International Journal of Technical Innovation in Modern Engineering & Science (IJTIMES)
Impact Factor: 3.45 (SJIF-2015), e-ISSN: 2455-2585
Volume 4, Issue 5, May-2018

Concurrency Control Algorithms for different databases: A review

Poonam Sharma
Computer Science Department, Amity School of Engineering and Technology,
Amity University, Haryana, India
e-mail:poonamsharma.2289@gmail.com

ABSTRACT:

Concurrency control is one of the most very important component in managing a transaction, which ensures the correctness of shared data items. Most of the concurrency control techniques use locking mechanism to obtain concurrency control. Transaction processing, concurrency control and recovery issues have played a major role in conventional databases, and hence have been an important area of research for many decades. Concurrency control is a mechanism applied in concurrent operations of database so that all the operation complete successfully without interfering in the results of each other. Several research papers have been presented on concurrency control issues in nested transaction environments. There are several concurrency control techniques that are proposed to prevent data inconsistency. In this paper we are discussing the types of concurrency control techniques (lock based and lock free) available. We are also comparing the lock based and lock-free techniques for concurrency control.

I. INTRODUCTION:

In today's age of information, database is an essential component of any Information system and it can be traditional, distributed, centralized, real-time or mobile. Database is a structured organized way to arrange information. To manage the database in any system, there are many approaches. Database systems are essential for many applications, ranging from space station operations to automatic teller machines. A database state represents the values of the objects of database that represent some real-world entity. The database state is changed from the previous state by the execution of a user transaction. Individual transactions running in isolation are assumed to be correct. When multiple users access multiple database objects residing on multiple sites in a distributed database system, the problem of concurrency control arises.

Concurrency control is a method used to handle conflicts that occur when the data is simultaneously accessed or altered in a multiuser system. It is one of the most important components of transaction management and its main challenge is to ensure that the shared data item when updated by multiple transactions at the same time will remain correct. In order to maintain data consistency, concurrency control is used in Database Management Systems (DBMS) to ensure transactions isolation whenever there are concurrent operations requesting access to the same object and it must be coordinated in order to avoid inconsistencies. Concurrency control methodology permits the users to access the database in a multiple programmed aspect while preserving the wrong idea that each user is executing on a dedicated system. The qualities of a concurrency control algorithm have been evaluated on the basis of response time and throughput. The main dilemma in achieving this is to prevent the database updates performed by one agent/client from interfering with database retrievals and updates are performed by another agent/client. The problem is much worse in a distributed environment because: (i) agents/clients access the data stored in different systems, and (ii) a concurrency mechanism at one system cannot be acknowledged instantly. Two kinds of transactions are available in database management system. They are **Read-only Transactions** (ROTs) and **Update Transactions** (UTs). These transactions should have the ACID properties. ACID properties are **Atomicity, Consistency** (Concurrency), **Isolation** (Independence) and **Durability** (or Permanency).

The database concurrency control strategies in use are based on three mechanisms i.e. pessimistic (locking), Optimistic and time-stamp method. There exist some hybrid techniques but still have a shortcoming similar to that of the three fundamental techniques. **Pessimistic or locking** - Block an operation of a transaction, if it may cause violation of the rules, until the possibility of violation disappears. Blocking operations is very much involved with performance reduction.

Optimistic -Delay the checking of whether a transaction meets the isolation and other integrity rules eg. serializability and recoverability) until its end, without blocking any of its (read, write) operations and then abort a transaction to prevent the violation, if the desired rules are to be violated upon its commit. An aborted transaction is immediately restarted and re-executed, which incurs an obvious overhead (versus executing it to the end only once). If not too many transactions are aborted, then being optimistic is usually a good strategy. **Time stamp** -Timestamp is a unique identifier created by the DBMS to identify the relative starting time of a transaction. Typically, timestamp values are assigned in the order in which the transactions are submitted to the system. So, a timestamp can be thought of as the transaction start time. Therefore, time stamping is a method of concurrency control in which each transaction is assigned a transaction timestamp.

