Module 12: Lab

Elita Danilyuk

1. Rigorous 2PL with timestamps used for deadlock prevention (using Wait-Die policy).

- S1:
  - T1 acquires shared lock on X
  - for an exclusive lock on X, T2 will be aborted when asked because it has a lower priority
  - T3 gets an exclusive lock on Y
  - when T1 asks for an exclusive lock on Y (still held by T3) since T1 has a higher priority, so T1 will be blocked waiting
  - T3 finishes writes, commits, then releases all locks
  - T1 wakes up, gets the exclusive lock on Y and finishes
  - T2 can now be restarted successfully
- S2: similar to the sequence of S1 above, except T2 would be able to advance a bit further before it gets aborted

2. Rigorous 2PL with deadlock detection. (Show the waits-for graph in case of deadlock.)

- Deadlock detection allows transactions to wait and they're not aborted until a deadlock is detected.
- S1:
  - T1 gets shared lock on X
  - T2 blocks waiting for exclusive lock on X
  - T3 gets exclusive lock on Y
  - T1 blocks waiting for exclusive lock on Y
  - T3 finishes, commits, releases locks
  - T1 wakes up, gets exclusive lock on Y, finishes, releases locks on X & Y
  - T2 gets exclusive locks on X & Y then finishes
  - (No deadlock)
- S2: a deadlock occurs, T1 waits for T2, while T2 waits for T1

3. Timestamp concurrency control with buffering of reads and writes (to ensure recoverability) and the Thomas Write Rule.

- initiate timestamp of X and Y to 0
- S1:

  T1: R(X) allowed $TS(T1) \geq WTS(X) = 0 \rightarrow TS(T1) \geq WTS(X)$. $RTS(X) \rightarrow 0$

  T2: W(X) allowed $TS(T2) = 2, RTS(X) = 1, WTS(X) = 0$

  $\rightarrow TS(T2) \geq RTS(X)$ & $TS(T2) \geq WTS(X)$. $WTS(X) \rightarrow 2$

  T3: W(Y) allowed $TS(T3) = 3, RTS(Y) = 0, WTS(Y) = 2$

  $\rightarrow TS(T2) \geq RTS(X)$ & $TS(T2) \geq WTS(X)$. $WTS(X) \rightarrow 3$

  T1: W(Y) ignored $TS(T1) = 1, RTS(Y) = 0, WTS(Y) = 3$

  $\rightarrow TS(T2) \geq RTS(X)$ & $TS(T1) < WTS(Y)$
- S2: same as sequence S1 above

4. Validation (Optimistic) version control

- Both transactions will execute, read from the db and write to a private workspace. Then they get a timestamp to enter validation. The timestamp of transaction T_i is i.
- S1: T1 gets earliest timestamp so it commits without any issues, but when T1 validates T2 against T1, one of the following must hold true:
  - T_1 completes before T_2 begins
  - T_1 completes before T_2 starts a write & T_1 doesn't write any db object read by T_2
  - T_1 completes its read before T_2 completes its write & T_1 doesn't write any db object that's read or written by T_2

  none of the hold true, so T_2 will be aborted and restarted later, so T_3 is the same as T_2
- S2: same as sequence S1 above

5. Multiversion timestamp concurrency control

- S1: T1 reads X therefore RTS(X) = 1
  T2 can write X; TS(T2) ≥ RTS(X), RTS(X) & WTS(X) → 2
  T2 writes Y; RTS(Y) & WTS(Y) → 2
  T3 can write to Y; TS(T1) < RTS(Y)
  → T1 would need to be aborted & restarted later with a larger timestamp
- S2: similar to the sequence of S1 above