Concurrency Control in Transactional Systems: Spring 2023

Programming Assignment 2: Implementing BTO and MVTO algorithms

Rohit Kapoor(CS21MTECH12011)

Report

Objective:

In this assignment we have make total 4 schedulers one BTO and 3 variants of MVTO.

Details As shown in the book, we have to implement BTO and MVTO algorithms studied in the class in C++. But unlike the version that we studied in the class, we have to implement these algorithms you are using optimistic concurrency control approach, i.e. all writes become visible only after commit. It can be seen that the advantage of this approach: upon an abort of a transaction, no rollback is necessary as none of the writes of the transactions will ever be visible.

MVTO Variants: An important requirement with MVTO algorithm is to delete the unwanted (garbage) versions. There are two ways to do it. Either through garbage collection. Or have a fixed number of of versions, say like 5. Once all the 5 versions have been written onto, the next transaction write overwrites the first version. This way at any time only a fixed number of versions (5 in this case) will be in the system for any data-item. So, we have to implement the garbage collection procedure as well that we studied in the class. So for MVTO, we have to implement three variants:

1. MVTO: Normal MVTO that we studied in the class.
2. MVTO-gc: The variant of MVTO with garbage collection.
3. K-MVTO: The variants with k versions of a data-item maintained any time.

Implementation:

BTO **:**

- *begin_trans():* In this function we are basically creating transaction object. After creating transaction object we assign id to them atomically ans lastly we returned the value of transaction id.
- *read():* In this function we validatng the read operation by comparing the *Time stamps*. i.e if some transaction T_i wants to perform read operation on a dataitem D_i. So we allow this read operation only when *Write Timestamp*(D_i)<*Timestamp*(T_i)And on a succesfull read operation it store the dataitem in it's local buffer and return 1.
- *write():* In this function we validatng the write operation by comparing the *Time stamps*. i.e if some transaction T_i wants to perform write operation on a dataitem D_i. So we allow this read operation only when (*Write Timestamp*(D_i) and *Read Timestamp*(D_i))<*Timestamp*(T_i)And on a successful write operation it modified the dataitem in it's local buffer and return 1.
- *tryC():* In this function we check the correct status of the transaction which invoked the *tryC* function. If the transaction status is not abort then we copy the local buffer data to the *shared memory*.

MVTO, MVTO-K, MVTO-GC **:**

- *begin_trans():* In this function we are basically creating transaction object. After creating transaction object we assign id to them atomically ans lastly we returned the value of transaction id.

- *read():* In this function we validating the read operation by comparing the *Time stamps*. i.e if some transaction T_i wants to perform read operation on a dataitem D_i. So we allow this read operation only when *Write Timestamp*(D_i)<*Timestamp*(T_i)And on a successfully read operation it store the dataitem in it's local buffer and returns the local value.
- *write():* In this function we validatng the write operation by comparing the *Time stamps*. i.e if some transaction T_i wants to perform write operation on a dataitem D_i. And on a successful write operation it modified the dataitem in it's local buffer. writing on a dataitem creates a new version. In MVTO with K versions can store only K version of a specefic dataitem.On a succesful write operation it returns a non-negative value and on unsuccesful write operation it set transaction status as *Abort*.
- *tryC():* In this function we check the correct status of the transaction which invoked the *tryC* function. If the transaction status is not abort then we copy the local buffer data to the *shared memory*.

Comparison:

Test Conditions: i)No. of Threads=10. ii)lambda=100 iii) In MVTO-K K=30.

