THE UNIVERSITY OF MELBOURNE
SCHOOL OF COMPUTING AND INFORMATION SYSTEMS
COMP90020 DISTRIBUTED ALGORITHMS

Tutorial Week 11: Transactions and Concurrency Control Continued.

Solutions

46.

(a) $rl(x, T1), r(x, T1), wl(y, T2), w(y, 9, T2)$, CONFLICT - T1 holds a read lock on x but we need to write to x in order to progress, T1 could release the read lock but then it would violate 2PL as it needs to acquire a WL on y. Deadlock.

(b) $wl(y, T2), w(y, 9, T2), rl(x, T1), r(x, T1), wl(x, T1),$
$w(x, 8, T1), ul(x, T1), r(y, T2), ul(y, T2)$. No deadlock

(c) $rl(x, T1), r(x, T1), wl(y, T2), W(y, 0, T2), rl(x, T2),$
$r(x, T2), ur(x, T2), wl(x, T1), w(x, 8, T1), uw(x, T1), ur(y, T2)$. No deadlock.

47.

(a) No, there is no guarantee of consistent retrievals. Consider a transaction reading two objects x,y. If there was a contending transaction that wrote to both x,y between the release of some initial read lock on x, to the subsequent read on y - it is possible that the read only transaction would have gotten values for x,y that are impossible to exist together.

(b) No, since releasing read locks early does not affect the state of the objects themselves - it is not possible (within a single tx) to have inconsistent object state. Note that after having a inconsistent retrieval, any subsequent txs using that retrieval is undefined behaviour.

48. consider the following input schedule: $r(x, T1), r(x, T2), w(x, T1), w(x, T2)$. In this case, there is a shared lock on x, after which T1 wishes to gain an exclusive (write) lock, however T2 later requires the same and therefore cannot give up its lock without violating 2PL.

One must abort. Lets say that T2 aborts by virtue of being (2) and having a higher tid

The tx is rescheduled - yet consider that this situation occurs repeatedly, where it repeatedly re-enters with a higher tid and is killed.

Change the aborting scheme - introduce timestamps and use a method such as wait and die.

Alternatively we can also consider conservative two-phase locking as discussed in 50.

49. Since in optimistic concurrency control, Txs don't acquire locks and instead retroactively / proactively check for conflicts before either aborting or proceeding - deadlocks are not possible.

In terms of starvation - depending on the method used it is possible however timestamp ordering - (assuming logical) prevents starvation

50.

(a) A transaction can only acquire locks prior to execution if these do not conflict with any others. The transaction scheduler will attempt to acquire all necessary locks on behalf of the trnasaction and if impossible the transaction has to wait, however it follows that under C2PL a blocked (or waiting) transaction never holds a lock. In simple terms - acquiring locks can be seen as atomic - similar to strict 2PL releasing all locks immediately at the time of commit.

(b) No, it is not possible. Given all locks need to be acquired before any operations can be done by a Tx - there cannot be a dynamic set read/write set (as this could require additional locks to be acquired). In some sense, you could lock every single object that exists and satisfy C2PL but then there would be no difference between a sequential execution.