantees that a read will never be rejected.
The following steps are performed while checking the rules of protocol.
1. x is the committed version of a data item.
2. T creates a second version x' after obtaining a write lock on x.
3. Other transactions continue to read x.
4. T is ready to commit so it obtains a certify lock on x'.
5. The committed version x becomes x'.
6. T releases it's certify lock on x', which is X now.

Consider the following example where three transaction T1….T3 running concurrently. The examples represents the sequence and order of execution of these four transactions and value of A. If we apply the rules of multiversion timestamp ordering protocol T3 does not have to abort, because it can read an earlier version of A as visible from the below sequence of schedule.

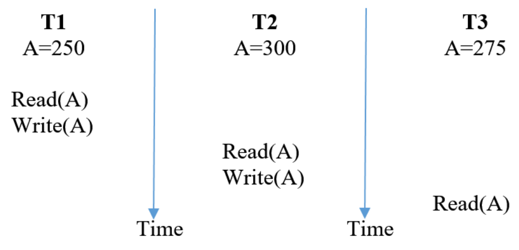


Fig. 9 Schedule S5 with three transactions T1, T2 and T3.

If we go by the execution sequence and order of transaction after applying rules of Multiversion Time Stamp ordering protocol, then we can see from below represented diagram that T1, T2 and T3 all will survive and committed. If we go by an execution sequence and rules of protocol, in the execution order first of all transaction T1 performs Read(A) operation then, then again transaction T1 performs Write(A) operation. After this sequence a new version for data item A is created. In next sequence transaction T2 get the contention of resource and performs Read(A) and Write(A) operations respectively. After Write(A) operation of transaction T2 again a new version of A is generated.

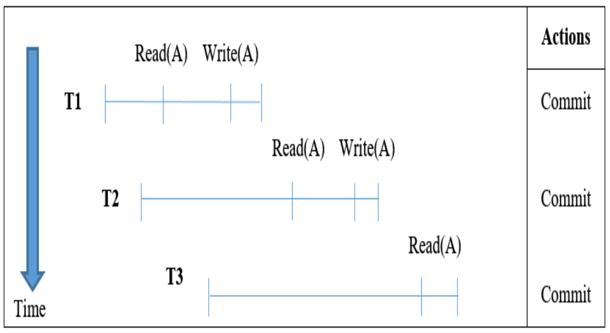


Fig.10 Execution Sequence of transactions T1, T2 and T3 under multiversion timestamp ordering protocol

Then there is a turn of transaction T3 which performs Write(A) operation, here rules are checked and since there is no violation of rules the transaction allowed to perform Write(A) operation because it can read the earlier various of A after that transaction T3 commit. So, in this example if we impose the multiversion time stamp ordering rules and go by the execution sequence then transactions all three transactions T1, T2 and T3 successfully completes its execution and commits successfully.

One of the major drawback of Multiversion Timestamp Ordering protocol is that significantly more storage is required to maintain multiple versions. To check unlimited growth of versions, a garbage collection is run when some criteria is satisfied, which results in extra processing overhead.

(iii) The problems with pessimistic approach:
In pessimistic locking method data to be updated is locked in advance. Once the data to be updated has been locked, the application can make the required changes, and then commit or rollback - during which the lock is automatically dropped. If anyone else attempts to acquire a lock of the same data during this process, they will be forced to wait until the first transaction has completed [15]. This approach is called pessimistic because it assumes that another transaction might change the data between the read and the update. In order to prevent that change from happening and the data inconsistency that would result the read statement locks the data to prevent any other transaction from changing it. This can lead to the following problems:

- The Lockout:
A transaction invoked by an application user selects a record for update, and executing the operations without finishing or aborting the transaction. All other users with their respectivetransaction that need to update that record are forced to wait until the user completes its transaction. So, on an average for other transactions consumes more time in waiting for other transaction to complete its execution rather than executing the operations of itself, which is undesirable for real-time time critical system.

- The Deadlock:
Transaction A and B are both updating the database at the same time. Transaction A locks a record and then attempt to acquire a lock held by transaction B who is waiting to obtain a lock held by transaction A. Both transactions go into an infinite wait state the so-called deadly embrace or deadlock [10].

