Team24 Assignment 5 Report

106034061 曾靖渝
106062116 馮謙
106062216 黃晨

1. Environment

- 

2. Implemetaion:

- LockTable
    - 首先在Locker class中新增 cLocker，以及相關的 method cLock(), cLocked() , hasCLock() and cLockable() .....等等。
    - 根據compatibility table，修改與lock相容性有關的部分，包括avoidDeadlock(), Lockable()。

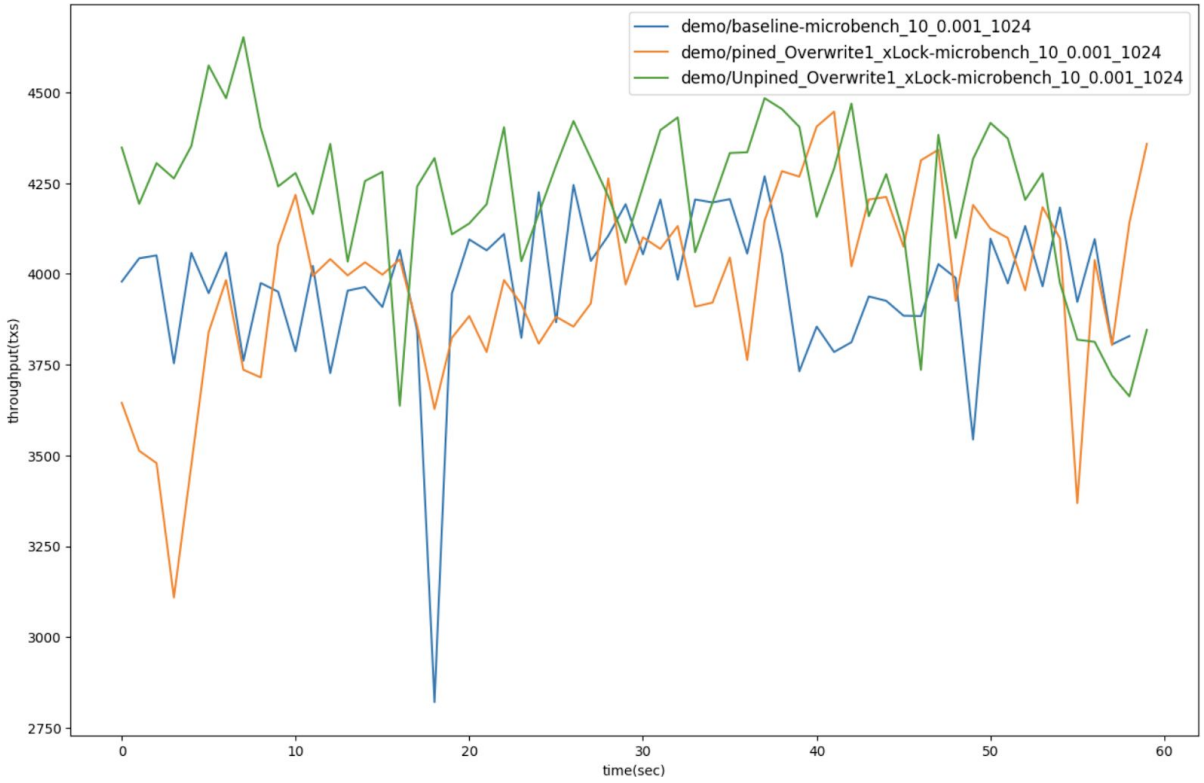
- RecordPage
    - setVal(): 原本這個 method會先拿這個 record 的 xLock, 將 log 寫到 logMgr，再將 val 直接寫到 buffer當中。而根據 spec 要求，我們寫完 log 之後，將 setVal 需要的資訊，包括 lsn, offset, val, blk, currentSlot 打包成一個新的class object，先暫時存入 Transection 的 private workspace 中。
    - getVal(): 根據 spec 要求，我們要先判斷讀取的資料是否曾經被同一個 transection 改寫過。如果有，則需要到 private workspace 中讀取最新的改寫值。若沒有，則照原本的作法直接從 buffer 中讀取。
- Transection
    - New class WriteInfo: 這個 class 是我們為了存取上述 setVal() 所需的資料而創建的。其中我們也實作了 equals()，是為了 ArrayList() 中的 method 所需。
    - InfoList: 這是我們創建的 private workspace，其資料結構是一個包裝 WriteInfo 的 ArrayList。這個 list 記錄了每一筆 setVal() 所需的資料，等到 WriteBack() 時再寫到 buffer 中。
    - addWriteInfo(): 這個 method 會在 recordPage setVal() 被呼叫，其功能是將這筆資料存到一個新的 WriteInfo object 中，並存放到 InfoList 中。
    - WriteBack(): 我們本來設計在 transection commit() 時，呼叫這個method，一次將所有被存到 WrtieInfo 中的 setVal() 寫到 buffer 當中。然而，我們遇到一個問題，就是一整個 transection 的 WriteInfo 太多，會造成在 load testbed 時，因為 private space 過大，需要不斷 swap page，而異常緩慢。因此，我們改在 RecordFile.close() 時，就呼叫 WriteBack()，並在這個 method 最後 call InfoList.clear()。以此減少 Private Space 的空間大小。
    - ReadModified(): 這個 method 會從 InfoList 尾端開始往前讀取，這麼做是因為我們認為 database modification 會有 temporal locality，因此最近被修改的 block 有較高的機率再被修改。而我們會比較 block, slotId, offset 來判斷讀取的和之前修改的是不是同一份資料。若沒有找到同一筆資料，則會回傳 null，讓 RecordPage 的 getVal() 判斷這筆資料沒有被該 transection 修改過。
- SerializableConcurrencyMgr
    - writeRecord:WriteBack()時將xLock 轉換成(Convert) cLock;

- modifyBlock(): 這個 method 只會在 FileHeaderPage 中的 setVal() 被呼叫。原本是拿 xlock，但為了讓 FileHeaderPage 在被改寫時不能同時被讀取，因此我們將其換成 clock。

