Programming assignment 1: Implementing BOCC and FOCC algorithms

Goal:
The goal of this assignment is to implement the BOCC and FOCC algorithms studied in class in C++.

Implementation Details:

The implementation of both BOCC and FOCC are similar. The major difference lies in the tryCommit function. In order to implement BOCC, I created a class BOCC, which has private member id and public members `begin_trans`(), `read(i, x, l)`, `write(i, x, l)`, `tryC(i)`.

How I handled tryCommit function in BOCC?

In BOCC, validation of a transaction tj:

Compare tj to all previously committed transaction ti and accept if ti has ended before tj has started or RS(tj) intersection WS(ti) is a null set.

```
for(int i=0; i<n; i++)
      {
            //ti has ended before tj has started
            //Compare to all previously committed transactions
            //Condition when txEndTime of i > start time of id
            //Read set of j intersection write set of i is not null
            //Violation
            if(txStatus[i] == 1)
            {
                  if(txEndTime[i] > txStartTime[id])
                        {
                              for(int j=0; j<m; j++)
                              {
                                    if(readSet[id][j] == writeSet[i][j]
&& writeSet[i][j] ==1)
                                          toAbort = 1;
                              }
                        }
            }
      }
```

I keep a counter toAbort and abort only in case when it is set to 1. Instead of checking A union B, I check for (Not A) intersection (Not B).

How I handled tryCommit function in FOCC?

In FOCC, validation of a transaction tj: Compare tj to all concurrently active transaction ti and accept if WS(tj) intersection RS(ti) is a null set.

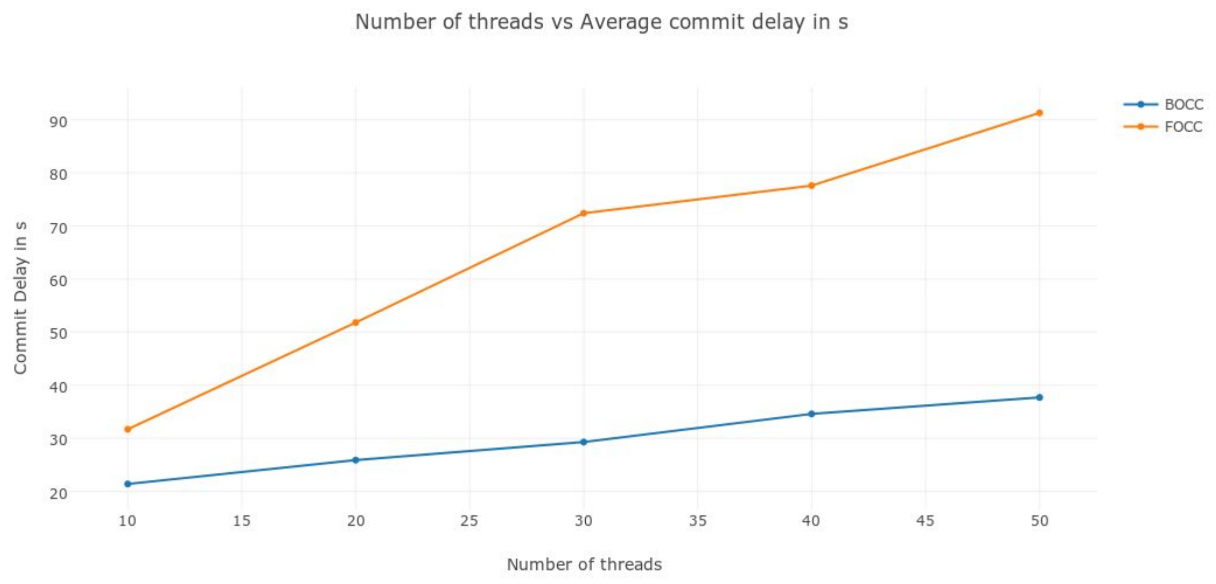
```
for(int i=0; i<n; i++)
     {
          //ti has ended before tj has started
          //Compare to all concurrently active transactions
          if(txStatus[i] == 2) // = 2 => its live
          {
               for(int j=0; j<m; j++)
                    {
                         if(writeSet[id][j] == readSet[i][j] &&
readSet[i][j] == 1)
                              toAbort = 1;
                    }
          }
     }
```

I use a variable, txStatus and I set it to 2, in case it is live or active but not yet committed or aborted. I use a read set and write set for every transaction which keeps track of data item read or written by a particular transaction.

