Team12 Assignment4 Report

106062137 徐郁閎、106033233 周聖諺、p123786579 王麒銘

1. Implementation

BufferMgr

BufferMgr.java

```
1. Buffer pin(BlockId blk)
2. Buffer pinNew(String fileName, PageFormatter fmtr)
```

縮小 synchronized (bufferPool) 涵蓋的範圍，由於 BufferPoolMgr 已經確保 synchronized，因此只需包住 waitingThreads.add/remove 的部分 (waitingThreads 是 Non Thread-Safe)、以及透過 synchronized 實作排隊機制。

```
1. void unpin(Buffer buff)
2. void flushAll()
3. void flushAllMyBuffers()
4. int available()
5. void unpinAll(Transaction tx)
6. void repin()
```

由於這些 Method 內部做的操作為 Thread-Safe，且內部呼叫的 BufferPoolMgr的 Method 已經確保為 synchronized，因此直接刪除這些 Method 中 的synchronized、bufferPool.notifyAll()。

BufferPoolMgr & Buffer

BufferPoolMgr.java

```
1. Buffer pin(BlockId blk)
2. Buffer pinNew(String fileName, PageFormatter fmtr)
```

刪除原本的 synchronized。首先，為了避免同個 blk 同時要求空的 buffer 而造成的 double buffering 的問題，我們將傳入的 blk 做 hash，並依預設的 buckets數量取餘數，synchronized 該 Object。此外，為了避免 Thread 同時對同個 buffer 做操作，針對每個 buffer 都宣告一個 lock，需要取的該 lock 才能將自己的 block swap 進或保留在該 buffer 中。

Buffer.java

```
1. void pin()
2. void unpin()
3. boolean isPinned()
```

避免同時改 buffer 的 pin 值，將其宣告成 AtomicInteger，使用 pins.incrementAndGet()、pins.decrementAndGet()、pins.get() 等對 pin 值做操作。BufferPoolMgr 中的 numAvailable 也是同個改法。

FileMg & BlockIdr & Page & (JavaNio/JaydioDirect) ByteBuffer

FileMg.java

```
1. IoChannel getFileChannel(String fileName)
```

新增 CocurrentMap，每個 filename 對應一個 ReentrantReadWriteLock，每次被呼叫 getFileChannel Method 的時候，就會到 CocurrentMap 檢查是否有對應的 ReentrantReadWriteLock 在 map 裡面，若無，則新增一個 ReentrantReadWriteLock 並回傳 lock，若有，則直接回傳 lock。

FileMg.java

```
1. void write(BlockId blk, IoBuffer buffer)
2. BlockId append(String fileName, IoBuffer buffer)
```

把在 CocurrentMap 回傳的 lock 鎖 writeLock 後，才呼叫 fileChannel 的 write 與 append。

FileMg.java

```
1. void read(BlockId blk, IoBuffer buffer)
2. long size(String fileName)
```

把在 cocurrentMap 回傳的 lock 鎖 readLock，允許多個thread同時做讀取的動作。

BlockId.java

```
1. int hashCode()
```

為了避免每次呼叫 hashCode() 都算一次，於是在 BlockId 新增 hash 變數，儲存該 block 的 hash 值，並只在 Constructor 計算一次。

Page.java

```
1. void read(BlockId blk)
2. void write(BlockId blk)
3. BlockId append(String fileName)
```

由於已在 fileMgr 處理 concurrency 問題，因此直接刪除此處的 synchronized。

Page.java

```
Constant getVal(int offset, Type type)
void setVal(int offset, Constant val)
```

刪除此處的 synchronized，而將平行化的處裡做在更底層的 (JavaNio/JaydioDirect)ByteBuffer 中。

JavaNioByteBuffer.java / JaydioDirectByteBuffer.java

```
IoBuffer get(int position, byte[] dst)
IoBuffer put(int position, byte[] src)
void clear()
void rewind()
void close()
```