B. Optimistic Concurrency Control Methods

(i) Overview of optimistic approach:
The main disadvantage of pessimistic approach is locking. Locks have an overhead associated with maintaining and checking them. They may be a situation in which deadlock can arise in a system. As an solution of pessimistic concurrency control approach an alternative proposed by Kung and Robinson in 1981 is optimistic concurrency control [9]. This approach let the transaction to execute itself without worry of conflict with other transactions. As the name implies the optimistic concurrency control algorithms are based on the assumption that conflicts between transactions are not frequent and regular. The

probability of a lock conflict is about 0.1 and the probability of a deadlock is much lower (< 0.001) as per study [16]. So there is no need of locking mechanism which result in operation overhead. When a conflict does arise, then the system will have to deal with it.

Optimistic concurrency control requires transactions to operate in a private workspace, so their modifications are not visible to other until they commit. When a transaction is ready to commit, a validation is performed on all the data items to see whether the data conflicts with operations of other transactions. If the validation fails, then the transaction will have to be aborted and restarted later. Optimistic control is clearly overcomes the problem of deadlock.

(ii) Benefits of optimistic approach:
This optimistic concurrency control provides the following benefits over its counterpart pessimistic approach. It is deadlock free and avoids any time consuming node-locked scenarios. This approach is generic in the sense if the transactions become query dominant, the concurrency control overhead becomes almost negligible. In this approach reading operations are completely unrestricted whereas write operations of transactions are severely restricted.

(iii) The three phases:
The optimistic concurrency control algorithm follows the process in three different phases: READ phase, a VALIDATION phase and optional WRITE phase which is executed if the transaction is allowed to commit [17], the process is highlighted in the below diagram.
If we generalize the three phases of optimistic approach in the READ all write operations take place on local copies of the object to be modified. During VALIDATION phase it is determined that the transaction will not cause a loss of integrity. Finally, in WRITE phase copies are made global.

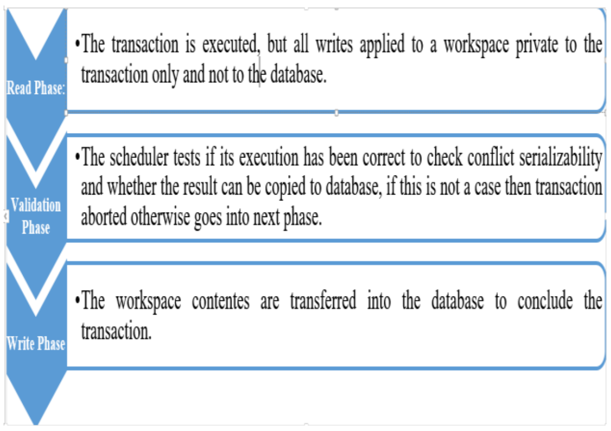

Read Phase:
•The transaction is executed, but all writes applied to a workspace private to the transaction only and not to the database.
Validation Phase
•The scheduler tests if its execution has been correct to check conflict serializability and whether the result can be copied to database, if this is not a case then transaction aborted otherwise goes into next phase.
Write Phase
•The workspace contentes are transferred into the database to conclude the transaction.

Fig.11. Three phases of optimistic concurrency control

- Read and Write phases:
Read is also considered as a working phase. Each transaction has a tentative version of each of the object that it updates READ operations are performed immediately WRITE operations record the new values of the objects as tentative values. Two records are kept of the objects accessed within a transaction: a read set and a write set. If validation succeeds, then the transaction enters the WRITE phase. After WRITE phase, all written values become global. When a transaction completes, it will request its validation and write phases via TtransactionEnd call.

- Validation phase:
As the name suggest hereby strong validation checks are performed on transactions which uses a particularly strong form of validation. This is especially important with long-running transactions method uses an overqualified update scheme to test whether the underlying data source has been updated by another transaction since the beginning of the current transaction. Kung and Robinson employ Serial Equivalence for verifying the correctness of concurrent execution of transactions.

- Validation of serial equivalence:
During VALIDATION phase each transaction explicitly assigns Transaction Number, T(i), at the end of the READ phase transaction numbers are assigned in order. If the transaction is validated and completes successfully, it retains this number but if it fails the validation checks and is aborted, or if the transaction is read-only, the number is released for reassignment. Transaction numbers are integers assigned in ascending sequence. The number of a transaction defines its position in time Tid satisfies the following property: T(i)<T(j) . Operations conform to the following validation conditions:

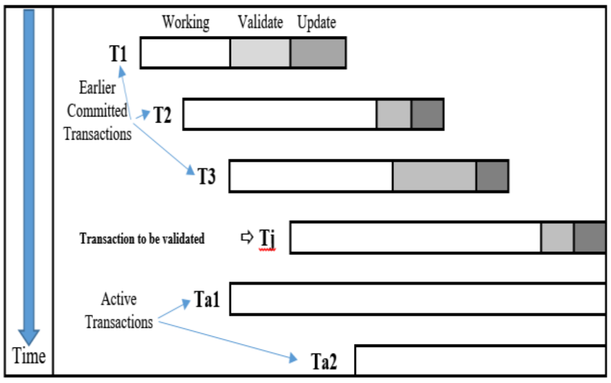
- Ti must not read objects being written by Tj
- Tj must not read objects being written by Ti
- Ti must not write objects being written by Tj and Tj must not write objects being written by Ti

(iv) Backward – Oriented Optimistic Concurrency Control (BOCC) method:
The two schemes Backward – Oriented and Forward – Oriented Optimistic Concurrency Control schemes were proposed by Harder [18]. In BOCC the read set of a validating transaction is compared to the write sets of all transactions that have finished the read phase before the validating transaction. Under backward-oriented optimistic concurrency control (BOCC), a transaction under validationexecutes a conflict test against all those transactions that are already committed [19, 20].

- BOCC validation condition:
Compare Tj to all previously committed Ti. Accept Tj if one of the following holds:

- Ti has ended before Tj has started, or
- $RS(Tj) \cap WS(Ti) = \varnothing$ and Ti has validated before Tj.

Consider the following example which shows the sequence of execution of earlier committed transactions and currently active transaction. It tests the condition of BOCC protocol.
The earlier committed transactions are T1, T2 and T3. T1 committed before Tj started. The transactions T2 and T3 committed before Tj finished its working phase. Validation consists of comparing the READ set of Tj with the write set of T2 and T3.



Fig, 12 Transaction execution sequence under BOCC method

Conflicts are resolved by aborting the transaction undergoing validation. If transaction being validated does not have any read operations, it does not have to be checked. Optimistic concurrency requires that the WRITE sets of old committed versions of objects corresponding to recently committed transactions are retained until there are no unvalidated overlapping transactions which might conflict. When a transaction is successfully validated, its transaction number and write set are recorded in a list that is maintained by the transaction service.

IV. COMPARATIVE ANALYSIS OF BASIC CONCURRENCY CONTROL METHODS

The following table shows the comparative study and performance evaluation of various pessimistic and optimistic concurrency control methods.

TABLE III COMPARATIVE ANALYSIS OF BASIC CONCURRENCY CONTROL METHODS

Sr. No.	Method	Eliminates Blocking (Y/N)	Overcomes Deadlock (Y/N)	Provides Consistency (Y/N)	Reduced Overhead (Y/N)	Flexible	Secure	Efficient
1	2PL	N	N	Y	N	Y	N	N
2	S2PL	N	N	Y	N	N	Y	N
3	BTO	Y	Y	Y	N	N	N	Y
4	Multi Version TO	Y	Y	N	N	N	Y	Y
5	BOCC	Y	Y	Y	Y	N	Y	Y

V. SUMMARY

Pessimistic locking based approach is suitable for update-intensive applications while optimistic methods are more suitable for read operation. As shown in the table Table no unnecessary overheads of locking of read-only transactions and will give good performance. The level of performance is degraded with standard locking techniques, if transactions are not compatible with each other, whereas transaction restarts to resolve deadlocks have secondary effect on performance [21]. Timestamp based protocol somewhat overcomes the situation of blocking. They are based on older-younger relationship. Timestamp can give better results if some available information about the transactions or the database can be used for increasing concurrency [22].

In a locking approach, transactions have to wait at certain points, while in an optimistic approach backing them up controls the transactions. In this approach 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 [23].

VI. CONCLUSION

In this study we have highlighted some of the basic concurrency control methods which follows both optimistic and pessimistic concurrency control techniques. In most commercial systems, the most popular mechanism for concurrency control is two-phase locking (2PL) and strict two –phase locking (S2PL). The protocols based on locking mechanism follows serializability without considering the type of transaction. The lock based methods are very efficient for update intensive application, but it has the overhead of locking and can find themselves in a deadlock. In timestamp ordering protocols transactions do not conflict, it is better than phased locking because transaction never block each other needlessly, but this method suffers from large amount of transaction rollback. Cascading rollbacks are the frequent in this environment. The basic timestamp methods we have highlighted in the paper are Basic Timestamp Ordering (BTO) and Multiversion Timestamp Ordering (MTO). Keeping the updated records of two timestamps for every data object is an overhead. The optimistic approach works on three different phases: Read, Write and Validate. In optimistic approach the transaction is committed only after the finish of validation phase. The optimistic methods which is covered in this paper is Backward Oriented Concurrency Control (BOCC) method. The conflicts can occur between two concurrent running transactions and if they are not prevented it may abort more transactions then previous methods because checks are made at the later stage. The optimistic protocol is best suited for read intensive applications. If we compare the performance of all basic methods of concurrency control the optimumperformance will be provided by optimistic concurrency control methods.