Some of the main goals of concurrency control algorithms are (i) **correctness**, as it the first and foremost goal of any system. Each transactions integrity must be kept while transactions are running concurrently. Thus the integrity of the entire transaction is maintained. (ii) **Serializability,** many problems can occur in DBMS updating if serializability is not maintained. (iii) **Recoverability**, it means that no committed transaction in a schedule has read the data written by an aborted transaction, these data disappear from the database after the transaction get aborted. (iv) **Replication**, the process of copying the data at more than one location for higher rate of availability. In this paper we will be discussing the lock based concurrency control technique and the lock-free concurrency control technique. [3]

II. LOCK BASED CONCUERENCY CONTROL:

A *lock* is a main-memory data item which belongs to an active transaction that permits the transaction access to a particular part of the database. A transaction can execute an action like read, write, insert, delete on a part of the database if and only if it has properly locked the relevant part of the database. A lock includes information about its name, mode, duration, and the owning transaction. The *lock name* identifies the data item or the set of data items in the database that is the target of the lock. The name can be used to categorize locks into logical and physical locks. The name of a *logical lock* identifies a part of the logical database, such as a single tuple in a relation, the set of tuples within a key range, or a whole relation. The name of a *physical lock* identifies a part of the physical database, such as the location of a tuple in a specific data page, or a whole page or file, or a node or a bucket in an index structure.

Locking is the most commonly used method for enforcing transactional isolation. Most database management systems apply some kind of locking, possibly grouped with some other mechanism such as transient versioning. With locking-based concurrency control, transactions are required to protect their actions by acquiring appropriate locks on the parts of the database they operate on. A read action on a data item is usually protected by a shared lock on the data item, which prevents other transactions from updating the data item, and an update action is protected by an exclusive lock, which prevents other transactions from reading or updating the data item. If a transaction requests a lock on a data item in a situation in which some other transaction holds an exclusive lock on the data item, the requesting transaction has to wait for the conflicting lock to be released. Most often this means waiting for the other transaction to commit and all the locks still held by a transaction at commit time are released when the transaction has committed. Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds .Binary Locks − A lock on a data item can be in two states; it is either locked or unlocked. Shared/exclusive − this type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

II.I TYPES OF LOCK PROTOCOL AVAILABLE:

i. **Simplistic Lock Protocol**

Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed. Transactions may unlock the data item after completing the ‘write’ operation.

ii. **Pre-claiming Lock Protocol**

Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand. If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.

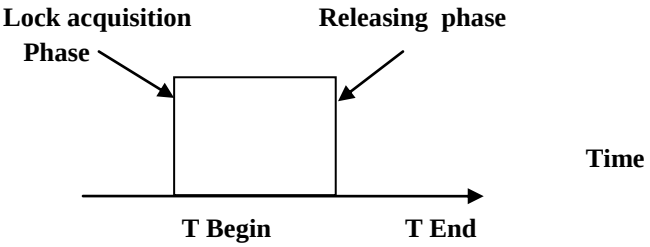

Fig.1 Pre-claiming lock protocol

iii**. Two-Phase Locking 2PL**

This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.

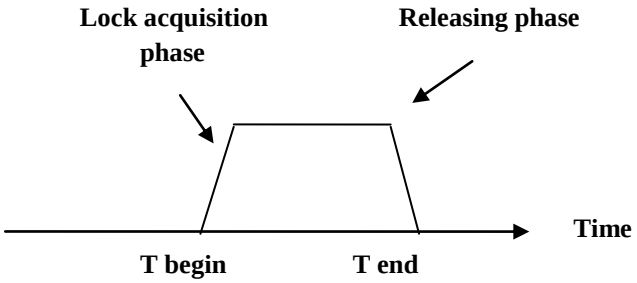

Fig.2 Two phase locking

Two-phase locking has two phases, one is growing, where all the locks are being acquired by the transaction; and the second phase is shrinking, where the locks held by the transaction are being released. To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to an exclusive lock

iv. **Strict Two-Phase Locking**

The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.[2]