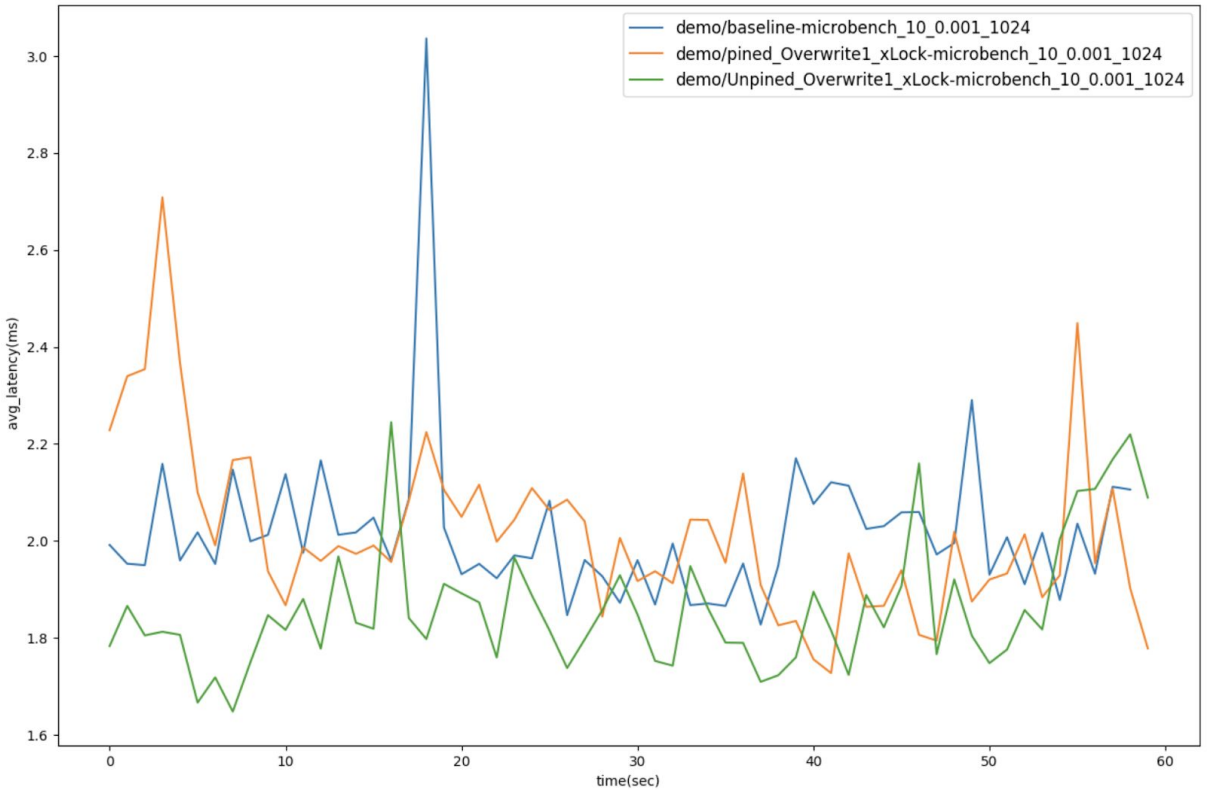
3. Optimization:

- Spatial Locality:

  我們猜測 database access 會有 spatial locality，因此當我們在進 writeback 的時候，我們並不會每寫一個 record 就 pin 一個 buffer 然後再 unpin 掉，而是判斷下一個 record 所用的 buffer 是否和他一樣，如果不同，那再做 unpin。這地方有個小細節，因為 pin buffer 時，重複的東西會做 pinCount++，所以我們要確保同一個 buffer 只 pin 過一次，因此我們這裡加了一個 flag 做判斷。
  - 由下圖可見：(最後面得數字為參數設定，於後面實驗會用到)
    - Unpined ： 判斷前後buffer是否相同，減少pin buffer的數量。
    - pined：不做判斷，在寫入時直接pin buffer，且寫完unpin buffer。
    - baseline: 原助教給的2PL版本
  - 修改前後比較:
  - Thoroughput:



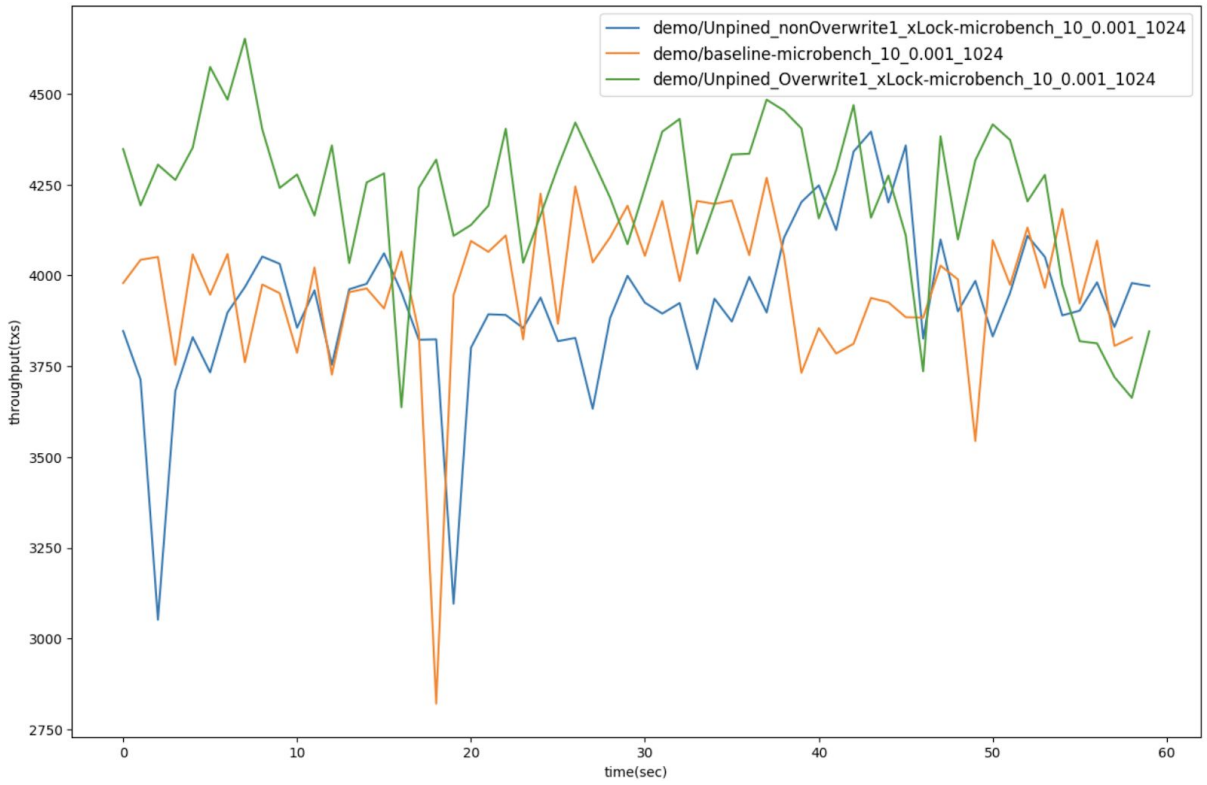
  - Latancy



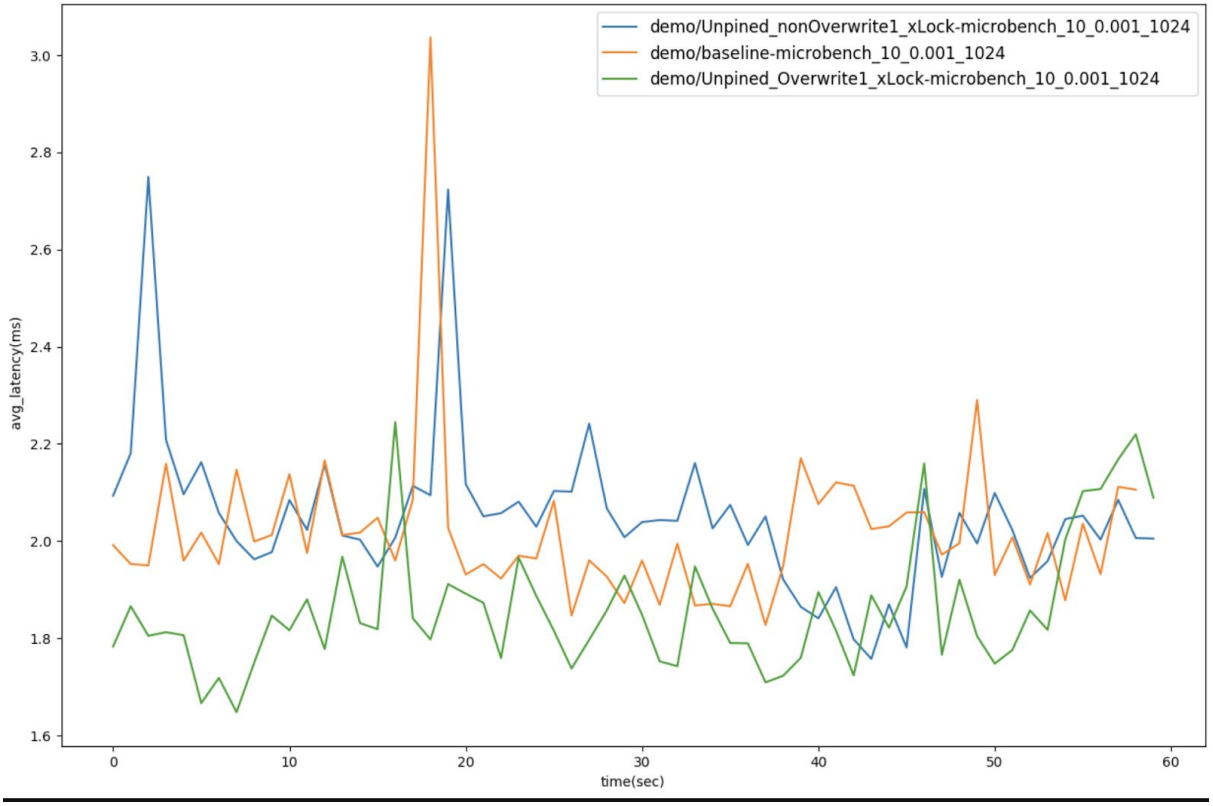
Spatial Locality	SpeedUp (優化後/優化前)
Throughput	253217/238437 = 6.1%
Ave_latency	0%

- Overwrite:

  我們這邊做了一個優化，在每次 addWriteInfo 時，先用 InfoList.contain() 判斷是否有相同 block, slot, offset 的 WriteInfo。如果有，就用這次的 value overwrite 舊的 value 。如此以來，在 WriteBack 的時候，就不會重複寫同一份資料了。
  - 由下圖可見：(最後面得數字為參數設定，於後面實驗會用到)
    - Overwrite ： 有使用contian判斷且覆蓋
    - nonOverwrite：不覆蓋重複的資料
    - baseline: 原助教給的2PL版本
  - 修改前後比較:
  - Throughput

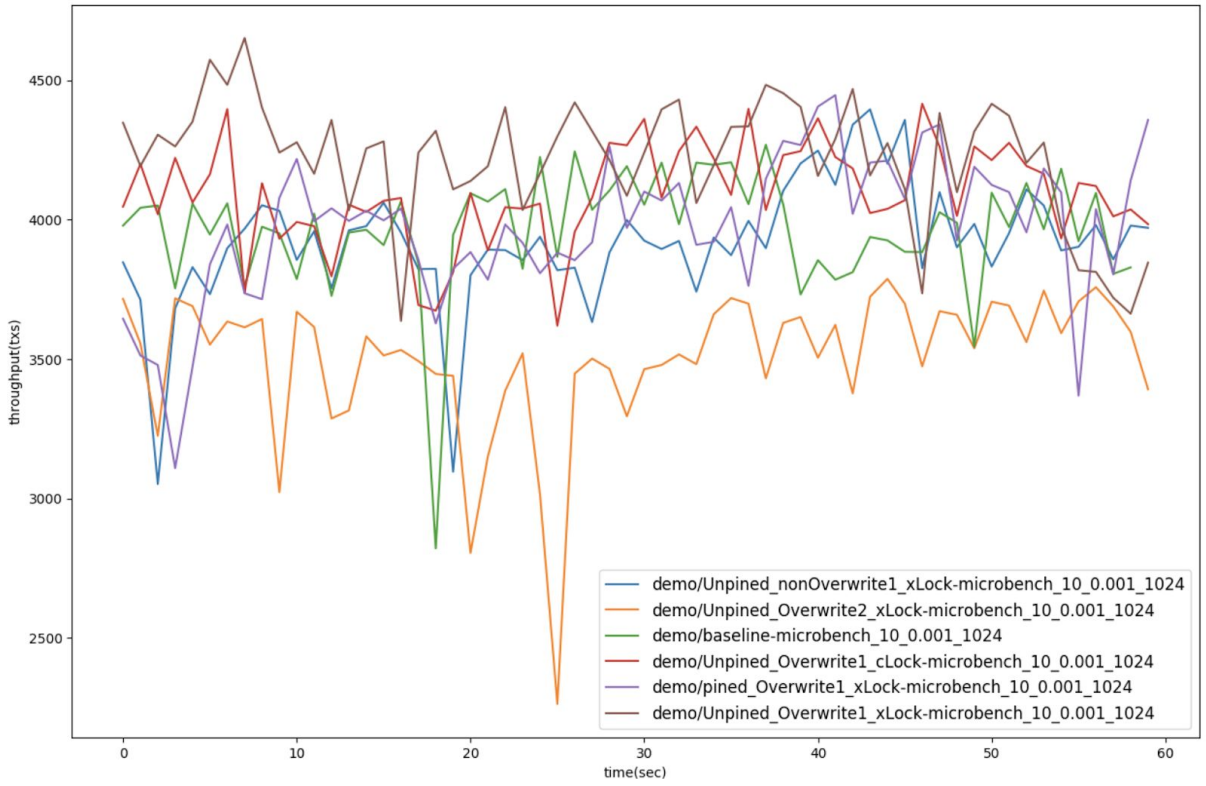

demo/Unpined_nonOverwrite1_xLock-microbench_10_0.001_1024
demo/baseline-microbench_10_0.001_1024
demo/Unpined_Overwrite1_xLock-microbench_10_0.001_1024
throughput(txs)
time(sec)

