Assignment 4 Phase 1 Report

110062229 翁語辰

110081014 程詩柔

110062171 陳彥成

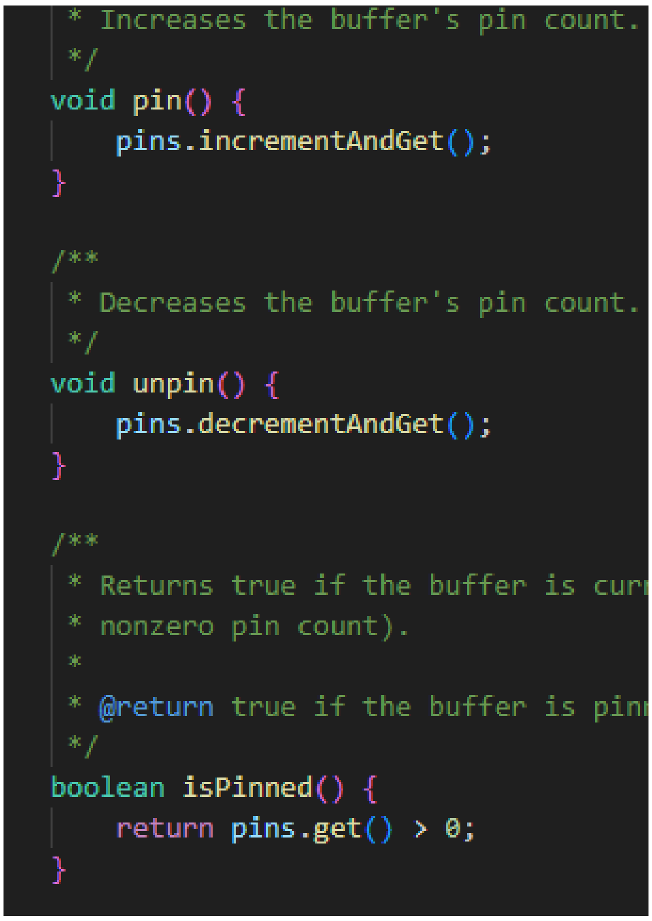
1. Implementation

我們主要分成兩次的優化，一次是buffer文件夾中的優化，另一次是file的，主要優化方為移除不必要的synchronize，由於被synchronize的function 一次只有一個thread能執行，其他thread必須在外等待，因此適當的移除或者更動synchronize位置可以使多個thread能同時執行更多程式，提升整體效能。

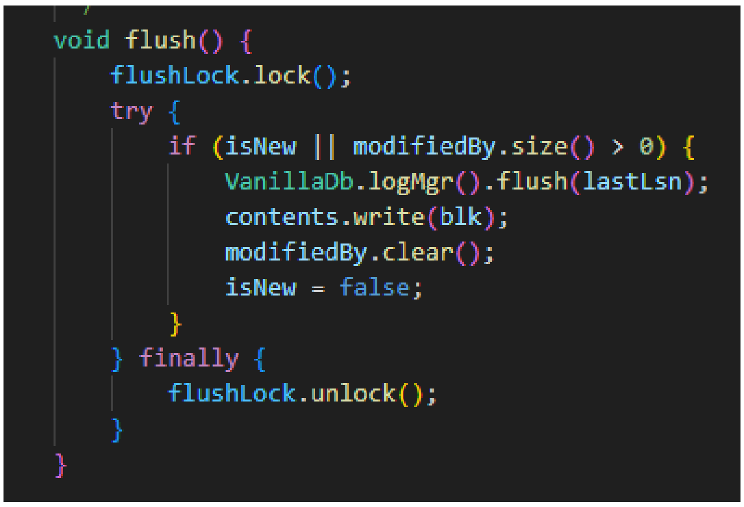
buffer optimization

`Buffer.java`

- 將pins改成java.util.concurrent.atomic.AtomicInteger所提供的實例。就可以把相關函數的synchronized關鍵字移除，以提高效能。flushLock做為ReentranLock的實例，讓flush函數也移除synchronized關鍵字。



