Programming assignment 2: Implementing BTO and MVTO algorithms

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

Implementation Details:

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

How I handled tryCommit function in BTO?

In BTO, for every data item x, I keep track of the following timestamps:

1) max-r-scheduled(x): the value of the largest timestamp of a read operation on x already sent to the data manager.

2) max-w-scheduled(x): the value of the largest timestamp of a write operation on x already sent to the data manager.

Now, when some pi(x) arrives, then ts(ti) is compared to max-q-scheduled(x) for each operation q that is in conflict with p. If ts(ti) < max-q-scheduled(x) holds, pi(x) is rejected, since it has arrived too late. Otherwise, it is sent to the data manager, and max-p-scheduled(x) is updated to ts(ti) if ts(ti) > max-p-scheduled(x).

```
f(type == 0)
      {
            if(txStartTime[id] < max_w_scheduled[rndId])
            {
                  toAbort = 1;
            }
            else if(txStartTime[id] > max_r_scheduled[rndId])
            {
                  max_r_scheduled[rndId] = txStartTime[id];
            }
      }
      else if(type == 1)
      {
```

```
            if(txStartTime[id] < max_r_scheduled[rndId] ||
txStartTime[id] < max_w_scheduled[rndId])
            {
                  toAbort = 1;
            }
            else if(txStartTime[id] > max_w_scheduled[rndId])
            {
                  max_w_scheduled[rndId] = txStartTime[id];
            }
      }
```

I keep a counter toAbort and abort only in case when it is set to 1.

Implementation details of MVTO:

In MVTO: Each version carries the timestamp ts(ti) of the transaction ti by which it has been created.

1) A step ri(x) is transformed into a step ri(xk), where xk is the version of x that carries the largest timestamp ≤ ts(ti) and was written by tk , k = i.
2) A step w i (x) is processed as follows:

(a) If a step of the form rj (xk) such that ts(tk) < ts(ti) < ts(tj) has already been scheduled, then wi(x) is rejected and ti is aborted,

(b) otherwise, wi (x) is transformed into wi(xi) and executed.

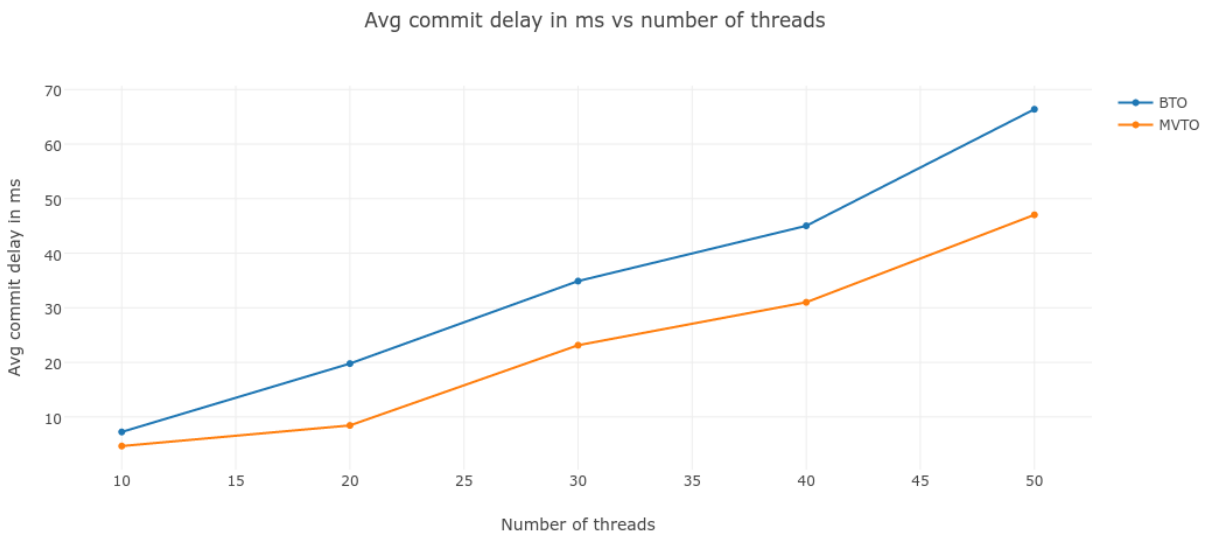
```
            int req = 0, i;

            for(i=0; i<n; i++)
            {
                  if(txStartTime[i] < txStartTime[id])
                  {
                        req = i;
                        break;
                  }
            }
            for(int j=i+1; j<n; j++)
            {
                  if(txStartTime[j] > txStartTime[req] &&
txStartTime[j] < txStartTime[id])
                  {
                        req = j;
                  }
```

```
}
if(req != id)
      {
            //cout << "req is " << req;
            readSet[id][x] = req;
      }
```