- Latancy

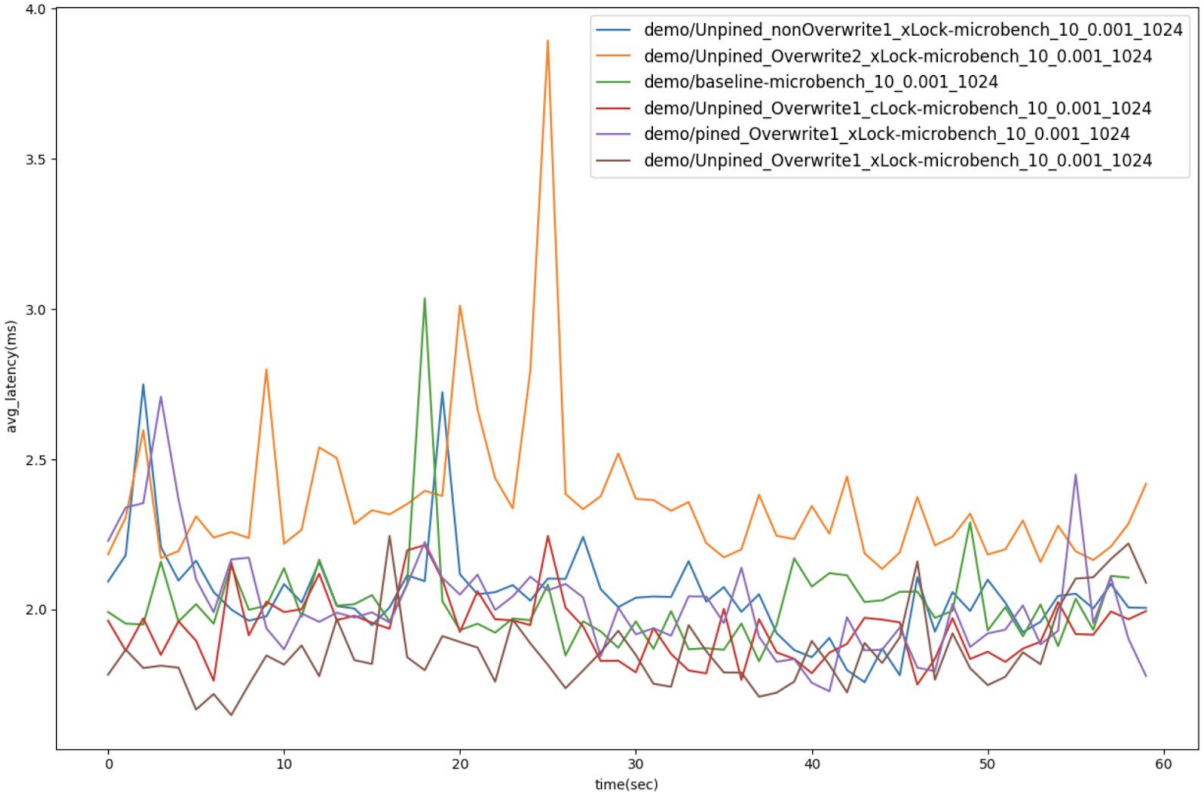

demo/Unpined_nonOverwrite1_xLock-microbench_10_0.001_1024
demo/baseline-microbench_10_0.001_1024
demo/Unpined_Overwrite1_xLock-microbench_10_0.001_1024
avg_latency(ms)
time(sec)

Overwrite	SpeedUp(優化後/優化前)
Throughput	253217/235118 = 7.6%
Ave_latency	0%

- 所有版本對照表
  - 以下是我們嘗試過的所有版本，用兩種overwrite 方式，以及判斷要不要重複pin buffer。最後取最好的版本來做實驗。
  - Throughput



  - Latancy



4. Problem & Solution:

- 問題1：

  當我們把開始跑 loadTestbed 時候，我們發現整個流程會卡住，於是我們開始想問題出在哪裡。

  一開始我們覺得是 deadLock 的問題，因此我們把 deadLock 的時間縮短，但結果還是不變，後來我們仔細的 trace mircobenchmark 的 code 後發現，基本上只會有一個 tx 再跑 loadTestbed，所以照理來說不會有deadLock 的問題。

  後來我們猜測可能是因為 Memory swap 的問題，於是我們把 Monitor 調出來看，發現 CPU 基本上都在閒置，而且 Memory 會在某個容量後卡住。也就是說，當我們把東西存到 private workspace後，整個流程會在 Memory swap 卡住，而可能 Java 或 OS 本身有限制記憶體用量的關係，使得記憶體會在某個上限後卡住。

