International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 5, July - 2012

Analysis of Effectiveness of Concurrency Control Techniques in Databases

Ruchi Dagar
M. Tech Student, CSE Deptt
MRIU, Faridabad

Rachna Behl
Assistant Professor, CSE Deptt
MRIU, Faridabad

ABSTRACT
There is an ever-increasing demand for more complex transactions and higher throughputs in transaction processing systems leading to higher degrees of transaction concurrency. In this paper, we have analyzed different techniques of concurrency control in distributed databases and compared their performance. Ideas that are used in the design, development, and performance of concurrency control mechanisms have been summarized. The locking, time-stamp, optimistic-based mechanisms are included.

Keywords
Distributed database management system, locking protocol, timestamp, optimistic, transaction processing.

1. INTRODUCTION
The databases are the best way of storing the data. A distributed system can be visualized as a set of sites, each site consisting of a number of independent transactions. A distributed database is a database in which storage devices are not all attached to a common CPU. It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers. A database state represents the values of the database objects that represent some real-world entity. The database state is changed 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.
The database system through a scheduler must monitor, examine, and control the concurrent accesses so that the overall correctness of the database is maintained. There are two criteria for defining the correctness of a database: database integrity and serializability. The database integrity is satisfied by assigning a set of constraints (rules) that must be satisfied for a database to be correct. The serializability ensures that database transitions from one state to the other are based on a serial execution of all transactions. This paper presents various concurrency control techniques that maintains both the properties.

2. CONCURRENCY CONTROL

Concurrency control deals with preventing concurrently running processes from improperly inserting, deleting or updating the same data i.e. it ensures that correct results for concurrent operations are generated, while getting those results as quickly as possible. Our main concern in designing a concurrency control techniques is to correctly process transactions that are in conflict. Each transaction has a read set and a write set. Two transactions *conflict* if the two operations belong to different transactions, the read set of one transaction intersects with the write set of the other transaction and the write set of one transaction conflicts with the write set of the other transaction.

2.1. Concurrency Control Problem
If transactions are executed serially, i.e., sequentially with no overlap in time, no transaction concurrency exists. However, if concurrent transactions with interleaving operations are allowed in an uncontrolled manner, some unexpected, undesirable result may occur. Problems that can occur due to multiple transactions executing concurrently are:

Lost update problem: This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect. Suppose that two transactions are submitted at the same time and their operations are interleaved in as shown in figure, then the final value of the item X is incorrect because transaction T2 reads the value of X before T1 changes it in the database and hence the updated value resulting from T1 is lost.
For example if X=80 at the start (originally there were 80 reservations on the flight) and N=5 (T1 transfers 5 seat reservations from the flight corresponding to X to the flight corresponding to Y) and M=4 (T2 reserves 4 seats on X).

The final result should be X=79 but it is X=84 because the update in T1 that removed the 4 seats from X was lost. This is shown in fig 1.

T1	T2
r(X);	
X=X-N;	
	r(X);
	X=X+M;
w(X);	
r(Y);	
	w(X)
Y=Y+N;	
w(Y);	

Fig1: Transactions showing Lost Update Problem

Temporary update problem (dirty read problem): This problem occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value.

For example T1 updates item X and then fails before completion, so the system must change X back to its original value. Before it can do so, transaction T2 reads the temporary value of X which will not be recorded permanently in the database because of the failure of the T1.

T1	T2
r(X);	
X=X-N;	
w(X);	
	r(X);
	X=X+M;
	w(X)
r(Y);	

Fig2: Transactions showing Temporary Update Problem

The value of item X that is read by T2 is called DIRTY DATA, because it has been created by a transaction that has not committed yet, hence the problem is known as dirty read problem.

Incorrect summary problem: If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.

T1	T2
	sum=0;
	r(A);
	sum=sum+A;
r(X);	
X=X-N;	
w(X);	
	r(X);
	sum=sum+X;
	r(Y);
	sum=sum+Y;
r(Y);	
Y=Y+N;	
w(Y);	

Fig3: Transactions showing Incorrect Summary Problem

3. CONCURRENCY CONTROL TECHNIQUES

Different concurrency control techniques are defined by different people. These techniques are unique in their own representation and method.

3.1. Lock Based Concurrency Control Technique

Concurrency control technique used in the development of the current database uses locking technology. In order to ensure serializability scheduling, locking protocol must be observed, that is, if a transaction requests the system for a lock on an entity, and the lock has been given to some other transaction, the requesting transaction must wait. To reduce the waiting time when a transaction wants to read, there are two types of locks that can be employed, based on whether the transaction wants to do a read operation or a write operation on an entity:

Read lock: The transaction locks the entity in a shared mode. Any other transaction waiting to read the same entity can also obtain a read lock.

Write lock: The transaction locks the entity in an exclusive mode. If one transaction wants to write on an entity, no other transaction may get either a read lock or a write lock.

After a transaction has finished operations on an entity, the transaction can do an *unlock* operation. After an unlock operation, either type of lock is released, and the entity is made available to other transactions that may be waiting.

