Paper Name: Database Management System

Topic: Concurrency Control Protocols

Paper Code: BCA CC410

Semester IV

By Ms. Manisha Prasad

Head, Assistant Professor,

Department of Computer Science

Patna Women's College

E – mail : manisha_prasad@yahoo.com

Concurrency Control Protocols/ Techniques

Time Stamp Ordering

We know that there are multiple transactions execute in concurrent manner, trying to access data items at the same time. It is better to decide the order between the transactions before they enter in the system so that in case of any conflict, it can be decided which transaction should execute first.

In the timestamp based method, a serial order is created among the concurrent transaction by assigning to each transaction a unique non decreasing number.

The usual value assigned to each transaction is the value of the system clock i.e. the time at which a transaction enters a system, hence name Timestamp ordering. We know that no two systems will have the same time stamp because at one instance of time only one process can enter the system. A transaction with a smaller timestamp value is considered to be an **'older'** transaction than another transaction with a larger timestamp value.

The serializability that the system enforces in the chronological order of the timestamps of the concurrent transactions. If two transactions Ti and Tj with the timestamp values ti and tj respectively, such that ti < tj are to run concurrently, then the schedule produced by the system is equivalent to running the older transaction Ti first, followed by the younger one Tj.
The contention problem between two transactions in the timestamp ordering system is resolved by rolling back one of the conflicting transactions.

A conflict is said to occur when an older transaction tries to

a) **read a value that is written by a younger transaction.**
b) **Write/modify a value that has already been read or written by a younger transaction.**

In such cases the read and write operations requested by the older transactions are not allowed and they need to rollback and enter the system with a fresh timestamp. Thus the timestamp-ordering protocol guarantees

serializability. Timestamp protocol ensures freedom from deadlock as no transaction ever waits. But the schedule may not be cascade-free, and may not even be recoverable.

Problem with timestamp-ordering protocol:

Suppose Ti aborts, but Tj has read a data item written by Ti, Then Tj must abort. if Tj had been allowed to commit earlier, the schedule is not recoverable.

Further, any transaction that has read a data item written by Tj must abort. This can lead to cascading rollback i.e. that is, a chain of rollbacks

Solution

A transaction is structured such that its writes are all performed at the end of its processing

All writes of a transaction form an atomic action, no transaction may execute while a transaction is being written

A transaction that aborts is restarted with a new timestamp

Locking Protocol

From the point of view of locking scheme a database can be considered as being made up of set of data items. A lock is a mechanism to control concurrent access to a data item A lock is a variable associated with each data item which gives status about the availability of the data item. The value of lock variable is used in the locking scheme to control the concurrent access and manipulation of data associated data item. The locking is done by a subsystem of the database management system called Lock Manager. The Lock requests are made to concurrency-control manager. Transaction can proceed only after request is granted. A locking protocol is a set of rules followed by all transactions while requesting and releasing locks. Locking protocols restrict the set of possible schedules.
Two types of locks are there.

1) Exclusive Locks : It is also called update or write lock. The intension of this mode of locking is to provide exclusive use of the data item to one transaction. If a transaction T locks the data item Q in an exclusive mode, no other transaction can access Q, not even to read Q until the lock is released by T.

2) Shared Lock : It is also called as read lock. Any number of transactions can concurrently lock and access a data item in a shared mode. But none of these transactions can modify the data item. A data item locked in a shared mode can't be locked in exclusive mode, until the shared lock is released.

To implement the locking concept following protocols are used.

1) Two phase protocol : It uses two phases for locking . i.e release of the locks is delayed until all the locks on all data items required by the transaction have been acquired. Two phases are a) Growing phase In this the number of locks increases from zero to maximum for the transaction. In this phase the transaction may obtain lock but may not release any lock. b) Shrinking phase- In this the number of locks decreases from maximum to zero. A transaction may release locks, but may not obtain any new lock. Thus it is a protocol which ensures

conflict serializable schedules. It can be proved that the transactions can be serialized in the order of their **lock points** i.e. the point where a transaction acquired its final lock.