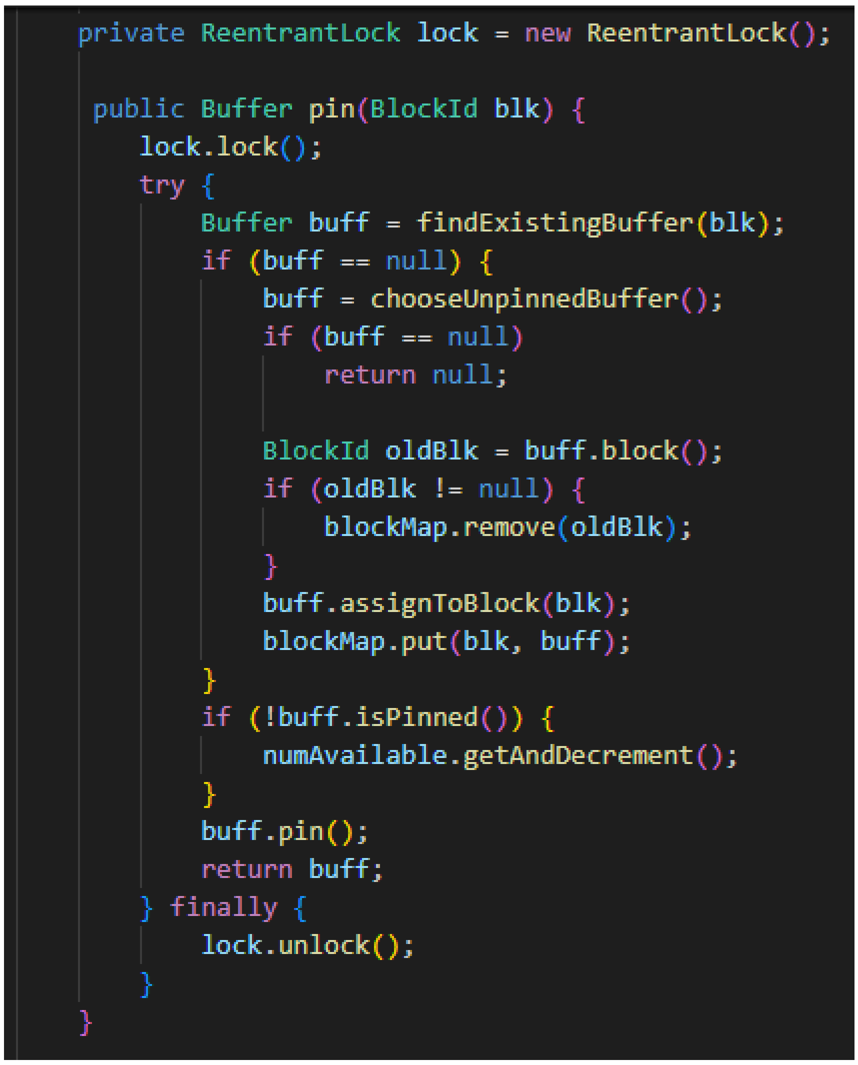
```java
 * Increases the buffer's pin count.
 */
void pin() {
    pins.incrementAndGet();
}

/**
 * Decreases the buffer's pin count.
 */
void unpin() {
    pins.decrementAndGet();
}

/**
 * Returns true if the buffer is curr
 * nonzero pin count).
 *
 * @return true if the buffer is pinn
 */
boolean isPinned() {
    return pins.get() > 0;
}
```



```java
void flush() {
    flushLock.lock();
    try {
        if (isNew || modifiedBy.size() > 0) {
            VanillaDb.logMgr().flush(lastLsn);
            contents.write(blk);
            modifiedBy.clear();
            isNew = false;
        }
    } finally {
        flushLock.unlock();
    }
}
```

BufferPoolMgr.java

- 同樣的方法。概念上都是用更精準的lock取代整個synchronized關鍵字，以提高效率。或是用concurrency中的資料結構來優化。



```java
private ReentrantLock lock = new ReentrantLock();

 public Buffer pin(BlockId blk) {
    lock.lock();
    try {
        Buffer buff = findExistingBuffer(blk);
        if (buff == null) {
            buff = chooseUnpinnedBuffer();
            if (buff == null)
                return null;

            BlockId oldBlk = buff.block();
            if (oldBlk != null) {
                blockMap.remove(oldBlk);
            }
            buff.assignToBlock(blk);
            blockMap.put(blk, buff);
        }
        if (!buff.isPinned()) {
            numAvailable.getAndDecrement();
        }
        buff.pin();
        return buff;
    } finally {
        lock.unlock();
    }
}
```