Locking an entity gives rise to two new problems: live_lock and deadlock. Live_lock occurs when a transaction repeatedly fails to obtain a lock. Deadlock occurs when various transactions attempt locks on several entities simultaneously; each transaction gets a lock on a different entity and waits for the other transactions to release the lock on the entities that they have succeeded in securing. The problem of deadlock can be resolved by the following approaches:

Each transaction locks all entities at once.

Assign an arbitrary linear ordering to the items, and require all transactions to request locks in this order.

Gray and Reuter [1] has described experiments in which it was observed that deadlocks in database systems are very rare and it may be cheaper to detect and resolve them rather than to avoid them.

Since the correctness criterion for concurrently processing several transactions is serializability, locking must be done correctly to assure the above property. One simple protocol that all transactions can obey to ensure serializability is called *Two-phase Locking* (2PL). The protocol simply requires that in any transaction, all locks must precede all unlocks. A transaction operates in two phases: The first phase is the growing phase, in which a transaction obtains more and more locks without releasing any. By releasing a lock, the transaction is considered to have entered the shrinking phase. During the shrinking phase the transaction releases more and more locks and is prohibited from obtaining additional locks. When the transaction terminates, all remaining locks are automatically released. The instance just before the release of the first lock is called lockpoint.

T1	T2	
read_lock(Y);	read_lock(X);	
read(Y);	read(X);	expanding
write_lock(X)	write_lock(Y);	phase
unlock(Y);	unlock(X);	
read(X);	read(Y);	
X=X+Y;	Y=Y+X;	shrinking
write(X);	write(Y);	phase
unlock(X);	unlock(Y);	

Fig4: Two-Phase Locking

It is important to point out that locking approaches are in general *pessimistic*. For example, two-phase locking is a sufficient condition rather than the necessary condition for serializability. As an example, if an entity is only used by a single transaction, it can be locked and unlocked freely. The question is, "How can we know this?" Since this information is not known to the individual transaction, it is usually not utilized.

3.2 Time stamp based Concurrency Control Technique

Concurrency control based on timestamp ordering do not use locks, hence deadlocks cannot occur. Timestamp is a mechanism in which the serialization order is selected a priori. Timestamp is a unique identifier created by DBMS to identify a transaction. Timestamp values are assigned by the scheduler in the order in which the transactions are submitted to the system. To achieve unique timestamps for transactions arriving at different nodes of a distributed system, all clocks at all nodes must be synchronized or else two identical timestamps must be resolved. Lamport [3] has described an algorithm to synchronize distributed clocks via message passing.
A concurrency control protocol that orders transactions in such a way that older transactions get priority in the event of a conflict. Read/write proceeds only if last update on that data item was carried out by an older transaction. Otherwise, transaction requesting read/write is restarted and given a new timestamp.

Basic timestamp ordering: Whenever some transaction T tries to issue a read_item(X) or a write_item(X) operation, the basic TO algorithm compares the timestamp of T with read_TS(X) and write_TS(X) to ensure that the timestamp order of transaction execution is not violated. If this order is violated, then transaction T is aborted and is resubmitted to the system as a new transaction with a new timestamp. If T is aborted and rolled back, any transaction T1 that may have used value written by T must also be rolled back. Similarly, any transaction T2 that may have used a value written by T1 must also be rolled back, and so on. This effect is known as cascading rollback and is one of the problems associated with basic TO, since the schedules produced are not recoverable.

The concurrency control algorithm must check whether conflicting operations violate the timestamp ordering in the following two cases:
1. Transaction T issues a write_item(X) operation:
If read_TS(X)>TS(T) or if write_TS(X)>TS(T), then abort and rollback T and reject the operation. This should be done because some younger transaction with a timestamp greater than TS(T), and hence after T in the timestamp ordering, has already read or written the value of item X before T had a chance to write X, thus violating the timestamp ordering.
If the above condition does not occur , then execute the write_item(X) operation of T and set write_TS(X)=TS(T)
2. Transaction T issues a read_item(X) operation:
If write_TS(X)>TS(T), then abort and rollback T and reject the operation. This should be done because some younger transaction with a timestamp greater than TS(T), and hence after T in the timestamp ordering, has already written the value of item X before T had a chance to read X.
If write_TS(X)<=TS(T), then execute the read_item(X) operation of T and set read_TS(X) to the larger of TS(T) and the current read_TS(T).

Strict timestamp ordering: A variation of basic TO called strict TO ensures that the schedules are both strict(for easy recoverability) and (conflict) serializable. In this variation a transaction T that issues a read_item(X) or write_item(X) such that TS(T)>write_TS(X) has its read or write operations delayed until the transaction T' that write the value of X (hence TS(T')=write_TS(X)) has committed or aborted. This algorithm does not cause deadlock, since T waits for T' only if TS(T)>TS(T')

Thomas's Write Rule: A modification of the basic TO algorithm, known as Thomas's write rule, does not enforce conflict Serializability, but it rejects fewer write operations, by modifying the check for the write_item(X) operation as follows:
If read_TS(X)>TS(T), then abort and rollback T and reject the operation.
If write_TS(X)>TS(T), then do not execute the write operation but continue processing. This is because some transactions with timestamp greater than TS(T), and hence after T in the timestamp ordering, has already written the value of X. hence, we must ignore the write_item(X) operation of T because it is already outdatesd and obsolete.
If the above two conditions does not occur, then execute the write_item(X) operation of T and set write_TS(X) to TS(T).