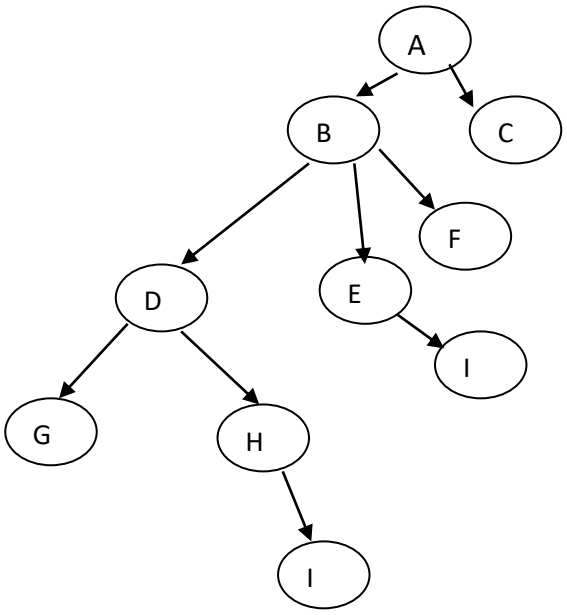
2) Graph based or intension locking protocol. It provides explicit locking at the lower level of the tree and intension locks are applied in all ancestors. Advantage of this protocol is that lock manager knows that some where the lower level node is locked without having the actual examination of lower level nodes. Thus Graph-based protocols are an alternative to two-phase locking

Impose a partial ordering → on the set D = {d1, d2 ,..., dh} of all data items.

If di → dj then any transaction accessing both di and dj must access di before accessing dj .

Implies that the set D may now be viewed as a directed acyclic graph, called a database graph.

The tree-protocol is a simple kind of graph protocol.



Only exclusive locks are allowed.

The first lock by Ti may be on any data item. Subsequently, a data Q can be locked by Ti only if the parent of Q is currently locked by Ti.

- Data items may be unlocked at any time. - The tree protocol ensures conflict serializability as well as freedom from deadlock.

- Unlocking may occur earlier in the tree-locking protocol than in the two-phase locking protocol. shorter waiting times, and increase in concurrency protocol is deadlock-free, no rollbacks are required the abort of a transaction can still lead to cascading rollbacks.

- However, in the tree-locking protocol, a transaction may have to lock data items that it does not access. increased locking overhead, and additional waiting time potential decrease in concurrency

Schedules not possible under two-phase locking are possible under tree protocol, and vice versa.

Validation Concurrency Control Technique

In validation techniques or optimistic scheduling scheme it is assumed that all data items can be successfully updated at the end of the transaction and to read in the data values without locking. Reading is done if the data item is found to be inconsistent (with respect to value read in) at the end of the transaction, then the transaction is rolled back. There are 3 phases in optimistic scheduling.

- Read phase: Various data items are read and stored in temporary local variables. All operations are performed on these variables without actually updating the database.
- During the validation phase, the transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database. If the validation test is positive, the transaction goes to a write phase. If the validation test is negative, the transaction aborts and restarts after discarding the changes.
- Write phase If the transaction passes validation phase the modification made by the transaction are committed i.e Written to the database.

  Optimistic scheduling does not use any lock hence it is deadlock free. However it faces the problem of data starvation. Thus in validation based protocols Execution of transaction Ti is done in three phases. 1. Read and execution phase: Transaction Ti writes only to temporary local variables 2. Validation phase: Transaction Ti performs a "validation test" to determine if local variables can be written without violating serializability. 3. Write phase: If Ti is validated, the updates are applied to the database; otherwise, Ti is rolled back. The three phases of concurrently executing transactions can be interleaved, but each transaction must go through the three phases in that order. It is called as Optimistic concurrency control since transaction executes with a hope that all will go well during validation