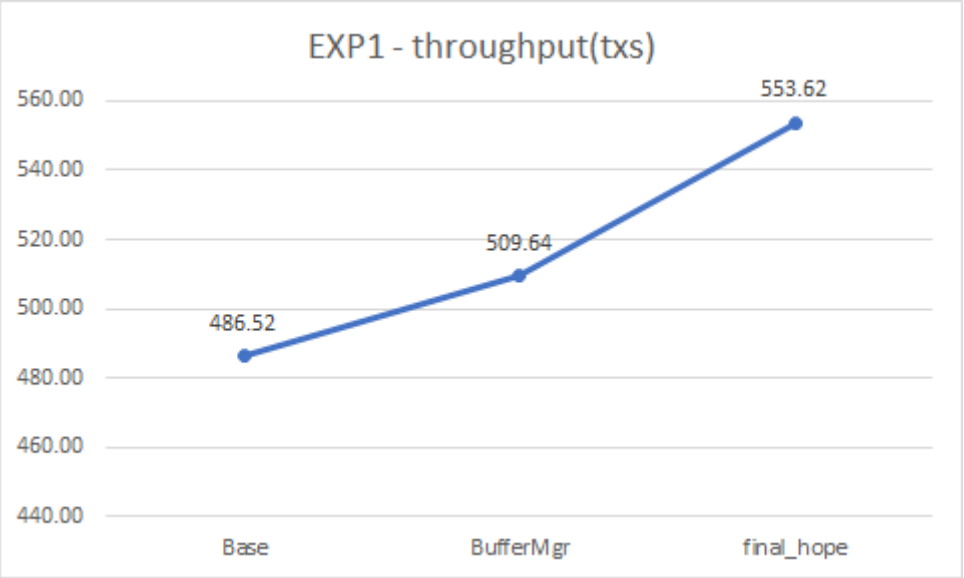
增加 synchronized 確保 Thread-Safe。

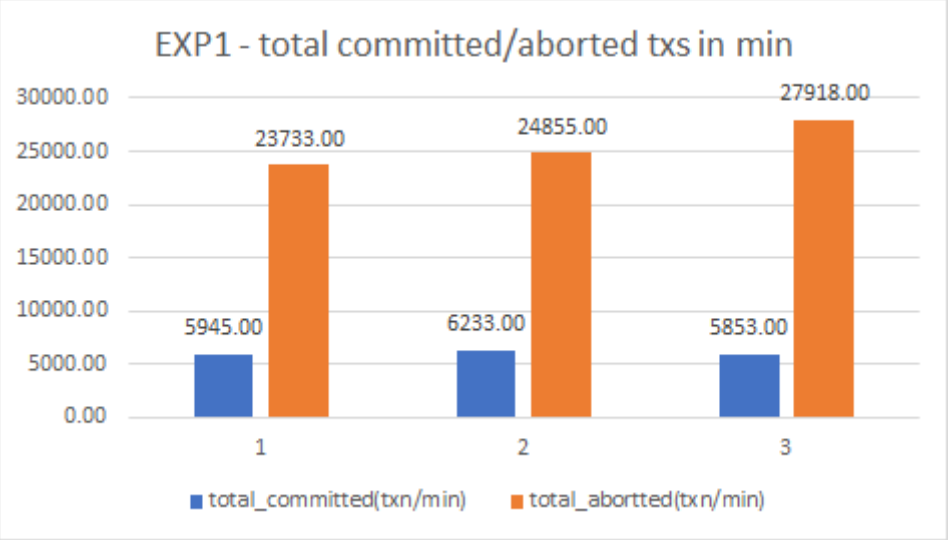
2. Environment

- CPU：Intel® Core™ i5-8250U Processor (6M Cache, up to 3.40 GHz)
- Operating Systems：Windows 10 Home
- RAM：8GB
- Disk：SSD 256 GB