I also keep a txStatus array, which keeps track of the status of a particular transaction. In my implementation, 1 means commit, -1 means abort and 2 means it is currently active but has not yet committed or aborted.

In BOCC, we look for all previously committed transactions whereas in FOCC, I look for only concurrent active transactions.

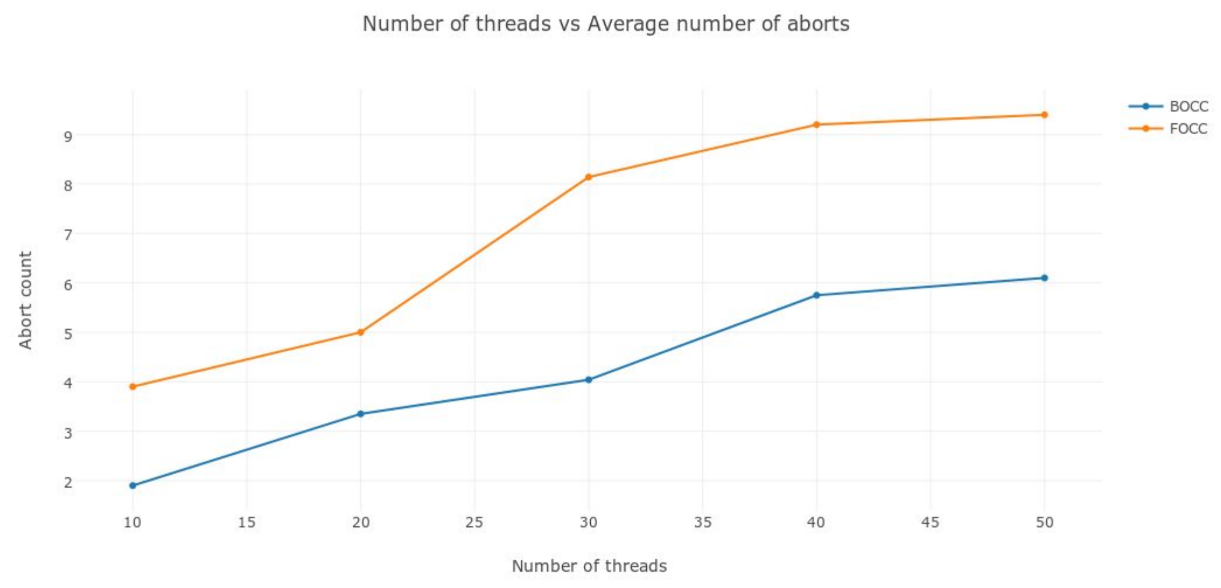
Graphs



Values

Number of threads	BOCC	FOCC
10	21.4	31.7
20	25.9	51.8
30	29.3	72.4
40	34.6	77.6
50	37.7	91.3

This is the plot for average number of threads vs commit delay in s. As the number of threads increases, the commit delay for both BOCC and FOCC increases. The commit delay for FOCC is almost always higher than the commit delay for BOCC.



<u>Values</u>

Number of threads	BOCC	FOCC
10	1.9	3.9
20	3.35	5
30	4.04	8.14
40	5.75	9.2
50	6.1	9.4

This is the plot for average number of threads vs abort count. As the number of threads increases, the number of times a transaction gets aborted before it successfully commits increases for both BOCC and FOCC. The number of aborts in case of FOCC is always higher than that in case of BOCC. FOCC took a longer time to run in most cases.

Conclusion

FOCC has higher commit delay as well as number of abort counts in comparison to BOCC. It is primarily because it looks for all the concurrently active transactions when attempting to commit unlike BOCC, which looks for only previously committed transactons. FOCC takes a longer time to run in most cases.