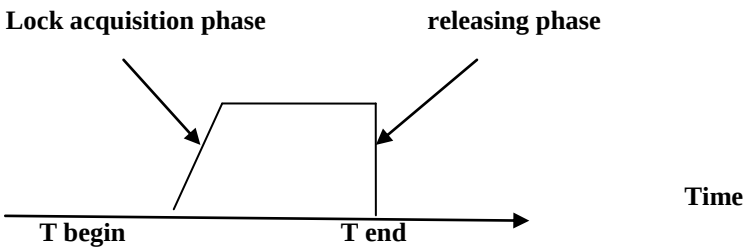

Fig.3 Strictly two phase locking

III. LOCK-FREE CONCURRENCY CONTROL:

Concurrency control is one of the most important components of transaction management, which ensures the correctness of shared data items. Most of the existing concurrency control techniques use locking mechanism to achieve concurrency control, which leads to transaction starvation and deadlock. On the other hand, the non-locking techniques (i.e. optimistic and timestamp ordering) are associated with high abortion rate and excessive transaction restart. The technique allows mobile devices to freely read data items and allowed to pre-commit while in disconnection mode, and latter propagate the pre-committed data during reconnection for global commitment. Optimistic strategies allows each user to freely access a portion of the database without lock, this feature makes optimistic schedules to be deadlock free but it has the problem of higher transaction restart and higher rate of abortion. Timestamp ordering has the advantage of non waiting and low rate of abortion but it's associated with the recovery failure and cascade rollback. Thus, a hybrid strategy is necessary that will combine the features of optimistic and timestamp ordering to achieve a low abortion rate, low transaction restart, non cascade rollback, deadlock free schedules, and recoverable schedules.[1]

III.I LOCK-FREE HYBRID CONCURRENCY CONTROL STRATEGY

A lock-free hybrid concurrency control strategy proposes the combination of the features of both optimistic and timestamp ordering strategies. Like in Optimistic strategy, mobile host can freely copy data item and is free to pre-commit locally but later the results are propagated to the coordinator for making the final commit decision. As in Timestamp Ordering each data item is associated with the timestamp of the last transaction that read it and the timestamp of the last transaction that wrote it. These timestamps are used to make final decision at commit phase. Combining the features of these two strategies, mobile client connect to the server just in the beginning and end of transaction; this reduce communications to minimum. Transaction operators (i.e. read/write) and timestamps (i.e. read/write timestamps) related to particular transactions are managed based on the concept of timestamp ordering schedulers and decisions are made regarding abortion or completion of a transaction. Therefore, this will reduce the number of aborted transactions significantly. Transactions are accomplished in two phases: First phase is the **Reading Phase** which is pictorially represented in Figure 1 while the second Phase is called **Commit Phase.**

READ PHASE:

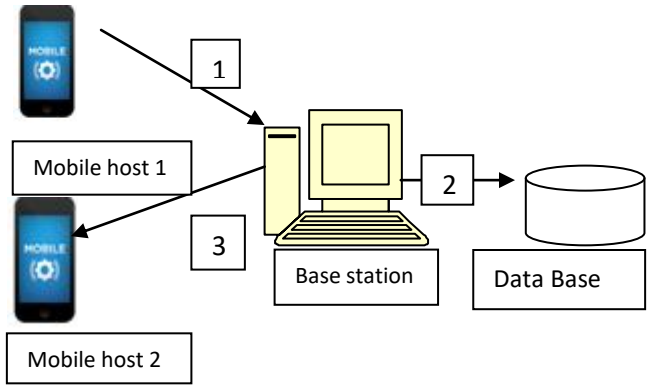

Fig.4 Reading phase

COMMIT PHASE:

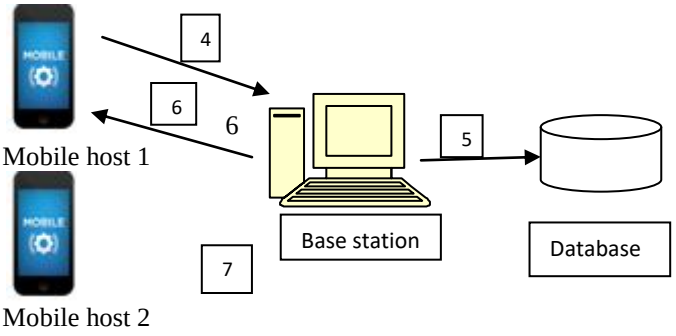

Fig .5 Committing phase

Reading Phase:

Step 1: Mobile host identified by TransactionId issue a

request to read a particular data items.

Step 2: Database is scanned for the required data items and the base station records transaction details. The information will help during conflicts resolution.

Step 3: The required data items are read by the mobile host and the transaction leaves *read timestamp (rts)* as foot print on the data items. It is free to update the copied data items at its own site (local commit) but this would not affect the original data in the database.Go to commit phase

Commit Phase

Step 4: If transaction commits locally, during reconnection the results are submitted to the Basestation to decide on the final commitment. The basestation check to ensure other mobile hosts are not using the same data items modified by the current transaction. If any mobile host is using similar data item it is ask to restart its transaction using the new values of the data item (Conflict Resolution) (step 7).

Step 5: If no other mobile host using similar data item,

base-station updates the data items.

Step 6: All mobile hosts are informed about the final

commit.[1]

Mobile environment face several challenges such as power consumption and frequent transmission interference due to the frequent connection and disconnection of mobile host while transaction is taking place. These challenges make simulation an invaluable tool for understanding the operation of this environment. Whilst real test (i.e. test beds or real life implementation and analytical models) are crucial for understanding the performance of mobile protocols, simulation has been chosen as method of study in this research because it provides an environment with specific advantages over the other methods. The performance in real database systems is determined by several parameters. Some of them are determined by the hardware and operating system architecture, some are set by the administrator, and others are results of the characteristics of the transactions done on the system. Some simulation parameters that have been used in the experiment are shown in the table below.[1]

Parameters	Value	
	Min	**Max**
Number of transactions	50	500
Interval between transactions	100ms	100ms
Operations in short transactions	1	2
Operations in long transactions	2	10
Read probability	60%	
Write probability	40%	
Restart delay	500ms	
Scheduler type	Optimistic,timestamp,hybrid	
Abort probability	0.1%	
Block size	4kb	

Figure 6 depicts the effect of transaction density on the performance of the concurrency control strategies in terms of number of aborted transactions. The figure shows that the number of aborted transactions in lock-free strategy is lower compared to optimistic and timestamp ordering strategies. For instance, when the transaction density is increase to 500, 49 transactions were aborted in lock-free strategy, 65 transactions were aborted in optimistic strategy while 80 transactions were aborted in timestamp strategy. This better performance of lock-free is as a result of its feature which restarts transaction that is aborted in validation phase with a new value as opposed to the basic optimistic and timestamp ordering approaches that aborted the whole transaction which fails in first validation.

