Concurrency Control in Transactional Systems

Programming Assignment 1: Implementing BOCC & FOCC algorithms

Name- Souvik Sarkar   Roll no- CS23MTECH02001

INTRODUCTION: In this assignment we built 3 types of optimistic scheduler .

- BOCC
- FOCC_CTA(In this case if we find any conflict between “current transaction” and “other concurrent transactions” then we will abort the current transaction)
- FOCC_OTA(In this case if we find any conflict between “current transaction” and “other concurrent transactions” then we will abort the other concurrent transactions)

DESIGN :

- For the Transaction , DataItem object for the respective scheduler I used class.
- In BOCC implementation I separately declare every classes in “.h” file and in “.cpp” file. And for main() and update_mem() function I wrote another “opt-test.cpp” file.
- For implementing FOCC_CTA & FOCC_OTA , I used only one “FOCC_CTA.cpp” & “FOCC_OTA.cpp” file .

Some Implementation details :

- Data Structures used :

```
unordered_map<int,Transaction*> transaction
```

  - For storing he Transactions i used a unordered_map , where I stored mapped Transaction id with the transaction pointers.

```
vector<Dataitem*> dataitem ;
```

  - For storing the data items I used vector of dataitem pointer.

```
set<Transaction *>readlist ;
set<Transaction *>writelist ;
```

  - Readlist and writelist I used vectors of transaction pointers.

```
set<int>readset ;
set<int>writeset ;
```

  - For readset and writese I used vectors if ints to store the transaction ids.

- begin_transaction() :
  - For assigning the transaction id we are using a Atomic counter.

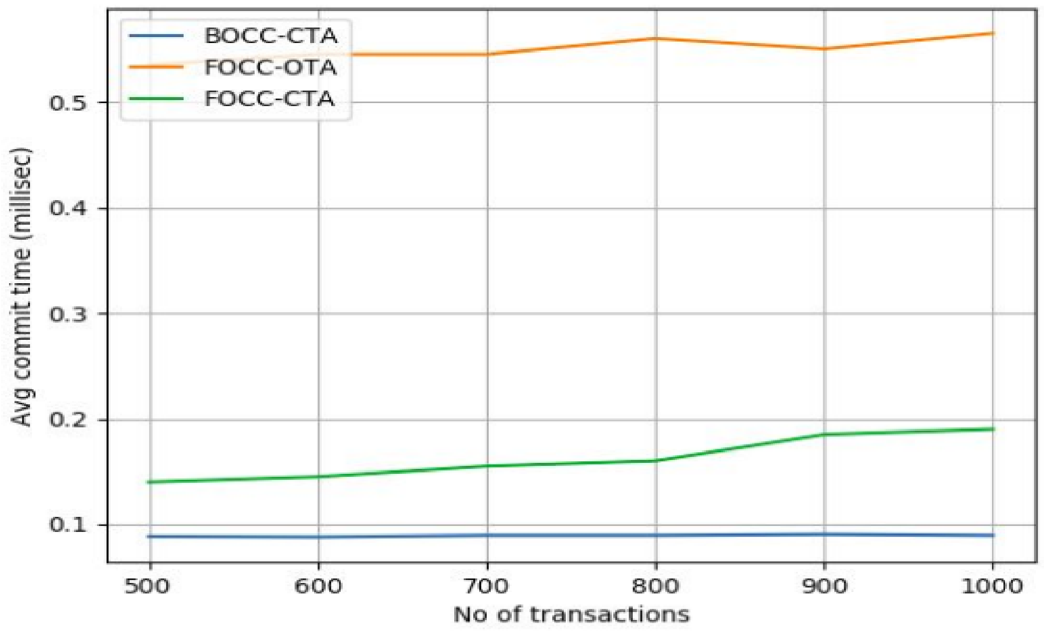
    - And after that we are storing the transaction's id along with it's transaction pointer. And returning the id.
- read() :
    - It first acquires the lock of the dataitem .
    - Then copy that data to its local buffer along the index.
    - Add that transaction id to that data items readlist.
    - It first the unlock of the dataitem .
- write() :
    - In write function we do not need any data lock because it basically perform on the local buffer.
    - Add that transaction id to that data items writelist.
- <u>tryC()</u>:
    - Try commit function is basically the validation phase.it is invoked when transaction want to commit.
    - In this function we are maintain a data structure for maintaining the conflict transaction base on that we "commit" or "abort" transaction.
    - <u>BOCC::tryC()</u> :
        - If in the Bocc if conflict transaction list is not empty then that implies that current transaction is conflicting we other transaction so we "aborting" the current transaction.
        - After that we remove the current transaction's id from all the dataitems readlist.
        - And then delete the transaction from the transaction list.
    - <u>FOCC_CTA::tryC()</u>:
        - If in the focc_cta if conflict transaction list is not empty then that implies that current transaction is conflicting we other transaction so we "aborting" the current transaction.
        - After that we remove the current transaction's id from all the dataitems readlist.
        - And then delete the transaction from the transaction list.
    - <u>FOCC_OTA::tryC()</u>:
        - If in the focc_ota if conflict transaction list is not empty then that implies that current transaction is conflicting we other transaction so we "aborting" the all other concurrent conflicting transaction.
        - After that we remove the other concurrent conflicting transaction's id from all the dataitems readlist.
        - nd then delete the transactions from the transaction list.