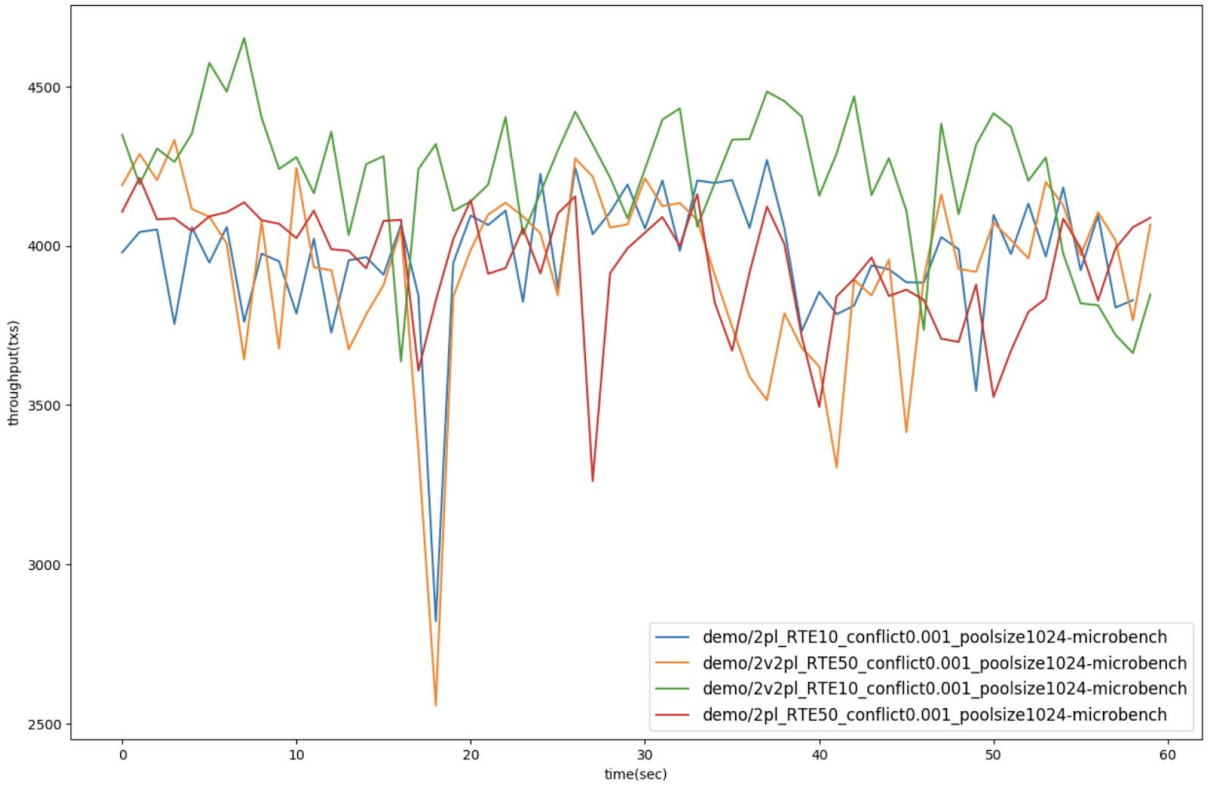
- 解決1：

  發現問題後，我們開始著手解決問題。多虧了助教提醒我們要記得把 private workspace 的記憶體 free 掉，我們也才能知道在哪邊做 In-place write 可能是比較好的地方。

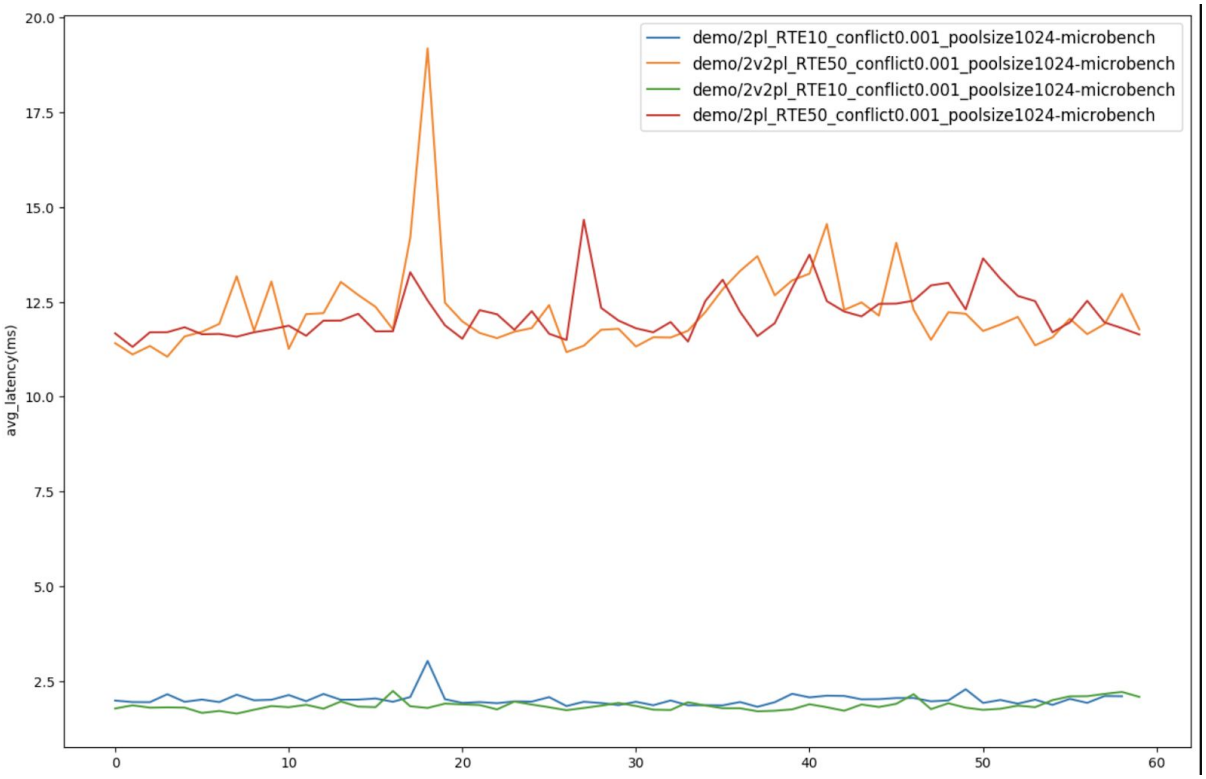
  最後我們是在 RecordFile 的 close() 裡面做 Writeback 的動作。（這也是多虧了助教在 spec 上的 hint）

5. Experiments:

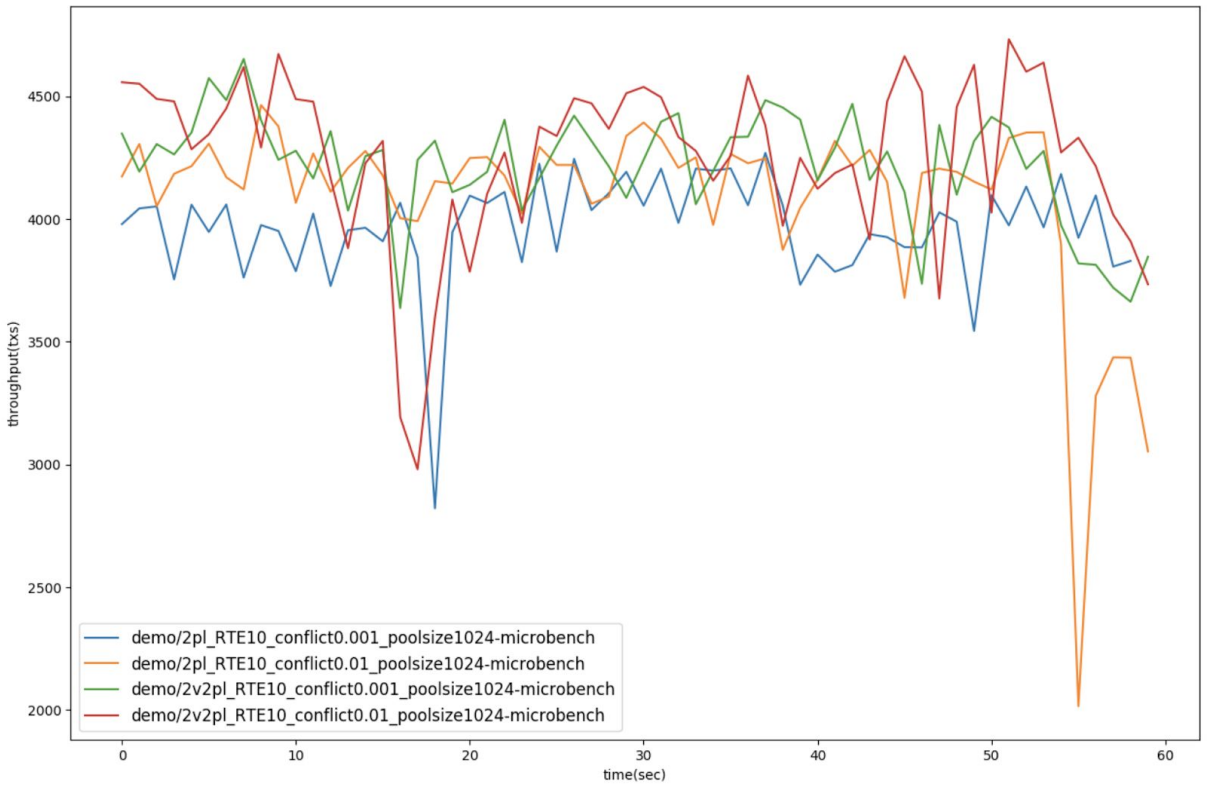
- Comparison of RTE(NUM_RTES: 10 vs.50)
  - MicroBenchmark
  - BufferPoolSize = 1024
  - Hot Conflict rate = 0.001
  - RTEs: 10 vs. 50
  - SpeedUp (RTE = 10)= 253217/238016 = 6.7%
  - SpeedUp (RTE = 50)= 235706/236486 = -0.31%
  - Throughput