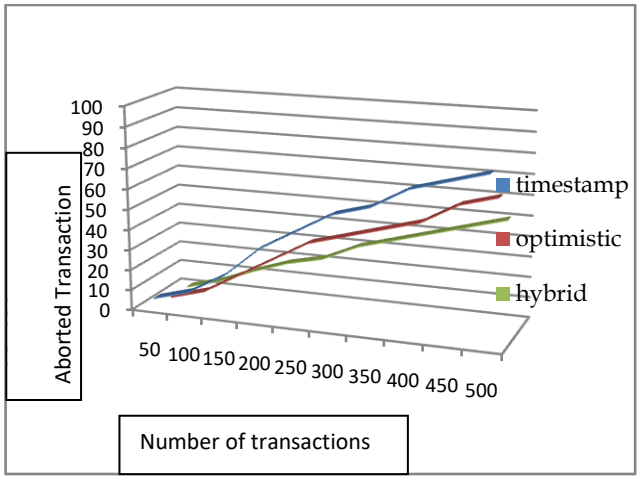

Fig.6 Abort Rate for Hybrid, Optimistic and Timestamp Strategies

Moreover, increasing the number of transactions in all general ships will encounter the possibility for more transaction to access similar data items hence increases the number of aborted transaction. For example, when the number of transaction is 100 only 4 (4%) transaction is aborted in the case of lock-free strategy but when the density is increased to 400 transactions then 39 (10%) transactions are aborted. Similarly in optimistic strategy with 100 transactions, 5 (5%) transactions is aborted while for 400 transactions, 50 (13%) transactions were aborted. Same in the case of timestamp ordering aborts 10 (10%) transactions out of 100 and 70 (18%) out of 400 transactions.[1]

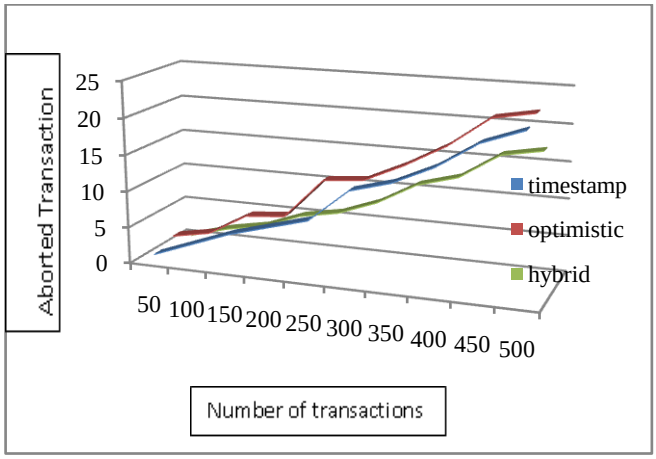

Fig .7 Response Time for Lock-Free Hybrid, Optimistic and Timestamp.

Figure 7 shows respectively the response time for lock-free, optimistic and timestamp ordering strategies .From the result we can see that, performance of lock-free strategy increases almost linearly as the transaction density increases. Similarly, lock-free strategy has a smaller response time as compared to timestamp ordering strategy, which in turn performs better than optimistic strategy. For instance, as the transaction density increases to 400, lock-free strategy has 13.5ms response time, timestamp-ordering with 17ms while optimistic strategy response in 19ms. Hence lock free strategy is found to be better compared to the available lock based strategies.

IV. CONCLUSION:

Concurrency control is one of the important building blocks of transaction management. Most concurrency control method used for real-time database system solves consistency issue. In this paper we have discussed in detail the various goals of concurrency control and the different approaches. The different types of concurrency control algorithms and the advantages of the different types are also discussed in here. In this paper, a lock-free hybrid concurrency control strategy was presented which is based on optimistic and timestamp concept. The performance of hybrid concurrency strategy was analyzed under varying transaction densities.

REFERENCES:

[1]Sirajo Abdullahi Bakura, Aminu Mohammed *"Lock-Free Hybrid Concurrency Control Strategy For Mobile Environment"*

[2]S. Sippu, E. Soisalon-Soininen, "*Transaction Processing*"

Springer International Publishing Switzerland 2014

[3] Sukhdev Singh Ghuman,"*Concurrency Control in DBMS- A Review"* IJCSMC, Vol. 5, Issue. 5, May 2016, pg.599 – 602

[4] Sheetlani Jitendra and Gupta V.K. "*Concurrency Issues of Distributed Advance Transaction Process"* Research Journal of Recent Sciences __ ISSN 2277-2502 Vol. 1 (ISC-2011), 426-429 (2012)

[5] Samuel Kaspi and Sitalakshmi Venkatraman, *Performance Analysis of Concurrency Control Mechanisms for OLTP Databases"* International Journal of Information and Education Technology, Vol. 4, No. 4, August 2014

[6] Ms. Nyo Nyo Yee **"***Concurrency Control Mechanism for Nested Transactions in Mobile Environment "*