ACS Programming Assignment 2

Kai Arne S. Myklebust, Silvan Adrian

Handed in: December 6, 2018

Contents

1 Question 1

1.1 (a)

We achieve before-or-after atomicity by either using shared or exclusive locks on the whole database. So that a read gets either the old or the new written data. In two level locking we set a global exclusive lock when inserting or deleting books and set a global shared lock otherwise. By this we achieve before-or-after atomicity on insertion and deletion. We also have local locks for each book (isbn). When we update a book we set an exclusive lock on this book and set shared lock otherwise. Hereby we make sure that only one client can update a book at a time and we achieve before-or-after atomicity.

1.2 (b)

We did several tests with concurrent threads. We bought books and added copies of books to ensure that we do not write on the same book at the same time. Also we checked for read problems, by reading while writing to a book and looking for differences there.

1.3 (c)

We did not feel the need to consider different testing strategies, because both classes are supposed to do the same thing. Different testing strategies would not be violation of modularity, but we can still consider it because different implementations can have different edge cases.

2 Question 2

For the SingleLockConcurrentCertainBookStore we have a version of Conservative Strict Two Phase Locking where we instead of having locks on data items have a lock on the entire database. The locking mechanism ensures that write operations from different threads occur sequentially. But we get concurrency improvements by having shared locks allowing read operations from multiple threads at the same time. By this we believe our locking in SingleLockConcurrentCertainBookStore is correct. For the TwoLevelLockingConcurrentCertainBookStore we use a version of Strict Two Phase Locking where we have local locks on each book. The books get locked during execution and all locks get released afterwards. In addition we have a global lock which ensures that inserts and deletions of books is sequential. Our locking protocol ensures sequential writes on single books and concurrent reads so we believe that our locking in TwoLevelLockingConcurrentCertainBookStore is correct.

3 Question 3

With `SingleLockConcurrentCertainBookStore` there won't be any deadlocks since we ensured writes are sequential (exclusive lock on Database) while reads can obtain a shared lock. For `TwoLevelLockingConcurrentCertainBookStore` we can experience deadlocks. With the local exclusive locks we might end up in a situation that threads wait on each other indefinitely, which would result in a deadlock.

4 Question 4

For the SingleLockConcurrentCertainBookStore there is the possible bottleneck of multiple clients wanting to write data, but all need to wait for each other due to the exclusive locks. The same problem can we get for TwoLevelLockingConcurrentCertainBookStore if all clients want to add new or delete books, because we have global exclusive locks on inserts and deletions. In Two Level Locking we do not have that big of an issue with scalability, because we can update multiple books at the same time. Something that is not possible in Single Lock. If all clients only want to read we could scale infinitely, but since most clients also want to write something Two Level Locking provides the best scalability.

5 Question 5

In `SingleLockConcurrentCertainBookStore` the concurrency is limited by the number of write operations, since those need an exclusive lock on the whole Database which puts all other operations on wait. If there are not too many write operations in comparison to read operation on the bookstore then the overhead would be more or less small which would be a good tradeoff. For `TwoLevelLockingConcurrentCertainBookStore` we do have a bigger overhead since our local locks have to be saved in a map, so we have to get the lock always out of the map. But the reach a higher concurrency (local write locks don't lock the whole database), which helps us in processing lots of requests in comparison to `SingleLockConcurrentCertainBookStore`