BufferMgr.java

- 主要更動是在pin方法，把synchronized拆解成多個部分，來達到優化的目的。

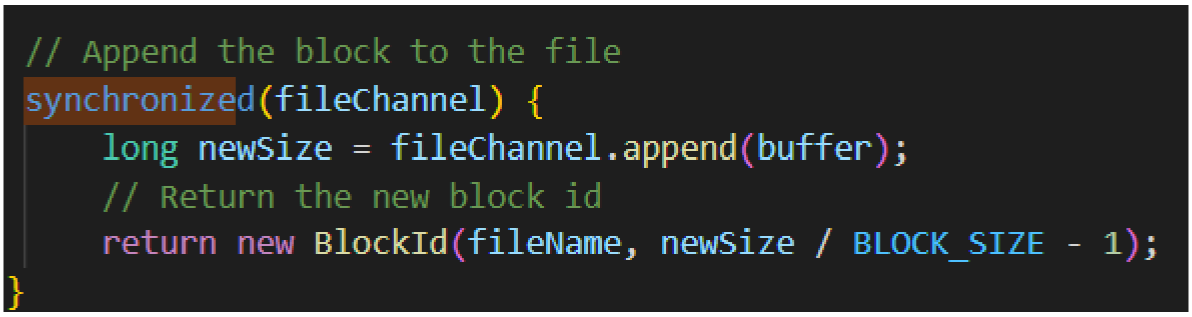
file optimization

`Blockid.java`

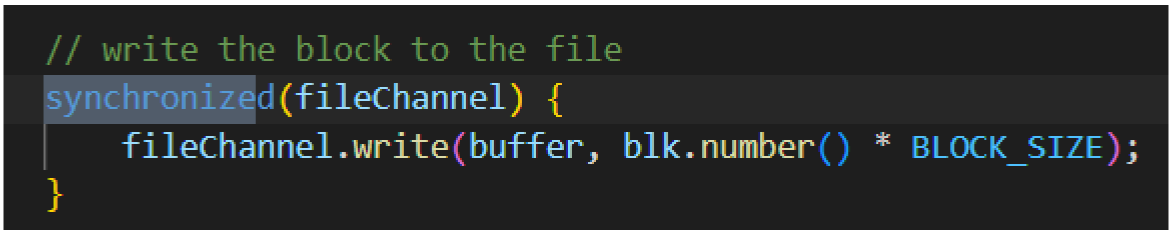
- 為了避免重複的計算，我們將toString()和hashCode()的計算部分在建構函數中調用一次，再存成實例的properties，而後便可以直接返回。不必多次計算。

`FileMgr.java`

- 移除read function的synchronized，因為read只有讀資料，多個thread同時讀取資料
- 將write,append function 的synchronized 移到function內部，因為只有部分會造成race condition



```java
// Append the block to the file
synchronized(fileChannel) {
    long newSize = fileChannel.append(buffer);
    // Return the new block id
    return new BlockId(fileName, newSize / BLOCK_SIZE - 1);
}
```



```java
// write the block to the file
synchronized(fileChannel) {
    fileChannel.write(buffer, blk.number() * BLOCK_SIZE);
}
```

`Page.java`

- 移除 read,write,append 的 synchronized.因為我們已經在FileMgr.java做好synchronized了

2. 各項優化結果比較

實驗環境

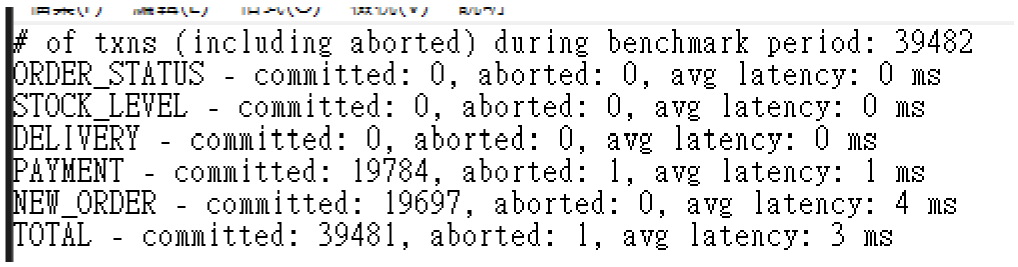
- 使用TPC-Cbenchmark
- CPU: AMD Ryzen 7 5800X 8-Core Processor 3.80 GHz
- RAM: 16.0 GB
- DISK: 512GB SSD
- OS: Windows 10

