International Journal of Electrical Electronics & Computer Science Engineering
Special Issue - AET 2017 | E-ISSN : 2348-2273 | P-ISSN : 2454-1222
Available Online at www.ijeecse.com

Issues with Concurrency Control Techniques

Sonal Kanungo[1], Dr. Rustom D. Morena[2]
[1]Smt. Z. S. Patel College of Computer Application, Surat, [2]Veer Narmad South Gujarat University,Surat
sonalkanungo@gmail.com, rdmorena@rediffmail.com

Abstract: When the shared data of database is accessed and updated concurrently by a number of transactions; consistency of the stored and retrieved data is an important issue. Each transaction which is executed alone (mutual exclusion) transforms a consistent state into a new consistent state. This consistency can be achieved through isolation that means only one transaction is executing at any time. A system that ensures this property is said to be serializable. Our primary goal here is to analyze the fundamental limitation of concurrency with various scheduling policies independently of any given transaction processing system.

Keywords Concurrency control, Consistency, Serializability, Locking, Deadlock.

I. INTRODUCTION

Database consistency can be preserve only by ensuring that the schedules of executing transactions are serializable. When several transactions execute concurrently in the database, however, the isolation property may no longer be preserved [8].The system must control the interaction among the concurrent transactions and it is achieved through mechanisms called concurrency-control schemes. The concurrency-control schemes are all based on the serializability property. All the schemes presented here ensure that the schedules are serializable. Concurrency control is the technique used to keep up the consistency and isolation properties of exchanges and it is required when two simultaneous transactions attempt to simultaneously perform Read or Write operations on the same objects [17].

II. CONCURRENCY CONTROL TECHNIQUES

The concurrency-control schemes are all based on the serializability property. Serializability is require that data items be accessed in a mutually exclusive manner; that is, while one transaction is accessing a data item, no other transaction can modify that data item. Transactions run in such a way that they appear to be executed one at a time, or serially, rather than concurrently [8]. In this paper, we had discussed Lock-Based Protocols, Timestamp-Based Protocols, Validation and Multiversion Schemes.

A. The Two-Phase Locking Protocol:

The basic Two-Phase Locking protocol is the most common locking protocol in conventional database systems. 2PL transaction execution consists of two phases. In the first phase, locks are acquired but will not be released. In the second phase, locks are released but new locks will not be acquired.

A transaction may be granted a lock on an item if the requested lock is compatible with another lock already held on the item by other transactions [14].Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive lock on the item no other transaction may hold any lock on the item.If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted [8].

The first phase can also be considered as the growing phase, in which a transaction obtains more and more locks without releasing any. Once the transaction unlocks an item it starts its shrinking phase and can no longer upgrade locks or request new locks. A shrinking phase, in which locks are only released. During the shrinking phase, a transaction is prohibited from acquiring locks [14]. If a transaction tries during its growing phase to acquire a lock that has already been acquired by another transaction, it is forced to wait. This situation might result in deadlock [8].

Points of Interest:

1) This protocol guarantees conflict-serializable schedules.

2) Two phase locking additionally separate amongst peruses and composes and their impact on the database.

3) Two phase locking ensures serializability paying little heed to the sorts of transactions which could work simultaneously with a given transaction.

4) This is protected as it functions admirably with update-intensive applications.

Hindrances:

1) It is considered as deadlock bringing on locking protocols. Two-phase locking does not guarantee occurrence from deadlocks.

2) Deadlock exists in most locking protocols. Starvation is additionally conceivable if concurrency control is seriously planned. On the off chance that locking protocol is not without deadlock, deadlock detection must be thought to be a piece of bolt upkeep overhead.

3) A similar transaction is over and again rolled back because of deadlocks [4].

4) Lock support speaks to an overhead. Indeed, even read-only transactions, which can't influence the uprightness of the information. Since the utilization locking the information being perused are not adjusted by different transactions in the meantime [5].

5) Cascading roll-back is conceivable under two-phase locking.

6) This protocol is wasteful for query-intensive applications in view of locking overhead and plausibility of deadlock and sits tight for bolted information [5].