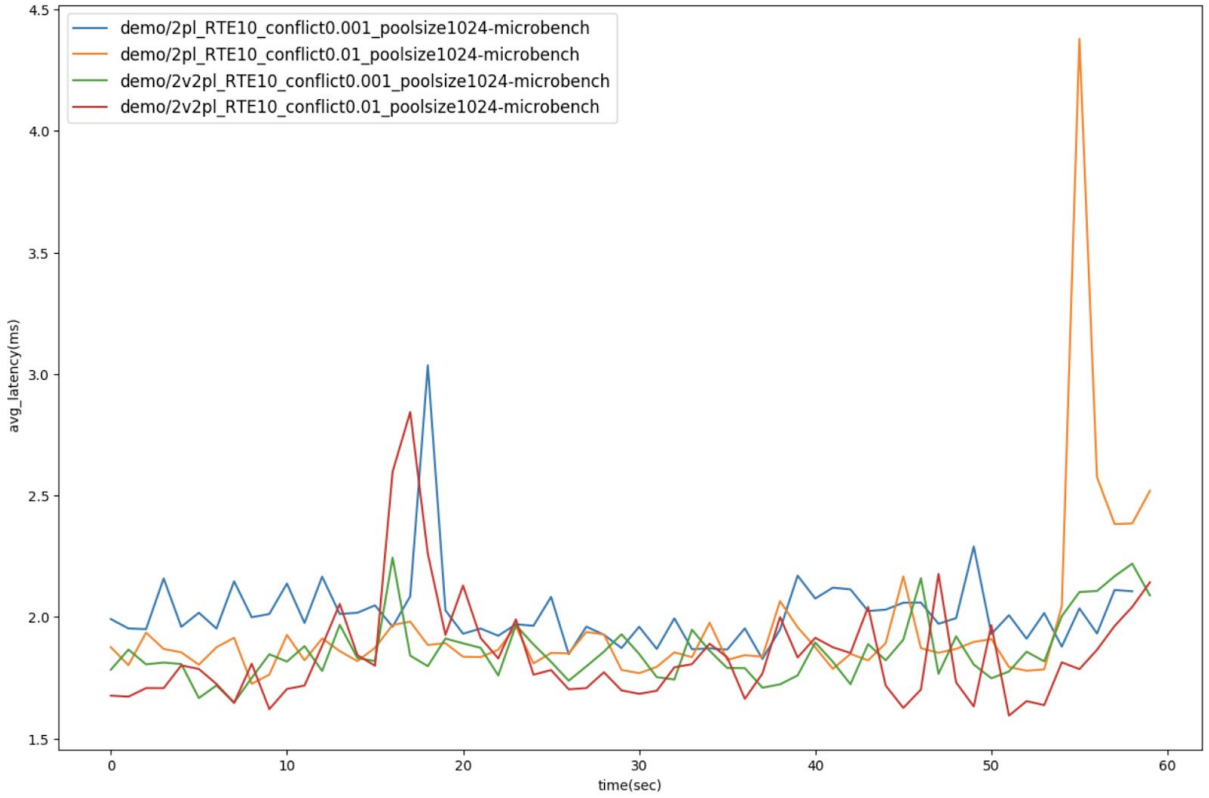
  - Latancy



- Comparison of Conflict Rate(Conflict rate: 0.001vs.0.01)
  - MicroBenchmark
  - BufferPoolSize = 1024
  - Hot Conflict rate: 0.001 vs. 0.01
  - RTEs = 10
  - SpeedUp (conflict rate = 0.01)= 255432/245322 = 4.1%
  - SpeedUp (conflict rate = 0.001)=253217/238016 = 6.7%
  - Thorughput