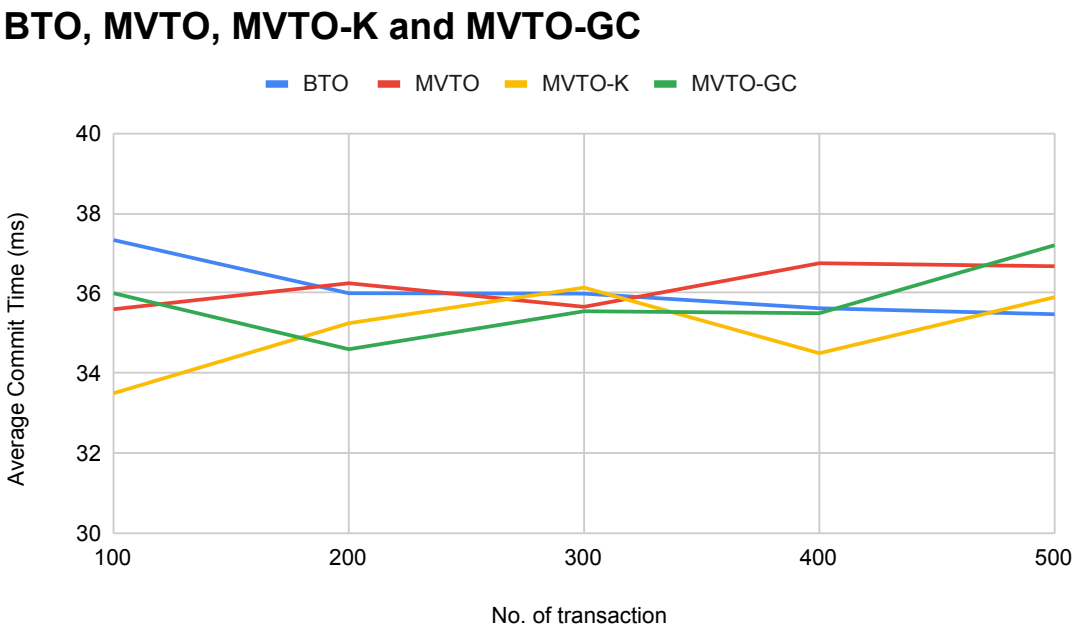

Figure 1: Average commit time Vs No. of Transactions

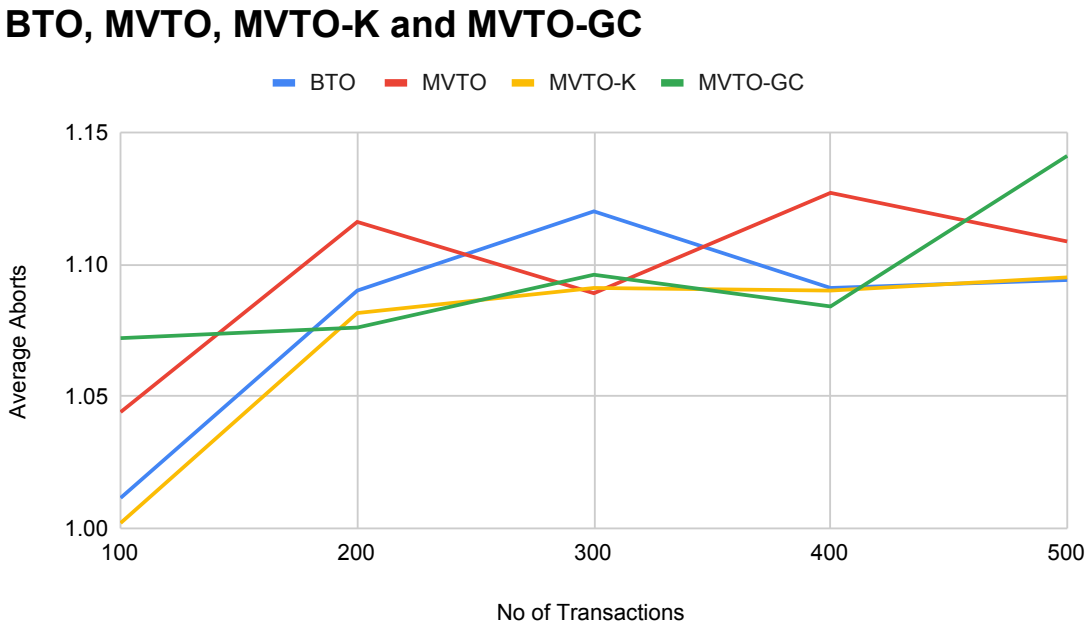

Figure 2: Average Aborts Vs No. of Transactions

Observation:

According to my implementation MVTO with K versions perform better then than BTO,MVTO and MVTO with garbage collection.The number of aborts and the average commit time are comparatively less in MVTO with K versions.The reason behind it is the restricted no of versions, because of it algorithm required less time to search the versions.The second best scheduler is BTO,then third is MVTO and lst one is MVTO with garbage collection.Also it is observed that the Commits to aborts ratio is equal to 1 or greater than 1.