3. Experiment

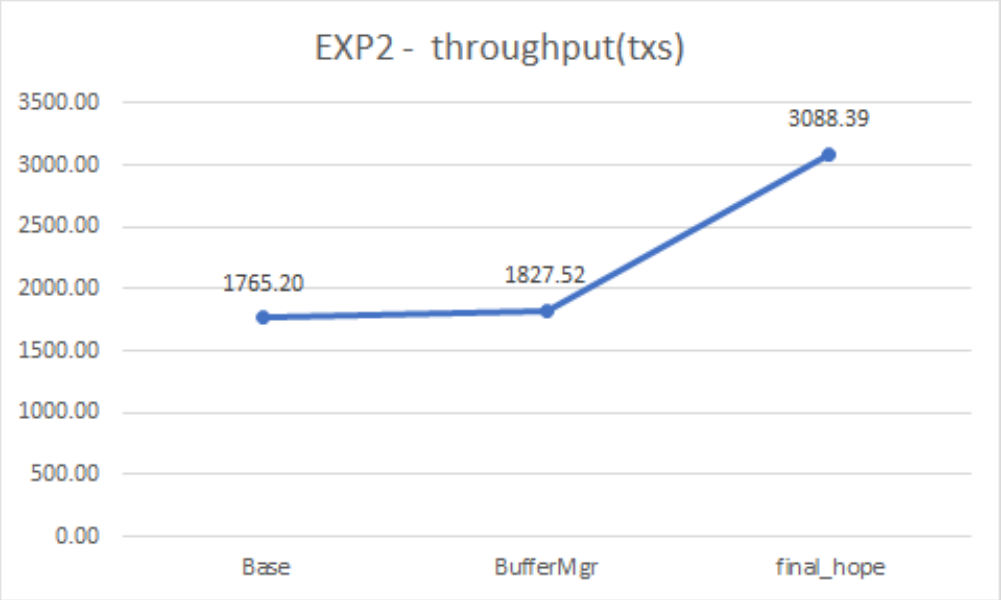
- EXP1 - TPCC Default
  - CONNECTION_MODE=2
  - BENCH_TYPE=2
  - NUM_RTES=4