B. Timestamp-Based Protocols:

Timestamp ordering transaction has a unique timestamp which is the starting time of that transaction. When transaction is executed, the timestamp is attached to every read/write request of the transaction. The read timestamp and write timestamp record the timestamps of the last transaction that reads and writes respectively the data. A write request from a write transaction will compares with the timestamp of the write transaction and with the read and write timestamps of the data requested. If the timestamp of the transaction is smaller than the read timestamp, it will immediately restart. If the timestamp of the transaction is larger than the read timestamp, but smaller than the write timestamp, the write request is ignored. A transaction that has a greater timestamp has updated thedata. If the timestamp of the write transaction is larger than both the read and write timestamp of the data, the former timestamp replaces both the latter timestamps and the request is granted after the request is granted the transaction immediately issues the next request [8]. Timestamps, a concurrency control mechanism can totally order requests from transactions according to the transactions timestamps [14].The transaction which is aborted and restarted, it is assigned a new timestamp.

Favorable Circumstances"

1) All updates are built into the database after the transaction submits so falling rollback is maintained a strategic distance from. Improved concurrency over phased locking since transactions don't obstruct each other unnecessarily.

2) Locking blocks, and timestamp aborts as opposed to sits tight for get to.

3) Transaction can read a similar thing at various circumstances, and it is sans conflict.

Detriments:

1) Suppose transaction aborts, yet this transaction has perused a data composed by other transaction, then this other transaction should likewise prematurely end.

2) Any transaction that has perused a data composed by second transaction must prematurely end. This can prompt falling rollback [14].

3) As transaction that aborts; will restart with another timestamp this will prompt starvation.

C. Validation-Based Protocols:

In many applications, locking add an unnecessary overhead to constrain concurrency. For read-only transactions lock maintenance represents an unnecessary overhead, which do not affect the integrity of the database. In all cases, there is noof need locking mechanisms for high concurrency. Deadlock-free locking mechanisms work well only in some cases but perform poorly in other cases. The large parts of the database reside on secondary storage, locking of objects that are accessed frequently while waiting for secondary memory access causes a significant decrease in concurrency [8]. Locks are not permitting to be released except at the end of the transaction, which leads to cascade daborts, decreases concurrency [16].

Use of locking is not necessary to guarantee consistency most of the time because most transactions do not overlap; locking may be necessary only in the worst cases. To avoid these disadvantages, Kungand Robinson [1981] presented the concept tof "optimistic" concurrency control [1].These transactions consist of three phases: a read phase, a validation phase, write phase. In the read phase, all write stake place on local copies as transient versions of the records to be written. In Validation phase the changes the transaction had made will not violate serializability with respect to all committed transactions, the local copies are made global. Validation is done by assigning each transaction a timestamp at the end of the read phase and synchronizing using timestamp ordering, only then, in the write phase, these copies become accessible to other transactions [1]. This validation scheme is called the optimistic concurrency control scheme since transactions execute optimistically, assuming they will be able to finish execution and validate at the end.

Favorable Circumstances:

1) If likelihood of conflicts is low this protocol is helpful and gives more prominent level of concurrency.

2) The serializability request is not pre-chosen in this manner generally less transactions should be rolled back [1].

Hindrances:

1) Optimistic concurrency control diminishes concurrency when the read and update sets of simultaneous transactions cover. The utilization of rollback which prompts re-trying of work as the fundamental component for keeping up consistency is a genuine drawback [1].

2) However, there is a probability of starvation of long transactions, because of an arrangement of conflicting short transactions that cause rehashed restarts of the long transaction.

3) Not proficient in high successive update frameworks.

4) May prematurely end a bigger number of transactions than either past strategy since checks timestamps later [14].

D. Multiversion Schemes:

Multiversion protocol creates a new version with every Commit; it never overwrites old values. These old values or versions are always available to tardy Reads [6]. A Read normally rejects because the value it was supposed to read has already been overwritten. Multiple versions item helps the scheduler to avoid rejecting operations that arrive too late [10]. This rejection of Read can be avoided by keeping old versions; a tardy Read can be given an old value of a data item, even though it was "overwritten" [3]. In multiversion concurrency control schemes, each write on data operation creates a new version of data. When a transaction issues a reading of data, the concurrency control manager selects one of the versions of data to be read [1]. Multiversion protocols maintain one or more old versions of objects in the database in order to allow work to proceed using both the current version and older versions [11].