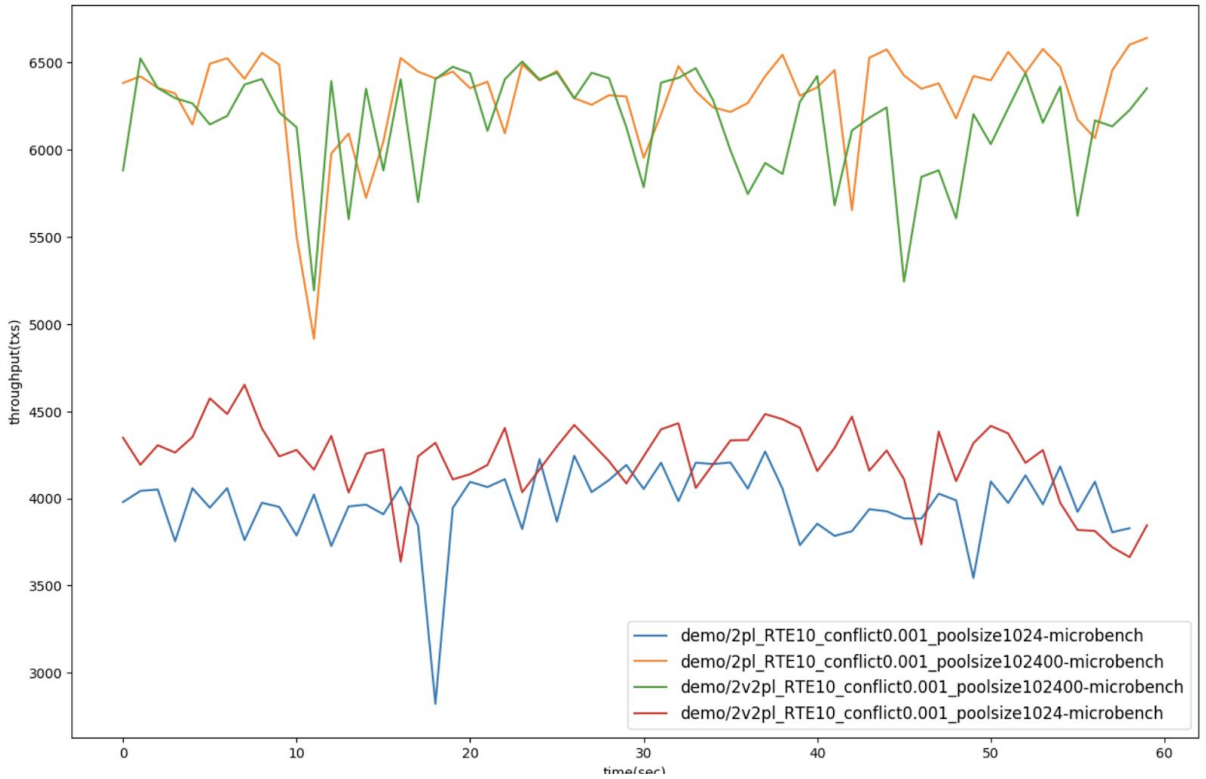


- Latancy

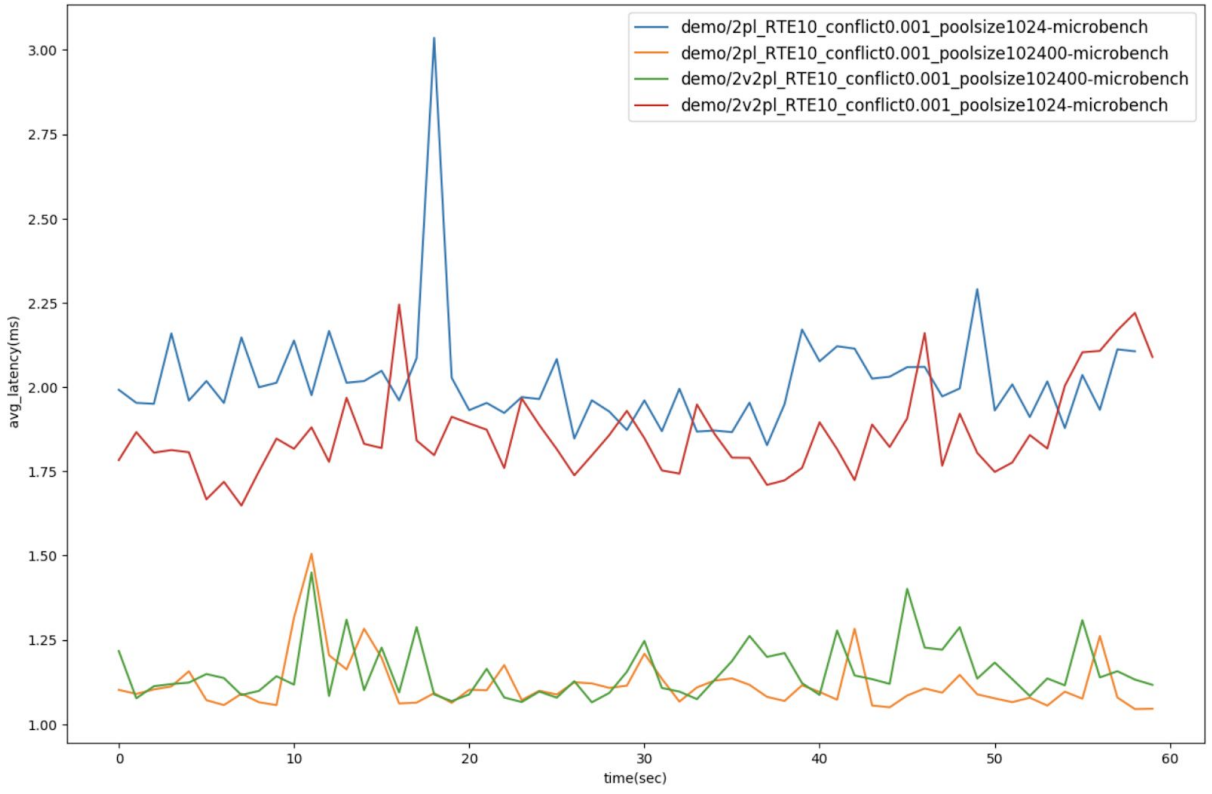


- Comparison of BufferPoolSize(PoolSize 1024 vs 102400)
  - MicroBenchmark
  - BufferPoolSize: 1024 vs. 102400

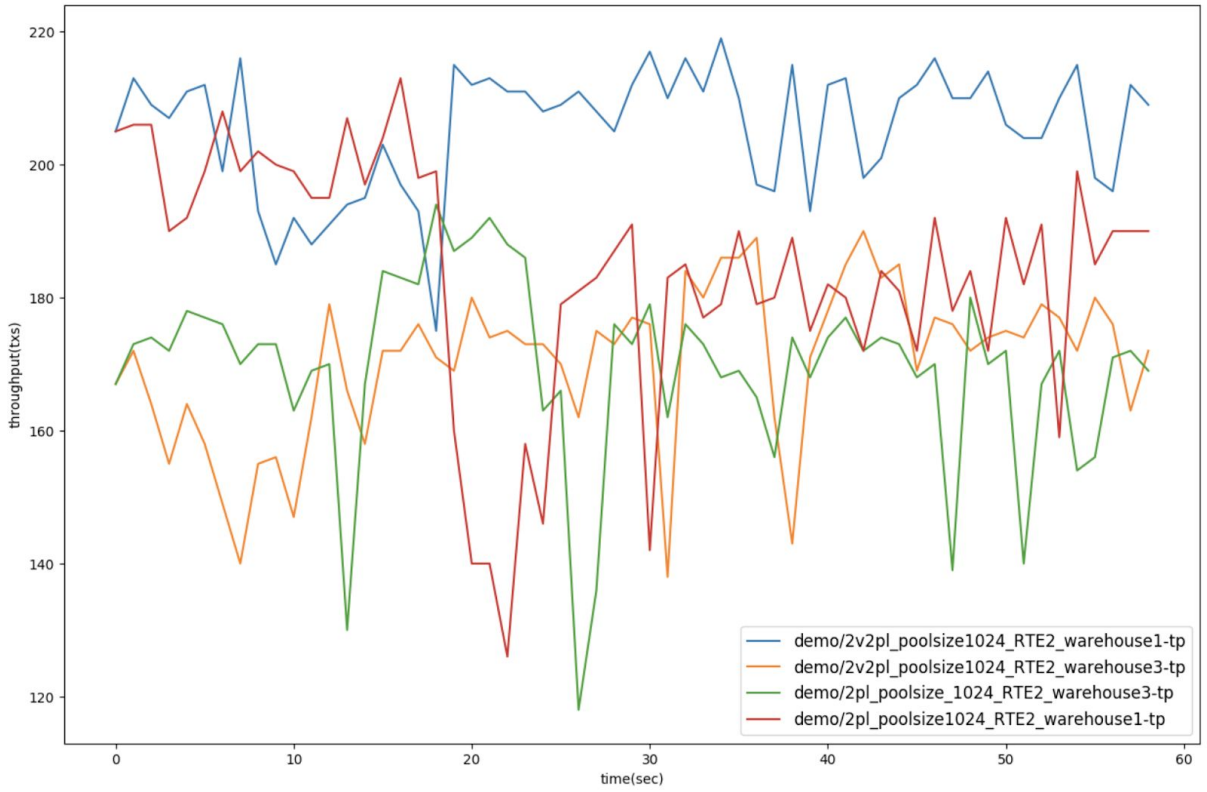
- Hot Conflict rate = 0.001
- RTEs = 10
- SpeedUp (poolsize = 1024)= 253217/238016 = 6.7%
- SpeedUp (poolsize = 102400)=369063/378273 = -2.4%
- Thorughput