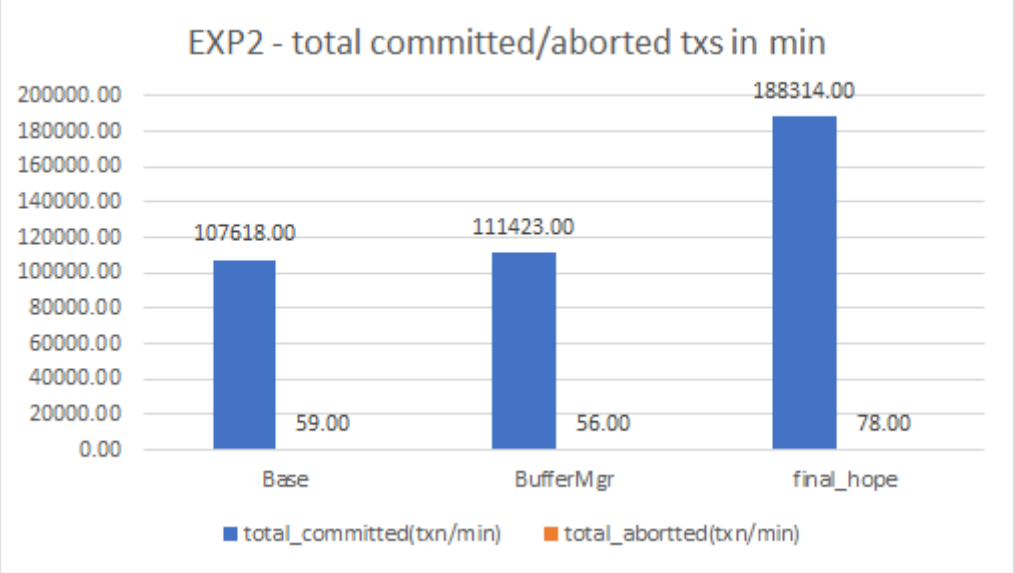




由折線圖可以看出，當我們優化 BufferMgr 之後，throughput 有一定程度的提升 throughput 提升約 13% 。

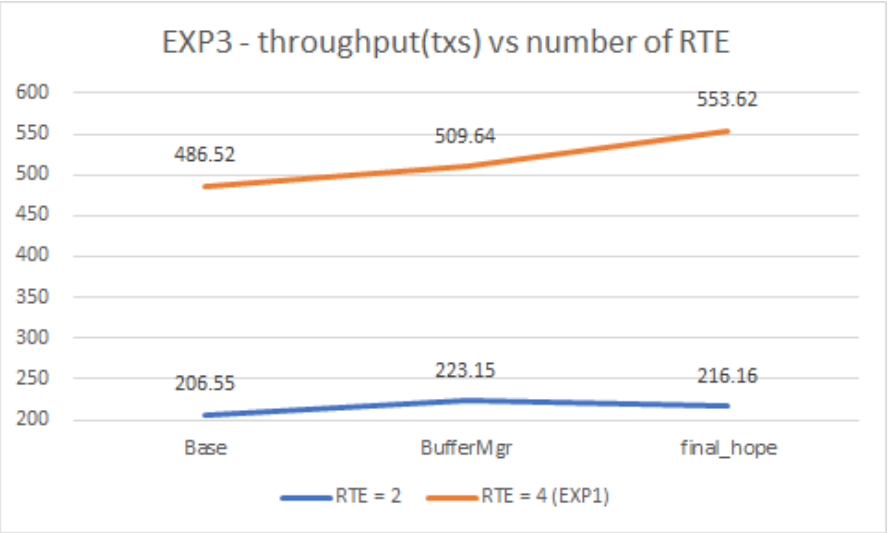
- EXP2 - MICRO Default
  - CONNECTION_MODE=2
  - BENCH_TYPE=2
  - NUM_RTES=4
  - WRITE_RATIO_IN_RW_TX=0.5