參數設定(我們發現原本的參數最能表現優化)

- NUM_WAREHOUSES=100
- FREQUENCY_TOTAL=100
- FREQUENCY_NEW_ORDER=50
- FREQUENCY_PAYMENT=50
- FREQUENCY_ORDER_STATUS=0
- FREQUENCY_DELIVERY=0
- FREQUENCY_STOCK_LEVEL=0
- ENABLE_THINK_AND_KEYING_TIME=false

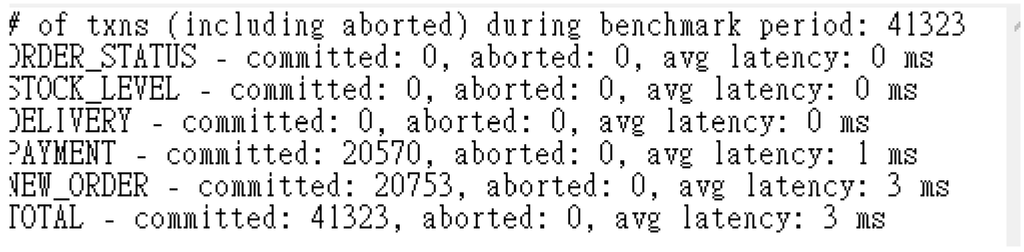
1.階段性優化比較

原始:



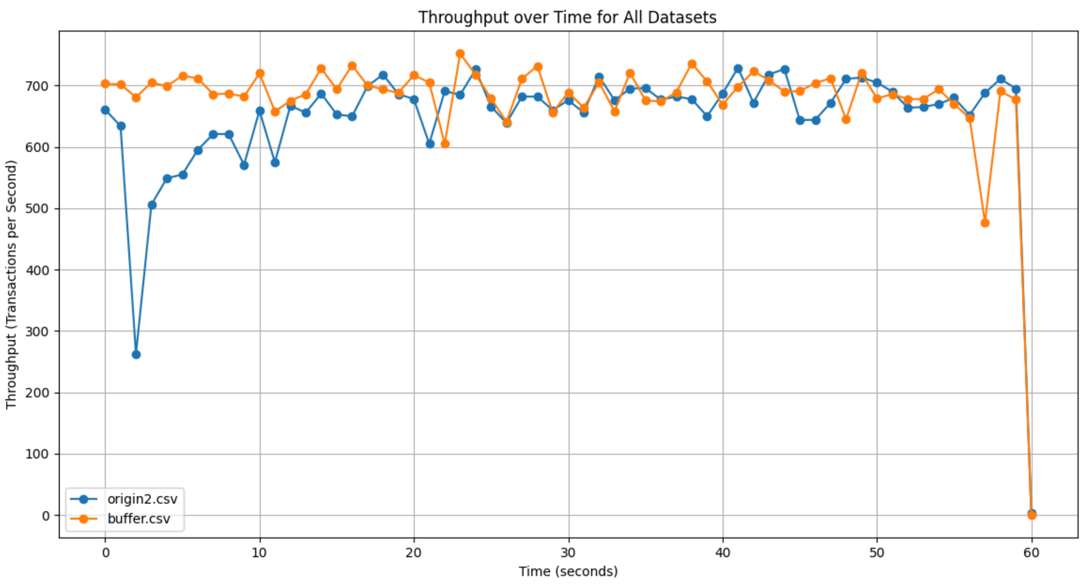
```
# of txns (including aborted) during benchmark period: 39482
ORDER_STATUS - committed: 0, aborted: 0, avg latency: 0 ms
STOCK_LEVEL - committed: 0, aborted: 0, avg latency: 0 ms
DELIVERY - committed: 0, aborted: 0, avg latency: 0 ms
PAYMENT - committed: 19784, aborted: 1, avg latency: 1 ms
NEW_ORDER - committed: 19697, aborted: 0, avg latency: 4 ms
TOTAL - committed: 39481, aborted: 1, avg latency: 3 ms
```

