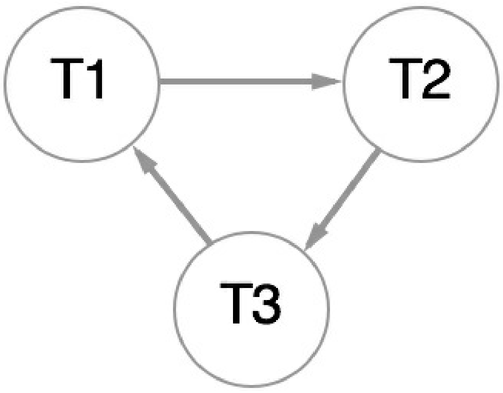
Assignment 2 - ACS 2015

Question 1: Serializability & Locking

A schedule is conflict-serializable if and only if it's precedence graph has no cycles. This is a graph of nodes and vertices, where the nodes are the transaction names and the vertices are attribute collisions.

Schedule 1 is not conflict serializable because the graph is cyclic:

- T1 - T2: read-write conflict on X.
- T2 - T3: write-read conflict on Z.
- T3 - T1: read-write conflict on Y .

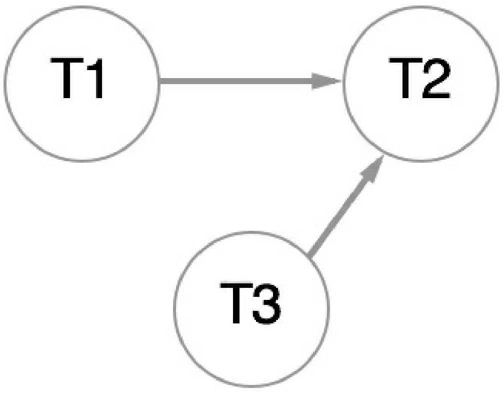


$$\begin{tikzcd}
T1 \arrow[rr] & & T2 \arrow[dl] \\
& T3 \arrow[ul] &
\end{tikzcd}$$

Because T1 has a shared lock on X when T2 writes to it, the schedule cannot be generated by S2PL.

Schedule 2 is conflict serializable because the precedence graph is acyclic:

- T1 - T2: X is accessed by T2 after T1 has committed.
- T3 - T2: Z is exclusively locked by T3, and is released prior to T2 acquiring a shared lock.



$$\begin{tikzcd}
T1 \arrow[rr] & & T2 \\
& T3 \arrow[ur] &
\end{tikzcd}$$

The schedule can be generated by S2PL because we can insert locks prior to the respective reads and writes to the objects in question.

Question 2: Optimistic Concurrency Control

Scenario 1: Because T1 finishes before T3 starts, the 1st condition holds. We then have to check that the 2nd condition holds for T2 and T3, but because T2 writes to the object that T3 reads from, this condition does not hold. Therefore T3 has to rollback.

Scenario 2: The 2nd validation condition does not hold for T1 and T3 because T1 writes to object 3 and T3 reads from it. Therefore T3 has to rollback. We can observe that the 3rd holds for T2, because T3 does not access object 8.

Scenario 3: The only object that T1 writes to is object 4 and because T3 does not read or write to that object, the 2nd validation condition holds. Then we check if the 2nd condition holds for T2 and T3. T2 only writes to object 6 and T3 does not access that object in any way, therefore T3 can commit.

Questions for Discussion

1. **(a)** For the implementation of the locking protocol I have used the ReentrantReadWriteLock class available in the java.util.concurrent.locks package.

   A ReentrantLock is owned by the thread last successfully locking, but not yet unlocking it. A thread invoking lock will return, successfully acquiring the lock, when the lock is not owned by another thread. The method will return immediately if the current thread already owns the lock.

   The ReentrantReadWriteLock allows both readers and writers to reacquire read or write locks in the style of a ReentrantLock. Non-reentrant readers are not allowed until all write locks held by the writing thread have been released. Additionally, a writer can acquire the read lock, but not vice-versa. Among other applications, reentrancy can be useful when write locks are held during calls or callbacks to methods that perform reads under read locks. If a reader tries to acquire the write lock it will never succeed.

   (b) Testing has been done locally and every client has been emulated as a thread that uses the helper classes implemented by the runnable interface. Creating several threads makes it possible to deal with the requests concurrently.

1. The protocol is implemented by using conservative S2PL by acquiring a lock at the beginning of each method and releasing it before the method returns.

2. Deadlocks cannot occur because the locks are acquire right at the start of the methods, transforming every call into an atomic one. This approach impacts the performance by lowering concurrency, but ultimately assures proper synchronization.

3. Scalability is a problem because the entire datastore is locked while the database is being modified. The difference between readers and writers have been taken into account and allows for multiple reads but only single writes. A solution to this problem would be the fact that the writing locks are acquire on the table entry level. For example, if a client modifies the stock of a book in the book

store, it should not wait for the stock update when adding a rating.

4. It is safe to assume that the overhead is not that big. Because we are dealing with a bookstore, the customers that get and read more information about the books are more frequent, compared to the buyers that modify and change the bookstore's stock and ratings. The implemented locking protocol is suitable enough for providing adequate isolation and performance.