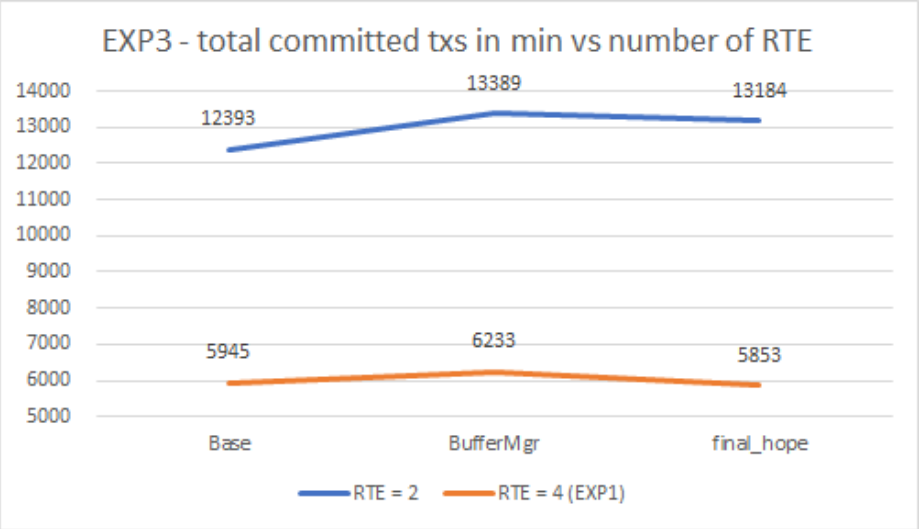


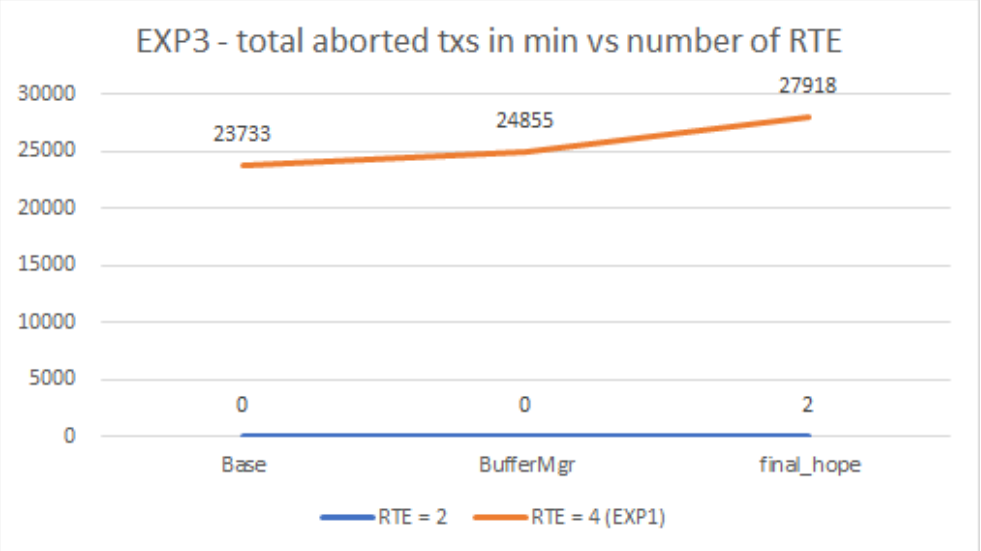


在本實驗中，很明顯可以看出 final_hope 的 throughtput 與 total committed 皆提升了1.74倍。而 abortted transaction(txn/min) 相較於TPCC，則下降很多。

- EXP3 - TPCC RTE=2
  - CONNECTION_MODE=2
  - BENCH_TYPE=2
  - NUM_RTES=2 / 4 (EXP1)