Secure Concurrency Control Protocol (SCCP): SCCP based on timestamp ordering provides concurrency control and maintains security. In a secure distributed database system a security level is assigned to each transaction and data. A security level for a transaction represents its clearance level and the security level for a data represents its classification level.
In a secure distributed database system, the global database is partitioned into a collection of local databases stored at different sites. It consists of a set of *N* number of sites, where each site *Ni* is having a secure database, which is a

partition of global database scattered on all the *N* sites. Each site has an independent processor.
Simple security property: A transaction T is allowed to read a data item (object) x , only if Lv (x) <= Lv (T).
Restricted Property: A transaction T is allowed to write a data item x only if Lv (x)=Lv(T). Thus, a transaction can read objects at its level or below, but it can write objects only at its level.

3.3 Optimistic Concurrency Control Technique

In all the concurrency control techniques a certain degree of checking is done before a database operation can be executed. For example in locking a check is done to determine whether the data item being accessed is locked. In transaction ordering the transaction timestamp is checked against read and write timestamp of the item. Such checking represents overhead during transaction execution with the effect of slowing down the transactions.

In optimistic concurrency control (Validation) technique no checking is done while the transaction is executing. Various concurrency control methods use validation technique. One such method consists of three phases. The three phases for OCC protocol are:

Read Phase: Transaction can read values of committed data items from the database. However, updates are applied only to local copies of the data items kept in the transaction workspace.

Validation Phase: Checking is performed to ensure serializability will not be violated if the transaction updates are applied to the database.

Write Phase: If the validation phase is successful, the transaction updates are applied to the database otherwise, the updates are discarded and the transaction is restarted.

The idea behind OCC is to do all the checks at once. Hence, transaction execution proceeds with a minimum of overhead until the transaction validation phase is reached. The technique is called “optimistic” because it assumes that little interference will occur and hence there is no need to do checking during transaction execution.

4. COMPARITIVE STUDY

In this section the effectiveness of various techniques studied above has been compared. The comparative study of all the techniques is shown in fig5.

Features/ Techniques	**Lock based Technique**	**Time stamp based Technique**	**Optimistic Technique**
Waiting Time	More	Less	Less than locking but more than TSO
Delay	Delay occurs	Delay occurs	No delay
Occurrence of Deadlock	Deadlock occurs	Deadlock-free	Deadlock-free
Serialization Order	Decided dynamically	Decided statically	Decided dynamically

Fig5: Comparative study of various concurrency control techniques

5. CONCLUSION

Different techniques show that there are many other ways to control concurrency in databases. Locking is the simple technique that prevents concurrency but it is considered as the pessimistic approach as it results in deadlock. Timestamp based concurrency control is a deadlock-free technique. Optimistic technique is prior to other techniques

as it assumes that not too many transactions will conflicts with each other. It is also a deadlock-free and allows maximum parallelism. Some studies are being done for object-oriented systems while others are dealing with semantics of transactions and weaker form of consistency.

6. REFERRENCES

[1] J.N. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, San Mateo, Calif., 1993
[2] Bharat Bhargava: Concurrency Control in Database systems: IEEE Transactions on knowledge and data engineering, VOL.11, No1,1999 (IEEE)
[3] L. Lamport, “Time Clocks and the Ordering of Events in a Distributed System,” *Comm. ACM*, vol. 21, no. 2, 1979.
[4] P.A. Bernstein and N. Goodman, “Concurrency Control in Distributed Database Systems,” *Computing Surveys*, vol. 13, no. 2, pp. 85-221, 1981
[5] Abdou R. Ali, Hany M. Harb: Two phase locking concurrency control in distributed databases with N-Tier architecture; IEEE 2004.
[6] Michael J. Carey: Improving the performance of an optimistic concurrency control algorithm through timestamps and versions; *IEEE Transactions on Software Engineering*, vol. SE-13, no. 6, pp. 746-751, June 1987.

[7] Shashi Bhushan, R. B. Patel and Mayank Dave: A Secure Time-Stamp Based Concurrency Control Protocol For Distributed Databases; Journal of Computer Science 3 (7): 561-565, 2007
[8] A. Thomasian: Distributed optimistic concurrency control methods for high performance transaction processing; *IEEE Transactions on Knowledge and Data Engineering*,10(1):173–189, 1998.
[9] E. Rahm: Concepts for Optimistic Concurrency Control in Centralized and Distributed Database Systems; *IT Informationstechnik,* (in German), vol. 30, no. 1, pp. 28-47, 1988.