The scheduler has to decide two things for each Read; when to send the Read and which of the versions of data item is to be Read. The existence of multiple versions is only visible to the scheduler, not to user transactions [3]. Multiversion scheme can be differentiate in to Multiversion Timestamp Ordering and Multiversion Two-Phase Locking.

1) Multiversion Timestamp Ordering:

Multiversion concurrency control algorithm produces a new copy or version of data item with each Write on data item. A list of versions of data item with the history of values is kept. A multiversion (MVTO) scheduler processes operations first-come first-served. For *each version*, the value of version and the two timestamps are kept the read timestamp of is the largest of all the timestamps of transactions that have successfully read version and the write timestamp of is the timestamp of the transaction that wrote the value of version. Whenever a transaction is allowed to execute a write operation, a new version of item is created, with both the write and the read set to transaction. A read is always successful, since it finds the appropriate version to read based on the write of the various existing versions of item [5].

A transaction may be aborted and rolled back when transaction is attempting to write a version of item that should have been read by another transaction whose timestamp is read; however, transaction has already read version of item, which was written by the transaction with timestamp equal to write. if transaction is rolled back, cascading rollback may occur.

Favorable Circumstances:

1) A read request never flops as it generally discovered proper form.

2) Deadlocks won't happen in view of rollbacks.

Hindrances"

1)The perusing of a data additionally requires the refreshing of the R-timestamp field, bringing about two potential plate gets to, as opposed to one.

2)As conflicts between transactions are settled through rollbacks, instead of through holds up. This might be costly option [5].

3)Multiversion timestamp-requesting plan does not guarantee recoverability and cascadelessness.

4)Uses rollbacks to determine conflicts rather than holds up.

2) Multiversion Two-Phase Locking:

Multiversion two-phase locking protocol is combine the advantages of multiversion concurrency control with the advantages of two-phase locking. This protocol differentiates between read-only transactions and update transactions [5].

Update transactions perform rigorous two-phase locking; that means all locks will be hold up to the end of the transaction Each version of a data item has a single timestamp. The timestamp here is a counter and will incremented during commit processing [8].In Two Version 2PL, a write lock on a data item prevents transactions from obtaining read locks on data item. When a transaction writes into item, it creates a new version item [2]. It sets a lock on data item that prevents other transactions from reading item, or writing a new version of item. However, other transactions are allowed to read the previous version of data item. Therefore, reads on data are not delayed by a concurrent writer of data. A two version 2PL (2 VZPL) scheduler uses three types of locks:

read locks, write locks, and certify locks. The scheduler sets read and write locks at the usual time, when it processes Reads and Writes. When a transaction has terminated, and is about to commit, it converts all of the transaction's write locks into certify locks. [5]

Favorable Circumstances:

1) A read request never fails.
2) Read-only transaction does best with this multiversion.
3) Ensures recoverability and cascadelessness.

Hindrances:

1) Deadlocks can occur because transactions go in the wait state.
2) Inhibits concurrent execution because of locking overhead. May lock an item no other transaction needs.

III. ISSUE WITH CONCURRENCY CONTROL TECHNIQUES

In a database system, several users may read and update information concurrently. If the operations of the various user transactions are not serialized in a correct fashion, several undesirable situations may arise such as creation of inconsistent data or users receiving inconsistent information [13]. The concurrency control problem is to coordinate the concurrent accesses of the database by the various transactions so that the effect is the same as if the transactions ran one-at-a-time.

Two Phase Locking:

Locking protocols are good for update-intensive applications. In a locking approach, transactions are controlled by having them wait at certain points. In a locking Sterilize approach proved by partially ordering the transactions by first access time for each object. The major difficulty in locking approaches is deadlock, which can be solved by using backup and these does not work well with query intensive applications [14].