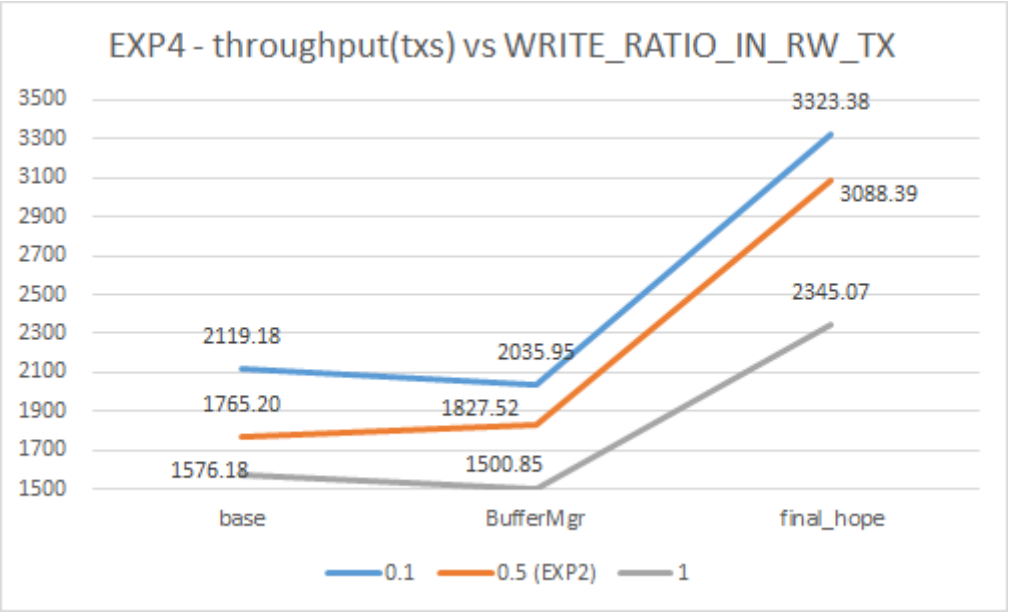


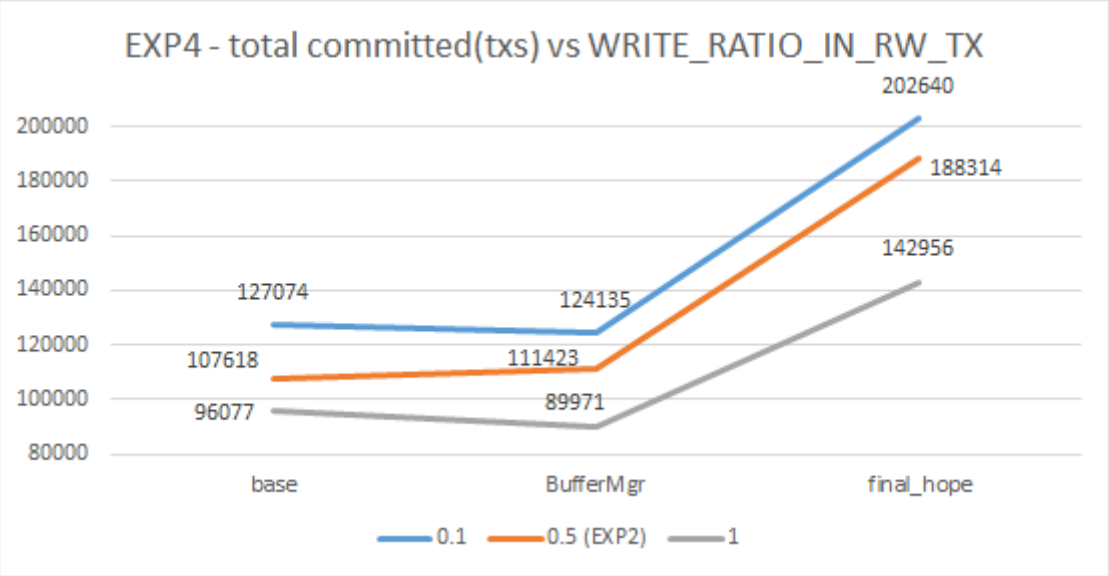


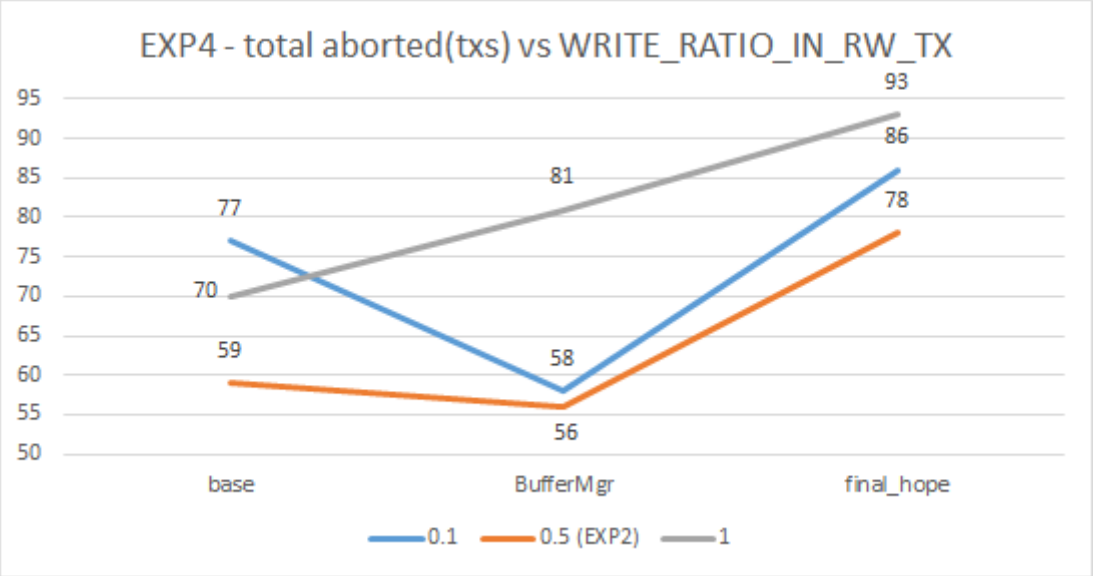


相較於實驗1，同樣都是TPCC，因為 RTE 數量減少，aborted transaction 大幅下降，使得 commited transaction 大幅上升。然而 throughtput 卻大幅下降，可推測當 RTE=4 時，其實對於 DB 而言是過載的狀況，會導致大量 deadlock。

- EXP4 - MICRO WRITE_RATIO_IN_RW_TX
  - CONNECTION_MODE=2
  - BENCH_TYPE=2
  - NUM_RTES=4
  - WRITE_RATIO_IN_RW_TX=0.1 / 0.5(EXP2) / 1







對比EXP2（WRITE_RATIO_IN_RW_TX=0.5），可預期地，所有版本都因為write比例減少而performance上升，包含throughput、committed的數量（aborted的數量趨勢要不明顯）。