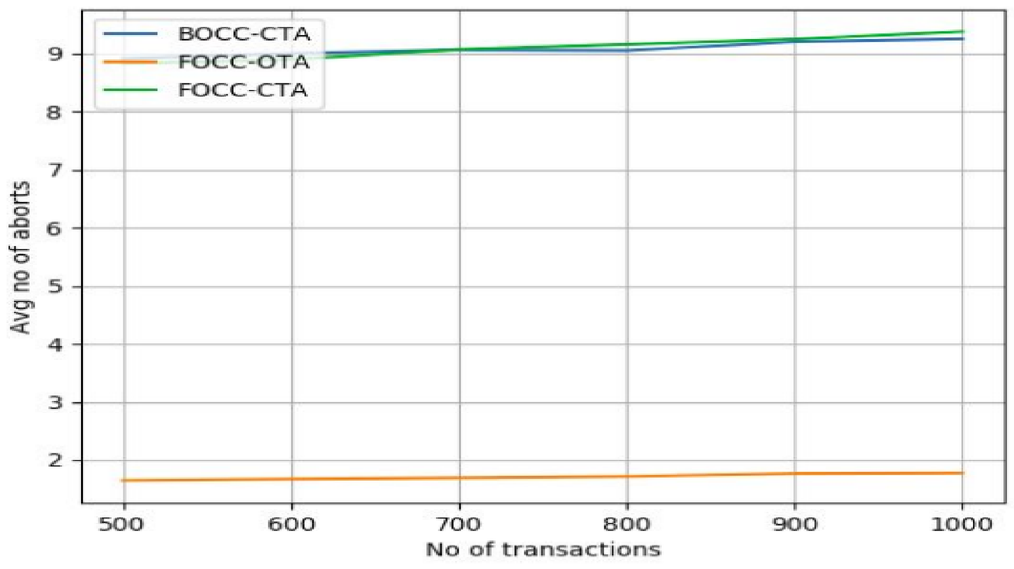
<u>EXPECTED COMPARISON</u>:

According to my understanding between FOOC_OTA, FOCC_CTA and BOCC

FOCC_OTA will give the best performance . And BOCC will give the worst performance.The Average commit times for the CTA algorithms are less than that for the ota it has to check for all the intersection of the write set with many other transaction readset’s, and abort the other transactions in OTA. But in CTA if we find at least one intersection of its write set with any read set of other transactions we can abort the current transactions.

<u>EXPECTED GRAPH</u> :


BOCC-CTA
FOCC-OTA
FOCC-CTA
Avg commit time (millisec)
No of transactions


BOCC-CTA
FOCC-OTA
FOCC-CTA
Avg no of aborts
No of transactions