Timestamp-Ordering:

The timestamp-ordering protocol ensures conflict serializability as conflicting operations are processed in timestamp order. Since no transaction ever waits for each other therefore this protocol ensures freedom from deadlock. There is possibility of starvation of long transactions because the sequence of conflicting short transactions may repeatedly have restarted [12]. When ordering is incorrect; for example, transaction that started later than the current one has accessed the file and committed, in this case the current transaction is too late and has to abort [14].

Optimistic Concurrency Control:

Transactions are required to operate in isolation, so that the rmodifications are not visible to other until they actually commit. At the instance when a transaction is ready to commit, a validation is performed on all the data items to see whether there are any conflicts in data with operations of other transactions [15]. If the validation fails, the transaction will have to be aborted and restarted later. Optimistic control is clearly deadlock free with no locking or waiting on resources and since no process has to wait for a lock [12]. Optimistic concurrency control does not allow to commit your transaction until conflicts are checked. The risk of having to re-do the work that is already done increases [1]. As the Optimistic concurrency control looks for conflicts only at the time of commit, the chances of being in conflict with other users increase [9]. Therefore, more time is required to determine conflict resolution in the optimistic approach which leads to greater overheads [14].

Multiversion Concurrency Control:

The scheduler mostly rejects operations that arrive too late; this can be handled by multiple versions for concurrency control [7]. The scheduler normally rejects a read because the value it was supposed to read has already been overwritten but with multiversion concurrency control, these old values are never over written because it creates a new version with every successful write and are therefore always available to tardy Reads [10].

Multiversion concurrency control increases the cost of maintaining multiple versions in storage space. This storage requirement can be controlled by periodically purge or archive versions. Purging versions must be synchronized with respect to active transactions as certain versions may be needed by active transactions [3]. This purging activity adds another cost of multiversion concurrency control. Still Multiversion works best among all in Update intensive queries[16].

IV. LITERATURE SURVEY

1) CHRISTOSH. PAPADIMITRIOU, PARIS C. KANELLAKIS, NATHAN GOODMAN DESCRIBED VARIOUS CONCURRENCY CONTROL METHODS USING MULTIPLE VERSIONS [2,3,4] Improved Multiversion concurrency control is given with powerful important and adequate conditions for an execution to be l-SR concurrency control and broadened concurrency control hypothesis. They gave a diagram structure, Multiversion serialization graphs (MVSGs), that checks these conditions and connected the hypothesis to three Multiversion concurrency control algorithms. One algorithm utilizes time stamps, one uses locking, and one joins locking with timestamps.

2) PHILIP A. BERNSTEIN and NATHAN GOODMAN [5] This paper had introduced hypothesis for dissecting the accuracy of concurrency control algorithms for Multiversions database. They exhibited some new Multiversion algorithms broke down these new algorithms and a few already distributed ones. This paper gives surveys concurrency control hypothesis for non multiversion databases and extends the hypothesis to Multiversion databases.

V. EXPERIMENTS AND RESULTS

We created sessions and dynamic framework which store random data and generate random transactions with each session. value of data changes with every session. Random transactions having random read and write operations will request to read and write these data. These transactions are predefined Read only (query) and Update transactions.

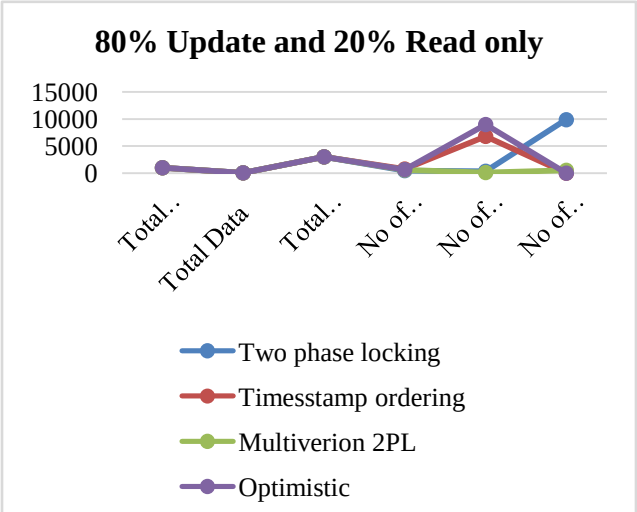
The simulation study of Two phase locking protocol and timestamp protocol shows that Timestamp protocols are good for update applications while Two phase locking protocol gave best results with Read only (query) applications. Two phase locking protocols goes to wait and suffers from deadlock while timestamp protocol generates large number of Rollback, which leads to restart transactions again and again.