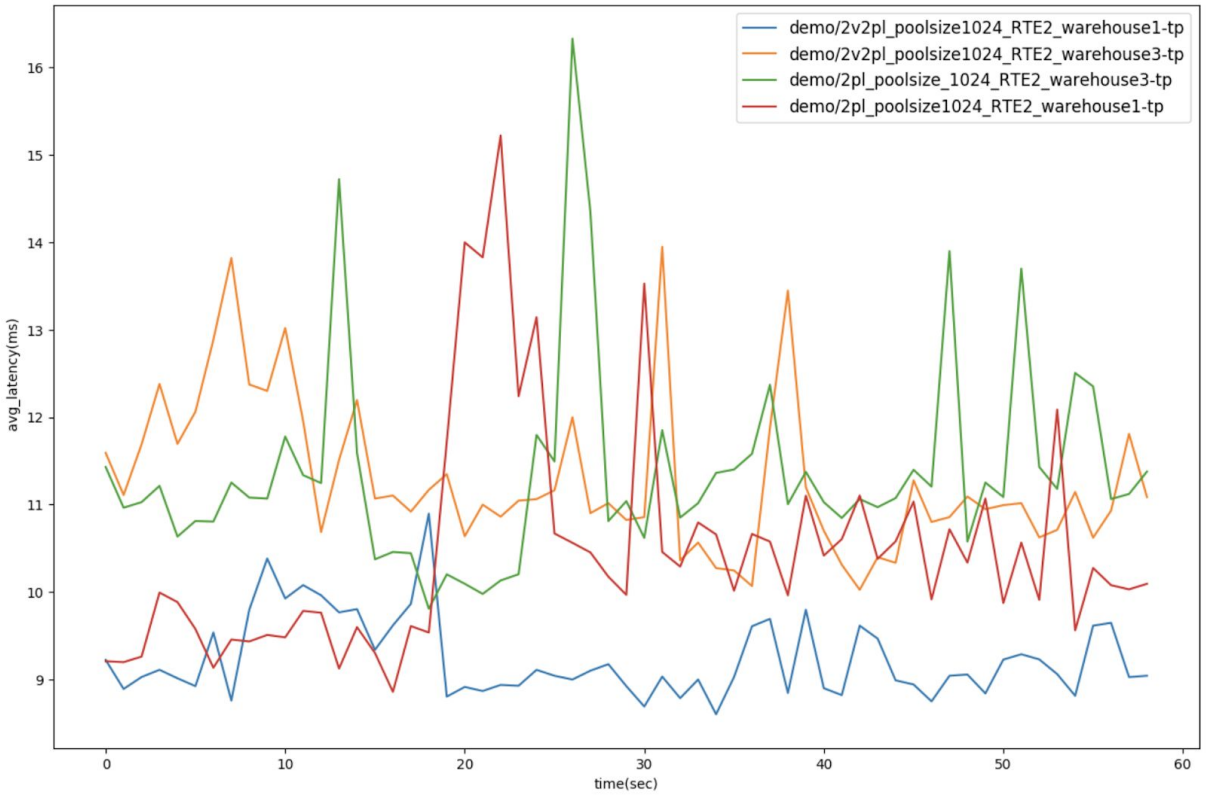
- Latancy



- Comparison of warehouse(NUM_WAREHOUSES: 1 vs. 100)
    - TPCCBenchmark
    - BufferPoolSize = 1024
    - RTEs = 10
    - NUM_WAREHOUSES: 1 vs. 100
    - SpeedUp(warehouse = 1) = 10217/10149 = 0.67%
    - SpeedUp(warehouse = 3) = 12336/11038 = 11.7%
    - Thorughput



    - Latancy



6. Analysis

- Comparison of RTE(NUM_RTES: 10 vs.50)
  - 由圖表可知，

NUM_RTES: 10	SpeedUp(2v2pl / 2pl)
Throughput	6.7%
Ave_latency	0%

NUM_RTES: 50	SpeedUp(2v2pl / 2pl)
Throughput	-0.32%
Ave_latency	0%

- Comparison of Conflict Rate(Conflict rate: 0.001vs.0.01)
  - 由圖表可知，增加conflict rate，2v2pl 與 2pl的差距減少了。

Conflict rate: 0.001	SpeedUp(2v2pl / 2pl)
Throughput	6.7%
Ave_latency	0%

Conflict rate: 0.01	SpeedUp(2v2pl / 2pl)
Throughput	4.1%
Ave_latency	0%

- Comparison of BufferPoolSize(PoolSize: 1024 vs 102400)
  - 由表格中可知，當buffer pool size增加，整體的throughput會增加，但2v2pl的效能會變差，變得比2pl低。

PoolSize: 1024	SpeedUp(2v2pl / 2pl)
Throughput	6.7%
Ave_latency	0%

PoolSize: 102400	SpeedUp(2v2pl / 2pl)
Throughput	-2.4%
Ave_latency	0%

- Comparison of warehouse(NUM_WAREHOUSES: 1 vs. 3)
  - 由表格中可知，當我們增加NUM_WAREHOUSES時，兩種lock方式的差距提升了。

NUM_WAREHOUSES: 1	SpeedUp(2v2pl / 2pl)
Throughput	0.67%
Ave_latency	0%

NUM_WAREHOUSES: 3	SpeedUp(2v2pl / 2pl)
Throughput	11.7%
Ave_latency	-10.1%