buffer優化

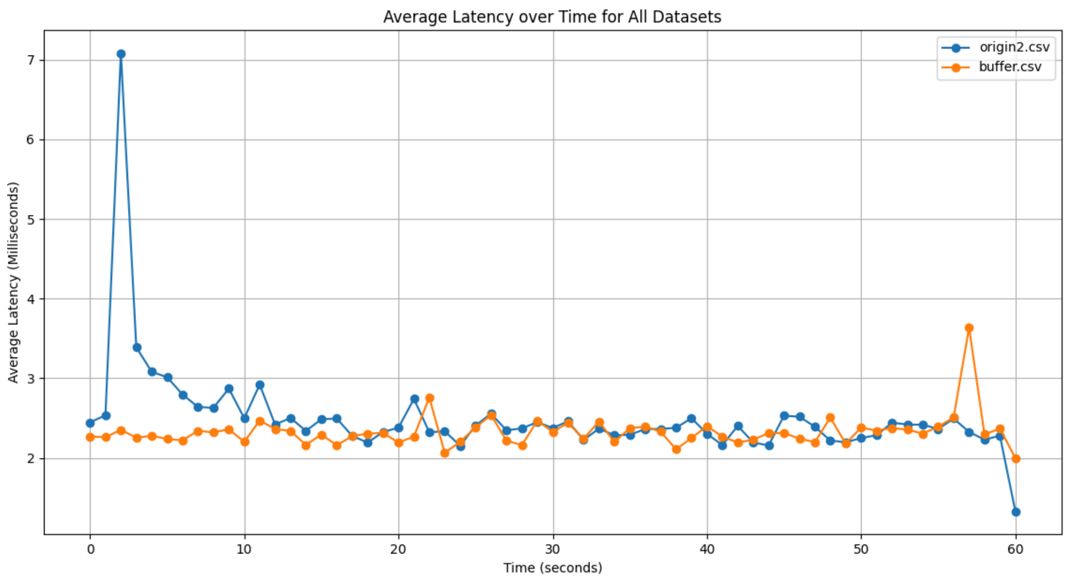


```
# of txns (including aborted) during benchmark period: 41323
ORDER_STATUS - committed: 0, aborted: 0, avg latency: 0 ms
STOCK_LEVEL - committed: 0, aborted: 0, avg latency: 0 ms
DELIVERY - committed: 0, aborted: 0, avg latency: 0 ms
PAYMENT - committed: 20570, aborted: 0, avg latency: 1 ms
NEW_ORDER - committed: 20753, aborted: 0, avg latency: 3 ms
TOTAL - committed: 41323, aborted: 0, avg latency: 3 ms
```

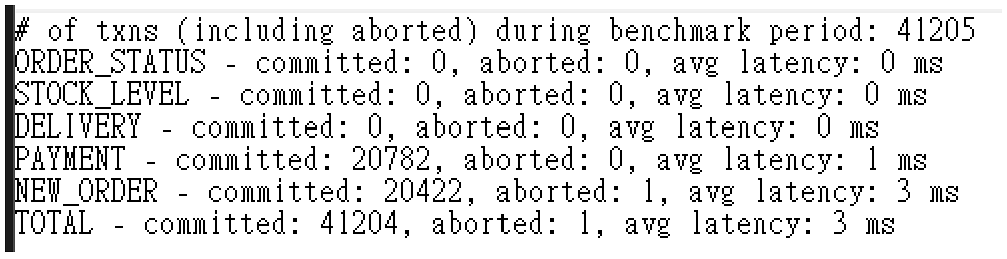
plot comparison-throughput (origin vs buffer optimization)



plot comparison-latency (origin vs buffer optimization)

