Assignment 3
Due: May 12, 2022

Question 1: Identify a schedule that would be allowed by a 2PL protocol but would not be generated by the validation protocol. Explain how the schedule you identify could be generated using 2PL by associating an appropriate lock and unlock operations. Also, explain clearly why the schedule would not be generated by the validation protocol. Now attempt do the same where you specify a schedule that a validation protocol can generate but would not be allowed by 2PL. Can you identify such a schedule? Explain.

Question 2: Let us define a scheduler as supporting **maximum concurrency** if it can generate the set of all conflict serializable schedules. For example, 2PL does not support maximum concurrency since the following schedule

$w_1(x)r_2(x)w_3(y)r_1(y)$

which is conflict serializable can never be generated by the 2PL scheduler. Similarly, it is not difficult to see that a TO scheduler does not support maximum concurrency. On the other hand, the SGT scheduler does support maximum concurrency but it can result in transactions aborting. Design a conservative version of the SGT scheduler that, like the basic SGT scheduler, supports maximum concurrency and that does not result in any transaction aborts. (Assume that transactions predeclare their read and write operations. List out all the steps of the conservative SGT scheduler you design. In particular, specify what happens when a transaction begins execution, what does the scheduler do in deciding on whether or delay an operation, and when a transaction commits.

Question 3: Paper-reading: In the performance study in [Agarwal et. al.] it is shown that under the assumption of infinite resources (that is, very low resource contention) dynamic 2PL starts thrashing as the MPL (Multiprogramming Level) is increased beyond a certain level, much before the optimistic scheme starts thrashing. In fact, in their experiments the throughput of optimistic scheme keeps increasing with the increase in MPL for the entire range of MPL tested. Explain why the dynamic 2PL scheme thrashes with increased MPL while the throughput of optimistic scheme keeps increasing.

Question 4: Illustrate using an example why a protocol will not work correctly (that is, it may not ensure serializability), if transactions acquired an S-lock to read and an X-lock to update data items. Try to generalize the protocol such that serializability can be ensured without requiring that transactions acquire X-locks on data items whenever they need to either read or write the data item.

Question 5: Consider a hot spot data item whose value needs to be maintained in the range 0 and max. Such a data item is maintained as a range [a, b], where the range signifies that the value of the data item at any instance is greater than equal to a and less than equal to b. A transaction wishing to decrement the data item acquires an E mode lock on the hot spot. Two E mode locks do not conflict, whereas an E mode lock conflicts with other S and X lock modes. After acquiring the E mode lock, the transaction tests the escrow logic, that is, it tests if the return value of its decrement will succeed for any value of the hot spot in the range. If that is the case, the transaction proceeds with its operation. Else, it can either delay or abort. Note that some mechanism has to be inplace to prevent two transactions from testing their escrow logic concurrently. In our discussion on the escrow technique in class, we had assumed that transactions only decrement the hot spot data item (e.g., reserve seats in the airline reservation example) by a given amount and do not increment the hot spot (e.g., cancel a reservation). Generalize the scheme discussed in the class to handle the case in which transactions may also increment the hot spot.

Question 6: In class we discussed that one common cause of deadlock is when a transaction holding an S lock wishes to convert its lock to an X mode. Two such transactions, both holding S lock on a data item, if they request lock conversion to X mode will result in a deadlock. One way to address this is to support an (U)pdate mode lock. A transaction that could possibly update the data item requests a U lock. A U lock is compatible with the S lock but is incompatible with other U and X locks. If the transaction, holding a U lock, decides to update the data item, it upgrades its lock to an X mode. Since a U lock is incompatible with other U locks, deadlock is prevented without preventing other transactions read access to the data item. One problem with this approach, however, is that the transaction that does eventually require to convert its U lock to an X lock may be starved if there is a steady flow of S mode requests on the data item (since S mode and U modes are compatible in our scheme). Note that this problem would not arise if the transaction had acquired an X mode lock instead of a U lock. However, that would result in lower concurrency. Suggest a refinement of the update mode locking that does not result in a loss of concurrency, and that at the same time prevents possible starvation of transaction's lock conversion request. Try to design a solution that does not complicate the logic of the lock manager by associating priorities with different transactions. (Hint: you may need to add additional lock types.)

Question 7: Write a brief summary of the [Tictoc paper](#) on validation based protocol. Please answer the following questions after summarizing the paper:

    a. What is the principal novelty of the paper?

    b. Describe the serializable guarantee provided by TicToc. Compare this to Conflict serializability.

    c. Describe the difference between Timestamp ordering in TicToc and the Multi-version timestamp ordering discussed in class.

    d. Present a valid critique of the paper (based on experiments or design discussion in the paper).