Optimistic concurrency will also give good results where conflicts are low. Large number of Rollback will generate as conflicts are checked at commit time. This will generate more number of Roll backas compare to timestamp ordering as validation will take place as later stage which leads to do lots of redo of work.

Multiversion 2phase locking give best results as Read only requests never fail. In timestamp ordering if conflicts found read request will abort and in two phase locking goes to wait. Every successful transaction commit generates new version of data, so when this data is lock then only tardy read can read older version. Read only never has to wait, as Read only never mind reading old or snapshot value.

Table Transactions (20% Readonly, 80% Update)

	Total Transactions	Total Data	Total No of Operations	No of Commit	No of Rollback	No of time Send to Wait
Two phase locking	1000	55	3000	456	356	9876
Timestamp ordering	1000	55	3000	768	6785	0
Multiversion 2PL	1000	55	3000	567	134	546
Optimistic	1000	55	3000	657	8970	0



VI. CONCLUSION

The result of this paper can be seen both in a negative and positive fight. Locking Protocolsare good for update-intensive applications theyinhibit concurrent execution but they have locking overhead and they are not free from deadlocks. In Time stamp protocols transaction, can read the same item at different times, it is conflict-free,it enhanced concurrency as they do notblock each other needlessly. While it suffers from largenumber of rollbacks. Optimistic protocol does not inhibit access prior to validation phase because conflict between transactions check only before commit. Optimistic protocols are good for query intensive transactions. Multiversion concurrency control (MVCC) methods are more robust and perform well for a broad range of workloads as query and Update never block each other.

VII. REFERENCES

[1] 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.

[2] 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.

[3] Henry F. Korth: Locking Primitives in a Database System, Journal of the Association for Computing Machinery, Vol 30, No1, January 1983, pp 55-79.

[4] 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.

[5] Philip A. Bernstein And Nathan Goodman: Multiversion Concurrency Control-Theory and Algorithms ACM Transactions onDatabase Systems, Vol. 8, No. 4, December 1983, Pages 465-483.

[6] 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.

[7] 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.

[8] Henry F. Korth, Abraham Silberchatz, S. Sudarshan:Concurrency Control: Database system Concepts (Forth Edition), Page : 591 -617.

[9] Pei-Jyun Leu,Bharat Bhargava: multidimensional timestamp protocols for concurrency control, CSD-TR-521,revised Oct. 1986.

[10] Michael J. Carey And Waleed A. Muhanna: The Performance of Multiversion Concurrency Control Algorithms Transactions on Computer Systems, Vol. 4, No. 4, November 1986, Pages 338-378.

[11] Bernstein, P. A., Hadzilacos, V., And Goodman:Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading, Mass., 1987.

[12] Naser S. Barghouti And Gail E. Kaiser: Concurrency Control in Advanced Database Applications, ACM Computing Surveys, Vol 23, No 3, September 1991.

[13] Patricia Geschwent: A Survey of Traditional and Practical Concurrency Control in Relational DatabaseManagementSystems,TECHNICALREPORT:MUSEAS-CSA-1994-006,Miami University.

[14] Alexander Thomasian: Concurrency Control : Methods,Performance, and Analysis ACM Computing Surveys, Vol. 30,No. 1, March 1998.

[15] Bharat Bhargava: Concurrency Control in Database Systems:IEEE Transactions on Knowledge and Data Engineering, Vol. 11,NO. 1, January/ February 1999.

[16] Sonal Kanungo, Dr. Rustom 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.

[17] Sonal Kanungo, Dr. Rustom D Morena: Comparison of Concurrency Control andDeadlock Handing in Different OODBMS, International Journal of Engineering Research & Technology (IJERT), Vol. 5 Issue 05, May-2016.