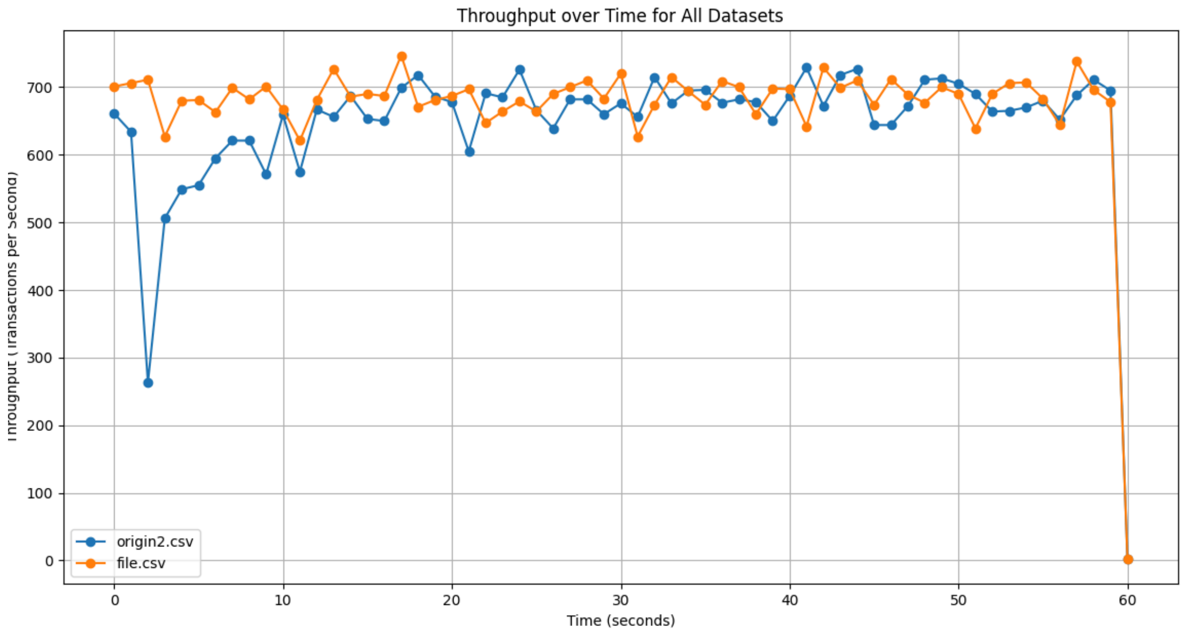


file優化:

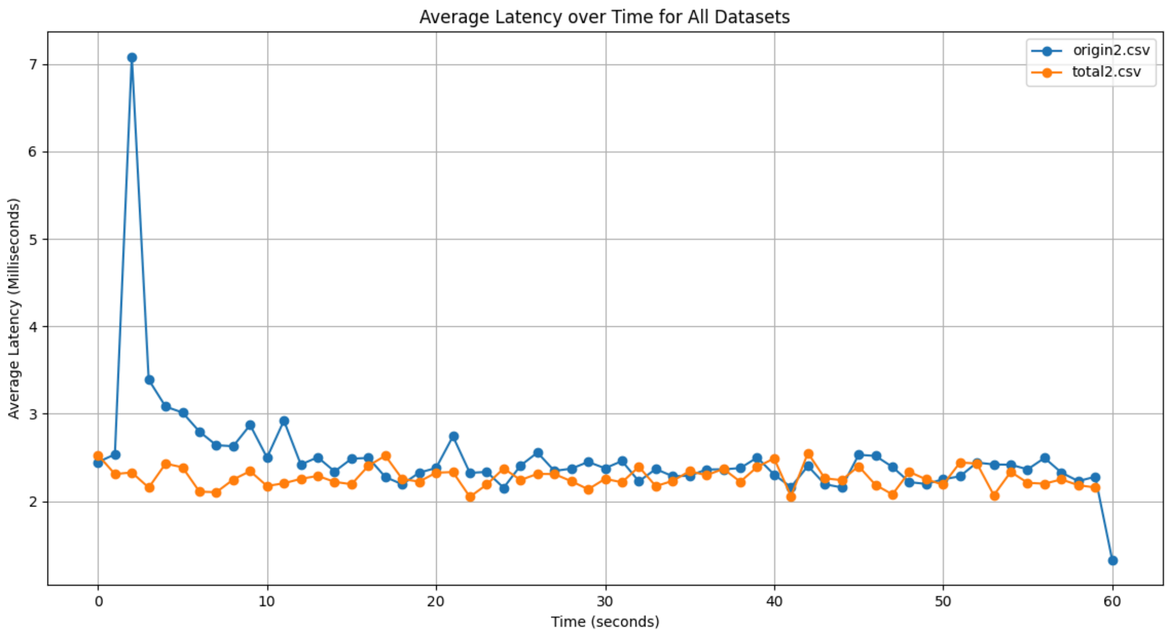


```
# of txns (including aborted) during benchmark period: 41205
ORDER_STATUS - committed: 0, aborted: 0, avg latency: 0 ms
STOCK_LEVEL - committed: 0, aborted: 0, avg latency: 0 ms
DELIVERY - committed: 0, aborted: 0, avg latency: 0 ms
PAYMENT - committed: 20782, aborted: 0, avg latency: 1 ms
NEW_ORDER - committed: 20422, aborted: 1, avg latency: 3 ms
TOTAL - committed: 41204, aborted: 1, avg latency: 3 ms
```

plot comparison-throughput (origin vs file optimization)