This is the changes made to satisfy condition 1 mentioned.

Graphs

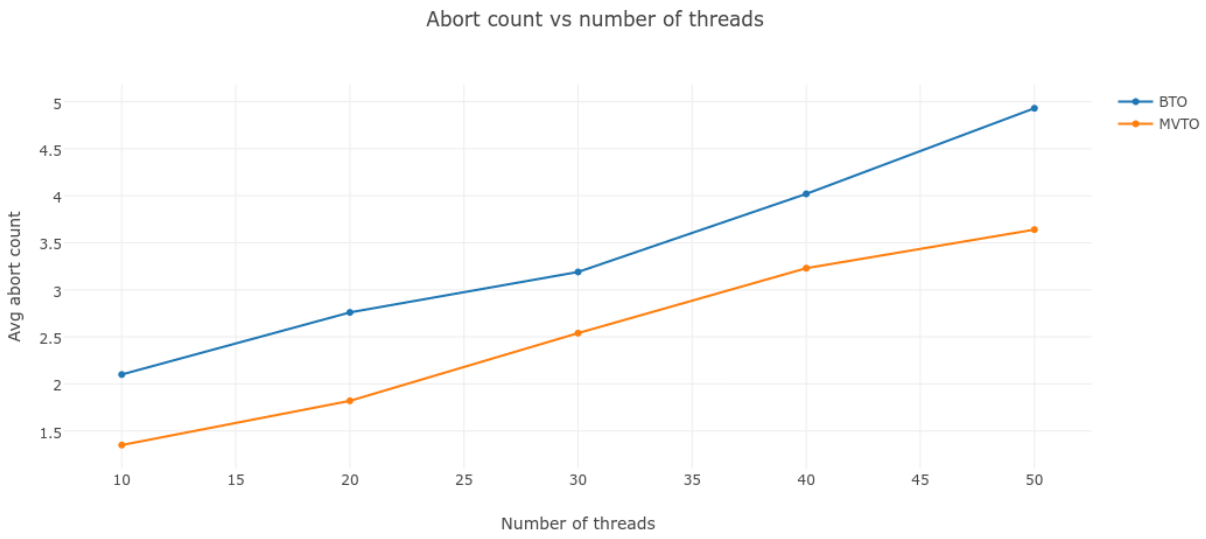


Values

Number of threads	BTO	MVTO
10	7.2434	4.6564
20	19.7719	8.4376
30	34.8981	23.1367
40	45.025	23.1367
50	66.3874	47.038

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 BTO and MVTO increases. The commit delay for BTO is almost always higher than the commit delay for MVTO.

Graph



Values

Number of threads	BTO	MVTO
10	2.1	1.35
20	2.76	1.82
30	3.19	2.54
40	4.02	3.23
50	4.93	3.64

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 BTO and MVTO. The number of aborts in case of BTO is always higher than that in case of MVTO.

Conclusion

MVTO performs better than BTO because in MVTO we take care of versions as well. We assign timestamp to each transaction and perform appropriate transformations for read and write operations. In case of write, we also check for interleaving which makes it more versatile. Whereas, in case of BTO we only check for max-read(x) and max-write(x). Also note that in MVTO we do not consider the case where a transaction can read an item after having written it; if this were allowed, a transaction would have to be able to see a version it has itself produced