HW 10

Concurrency Control

Submitted By:

Aman Pandita, panditaa@oregonstate.edu

Manpreet Kaur, kaurmanp@oregonstate.edu

Question 1:

Consider the following classes of schedules: serializable and 2PL. For each of the following schedules, state which of the preceding classes it belongs to. If you cannot decide whether a schedule belongs in a certain class based on the listed actions, explain briefly. Also, for each 2PL schedule, identify whether a cascading rollback (abort) may happen. A cascading rollback will happen in a schedule if a given transaction aborts at some point in the schedule, and at least one other transaction must be aborted by the system to keep the database consistent (2 points).

The actions are listed in the order they are scheduled and prefixed with the transaction name. If a commit or abort is not shown, the schedule is incomplete; assume that abort or commit must follow all the listed actions.

1. T1:R(X), T2:R(Y), T3:W(X), T2:R(X), T1:R(Y)
2. T1:R(X), T1:R(Y), T1:W(X), T2:R(Y), T3:W(Y), T1:W(X), T2:R(Y)
3. T1:W(X), T2:R(X), T1:W(X)
4. T1:R(X), T2:W(X), T1:W(X), T3:R(X)

Solution 1:

1.

The sequence of operations provided forms a serializability graph, which in this case flows from T1 to T3 and then T3 to T2. Since there's no circular path in this graph, the schedule is deemed serializable. It also follows the two-phase locking (2PL) protocol.

A 2PL-compliant lock sequence can be drawn from this schedule. In this sequence, 'SLock' and 'SRelease' stand for the acquisition and release of a shared lock respectively, while 'XLock' and 'XRelease' denote the acquisition and release of an exclusive lock respectively. This lock sequence can be represented as follows:

T1 obtains a shared lock on X (T1:SLock(X)), then on Y (T1:SLock(Y)). T1 then reads X (T1:R(X)) and releases the shared lock on X (T1:SRelease(X)).

Next, T2 acquires a shared lock on Y (T2:SLock(Y)), reads Y (T2:R(Y)) followed by T3 obtaining an exclusive lock on X (T3:XLock(X)). T3 writes to X (T3:W(X)) and releases the exclusive lock on X (T3:XRelease(X)).

T2 then acquires a shared lock on X (T2:SLock(X)), reads X (T2:R(X)) and releases the shared lock on X (T2:SRelease(X)) and Y (T2:SRelease(Y)).

Lastly, T1 reads Y (T1:R(Y)) and releases the shared lock on Y (T1:SRelease(Y)). In this sequence, if T3 aborts after T2 reads X (T2:R(X)), the system would be compelled to abort T2 as well in order to maintain database consistency. Consequently, this schedule carries the potential for **cascading rollbacks**.

2.

$$\begin{tikzcd} T_1 \arrow[r] & T_3 \arrow[r, bend left] & T_2 \arrow[l, bend left] \end{tikzcd}$$

Here, the serialization has a cycle in T2 & T3. Hence it is not Serializable.

∴ It is not a 2PL Schedule.

3.

$$\begin{tikzcd} T_1 \arrow[r, bend left] & T_2 \arrow[l, bend left] \end{tikzcd}$$

~~Again~~ Again, since there is a cycle between ~~[illegible]~~T1 & T2, Therefore ~~[illegible]~~ ~~[illegible]~~ it is not serialize able & hene not a 2PL

4.

$$\begin{tikzcd} T_1 \arrow[r, bend left] \arrow[rr, bend left=40] & T_2 \arrow[l, bend left] & T_3 \end{tikzcd}$$

Again due to the cycle between T1 & T2, serialization is not possible & hence it is not 2PL

Question 2:

Consider the following schedule.

	T1	T2
0	start	
1	read X	
2	write X	
3		start
4		read X
5		write X
6		Commit
7	read Y	
8	write Y	
9	Commit	

What are the maximum degrees of consistency for T1 and T2 in this schedule? You must find the maximum degrees of consistency for T1 and T2 that makes this schedule possible (1 point).

Solution 2:

At the onset of its operation, T1 performs read and write operations on the data element X. However, since T2 has the ability to both read and write on the same data element X before T1 completes its operation, it's evident that T1 doesn't possess a prolonged exclusive lock on X. As such, the only type of exclusive lock T1 could potentially have on X is a short one. Consequently, the highest possible consistency degree for T1 is zero. The schedule doesn't indicate the involvement of another transaction with X prior to T2 completing its operation.

Now, as in this schedule, T2 can have a long exclusive and shared lock on X. The maximum degree of consistency of T2 will be 3.

Question 3:
Consider the following protocol for concurrency control. The database system assigns each transaction a unique and strictly increasingly id at the start of the transaction. For each data item, the database system also keeps the id of the last transaction that has modified the data item, called the transaction-id of the data item. Before a transaction T wants to read or write on a data item A, the database system checks whether the transaction-id of A is greater than the id of T. If this is the case, the database system allows T to read/write A. Otherwise, the database system aborts and restarts T.

(a) Does this protocol allow only serializable schedules for transactions? If not, you may suggest a change to the protocol so that all schedules permitted by this protocol are serializable. You should justify your answer. (1.5 points)
(b) Propose a change to this protocol or the modified version you have designed for part (a) that increases its degree of concurrency, i.e., it allows more serializable schedules. (1.5 points)

Solution:

a.
No, the given protocol does not ensure that transaction schedules are serializable. Because it permits transactions to read or write data items even when their transaction-id is not larger than the id of the current transaction, this protocol permits non-serializable schedules.

We can alter the protocol by implementing stringent two-phase locking (2PL) to guarantee serializability. Prior to executing any data access, a transaction must get all of its locks, and it must maintain all of its locks until the transaction is committed or terminated.

The improved protocol makes sure that transactions acquire locks in a tight sequence and keep them until the transaction is finished by enforcing stringent 2PL. Since no transaction may access a data item until the prior transaction holding a conflicting lock has finished its operation and released the lock, this ensures serializability.

b.
For the proposed idea in (a), a more and less conservative Optimistic Concurrency Control technique can be used. This can be done without acquiring any locks along with reducing lock contention and improving concurrency. There would be 3 phases to it:

- The transaction reads data from the database during the read phase (or execution phase), but it has not yet written anything. A local copy of the database is the only one that gets updates.

- Phase of Validation: Before committing, the transaction looks for any conflicts with other transactions. In most cases, this is accomplished by determining whether any other transaction that affected the database in the intervening period would have resulted in a conflict.

- Write Phase (or Commit Phase): The modifications from the transaction are applied to the database if the validation phase is successful (no conflicts). The transaction is restarted if it fails (possible conflict).

This increases concurrency since more transactions may run concurrently without waiting for locks. The catch is that OCC works great when there are very little conflicts. Transactions would need to restart often if conflicts occur frequently, which would have a severe effect on performance.