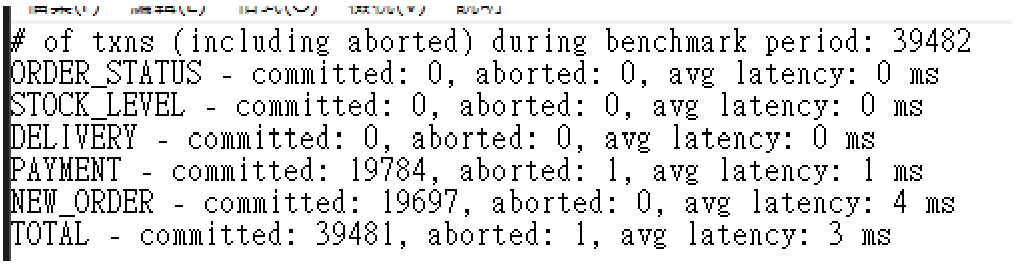
plot comparison-latency (origin vs file optimization)



我們發現file和buffer的優化程度都差不多，幾乎相同。可能是因為我們使用的優化策略大致相同。

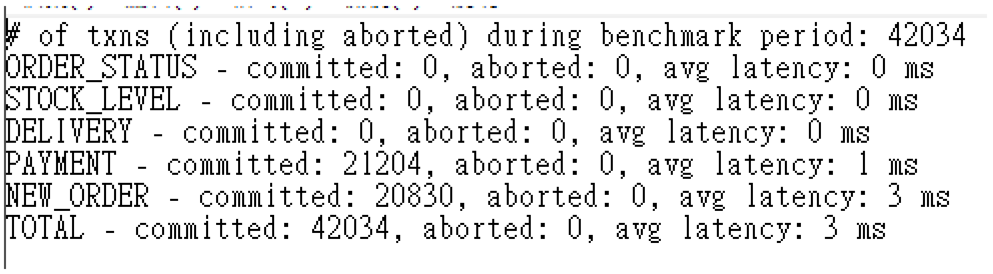
1. 整體優化比較:

原始:



```
# of txns (including aborted) during benchmark period: 39482
ORDER_STATUS - committed: 0, aborted: 0, avg latency: 0 ms
STOCK_LEVEL - committed: 0, aborted: 0, avg latency: 0 ms
DELIVERY - committed: 0, aborted: 0, avg latency: 0 ms
PAYMENT - committed: 19784, aborted: 1, avg latency: 1 ms
NEW_ORDER - committed: 19697, aborted: 0, avg latency: 4 ms
TOTAL - committed: 39481, aborted: 1, avg latency: 3 ms
```

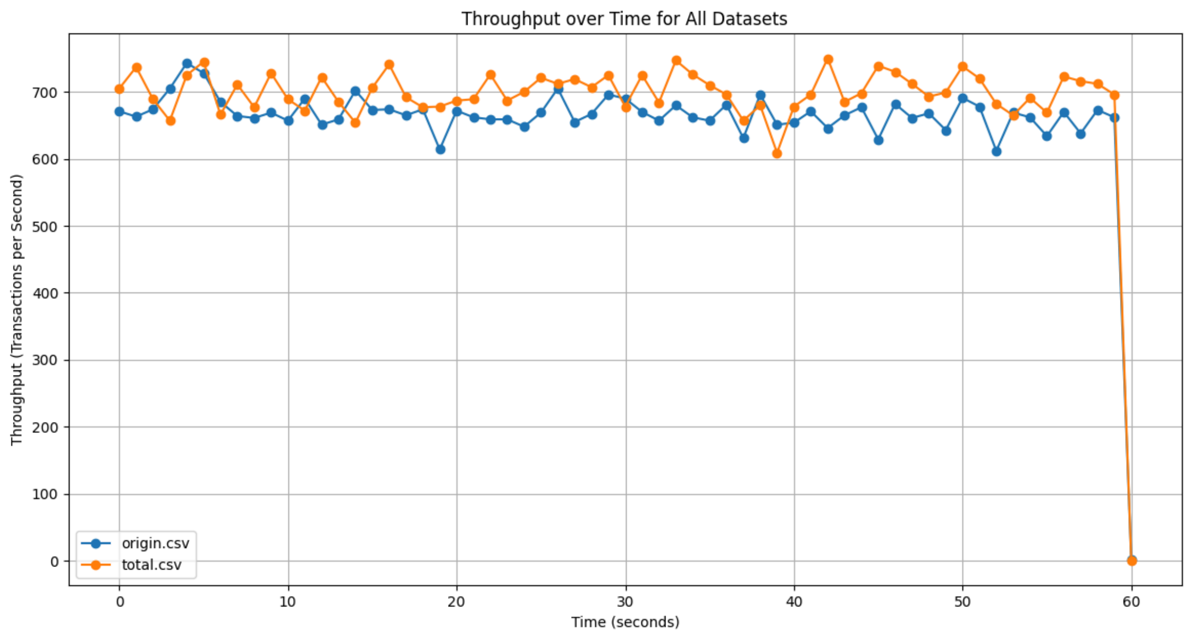
整體優化:



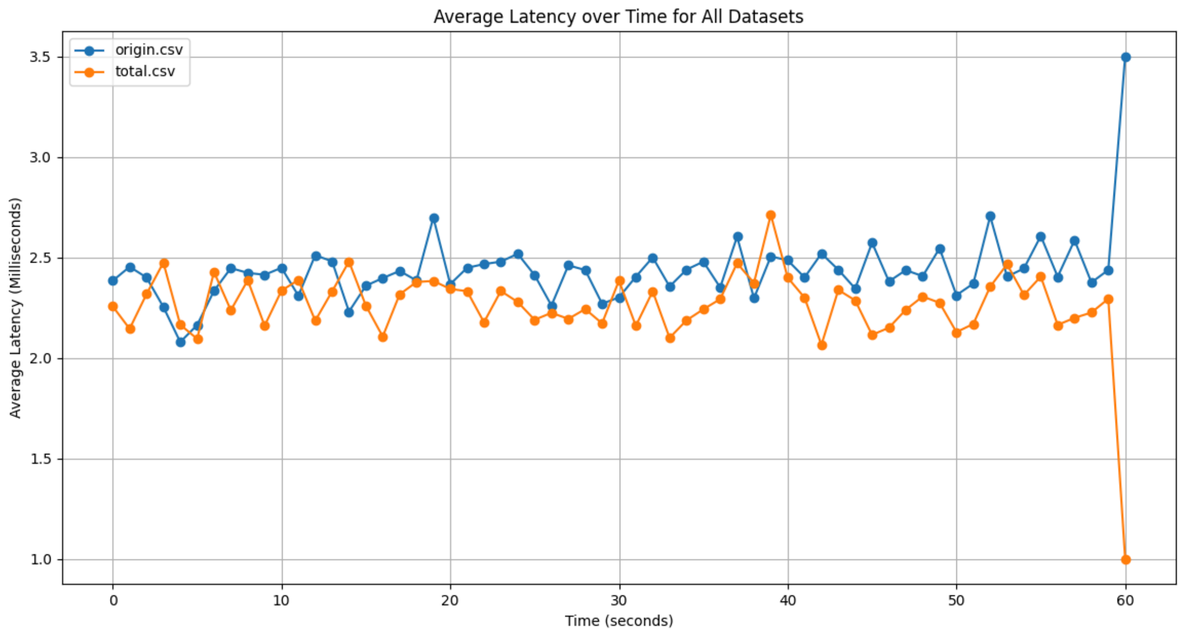
```
# of txns (including aborted) during benchmark period: 42034
ORDER_STATUS - committed: 0, aborted: 0, avg latency: 0 ms
STOCK_LEVEL - committed: 0, aborted: 0, avg latency: 0 ms
DELIVERY - committed: 0, aborted: 0, avg latency: 0 ms
PAYMENT - committed: 21204, aborted: 0, avg latency: 1 ms
NEW_ORDER - committed: 20830, aborted: 0, avg latency: 3 ms
TOTAL - committed: 42034, aborted: 0, avg latency: 3 ms
```

優化結果: committed 多了近2500, avg latency of new_order 減少1ms。

plot comparison-throughput (origin vs total optimization)



plot comparison-latency (origin vs total optimization)



從分析中我們可以發現經過整體優化後的程式碼在大部分時間點均有較高的thorughput以及較低的layency。