REFERENCES

[1] Kung H. T.. and Robinson J. T., "On Optimistic Methods for Concurrency Control". ACM Trans. on Database Systems, V. 6. No. 2,1981.

[2] Carey M. J.,"Improving the Performance of an Optimistic Concurrency Control Algorithm Through Timestamps and Versions", IEEE Trans. On Software Engineering. V. 13, No. 6,1987.

[3] Rishe N.. Tal D.. and Gudes E., "An optimistic Concurrency control algorithms for distributed-storage semantic database machines". Submitted for publication, 1989.

[4] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger. \The notions of consistency andpredicate locks in a database system." Communications of the ACM, 19(11):624{633,November 1976.

[5] J. N. Gray, R. A. Lorie, G. R. Putzolu, and I. L. Traiger. \Granularity of locks and degrees of consistensy in a shared data base." In G. M. Nijssen, editor, Modeling in Data Base

Management Systems, pages 365{395. North-Holland, Amsterdam, The Netherlands, 1976.
[6] P.A. Bernstein and N. Goodman, "Concurrency Control in Distributed Database Systems," ACM Computing Surveys, Vol. 13(2), June 1981, pp. 186 - 221.
[7] Bernstein, P. and N. Goodman, 1981. Concurrency control in distributed databasesystems. Comp. Sur., 13: 185-221.
[8] Franaszek, P. and J. Robinson, 1985. Limitation of concurrency in transaction processing. ACM Trans. Database Syst., 10: 1-28.
[9] Amer Abu Ali, 2006, On Optimistic Concurrency Control for Real-Time Database Systems, American Journal of Applied Sciences 3 (2): 1706-1710, 2006
[10] Joe Hellerstein : Concurrency Control, Locking, Optimistic, Degrees of Consistency Advanced Topics in Computer Systems ,Spring 2008 UC Berkeley
[11] 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
[12] Databases, ACM Transactions on Database Systems, Vol. 15, No. 2, June 1990, Pages 281-307
[13] Pei-Jyun Leu,Bharat Bhargava: MULTIDIMENSIONAL TIMESTAMP PROTOCOLS FOR CONCURRENCY CONTROL l,CSD-TR-521,revised Oct. 1986
[14] 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.
[15] J.A.Gohil, Dr.P.M.Dolia, "Comparative Study and Performance Analysis of Optimistic and Pessimistic Approaches for Concurrency Control Suitable for Temporal Database Environment", National Conference on Emerging Trends in Information & Communication Technology (NCETICT), 2013
[16] J.N. Gray, P. Homan, H. Korth and R. Obermack, 'A Straw Man Analysis of the Probability of Waiting and Deadlock in a Database System,' Proc. 5th Berkeley Workshop on Distributed Data Processing System, Febr. 1981.
[17] M.J. Carey and M.R. Stonebreaker, "The Performance of Concurrency Control Algorithms for Database Management Systems, Proc. 10th Intl. Conf. on Very Large Data Bases, Singapore, Aug. 1984, pp. 107 – 118.
[18] Härder, T. :Observations on Optimistic Concurrency Control Schemes. Information Systems. 9,111-120(1984)
[19] Lee, J.:Precise Serialization for Optimistic Concurrency Control, Data & Knowledge Engineering. 29, 163-178 (1999)
[20] Lee, J., Son, S.H., Using Dynamic Adjustment of Serialization Order for Real-Time Database Systems. In: proceedings of the Real-Time Systems Symposium, pp. 66-75(1993)
[21] Kamal Solaiman, Matthew Brook, Gary Ushaw, Graham Morgan. A Read-Write-Validate A Approach to Optimistic Concurrency Control for Energy Efficiency of Resource-Constrained Systems
[22] ALEXANDER THOMASIAN : Concurrency Control : Methods, Performance, and Analysis ACM Computing Surveys, Vol. 30, No. 1, March 1998
[23] Joe Hellerstein : Concurrency Control, Locking, Optimistic, Degrees of Consistency Advanced Topics in Computer Systems ,Spring 2008 UC Berkeley.