Assignmen3 Report

Implement Explain operantion

- Smaller Critical Section:
  在 BufferMgr 這個檔案中，我們更動了一些 syncronized 的長度，並把在不同 class 中重複 syncronized 的部份消除，以釋放出更多資料提供其他工作程序使用。
  (1) pin()

```
// Pinning process
try {
    Buffer buff;
    long timestamp = System.currentTimeMillis();
    synchronized (bufferPool) {
        // Try to pin a buffer or the pinned buffer for the given BlockId
        buff = bufferPool.pin(blk);

        // If there is no such buffer or no available buffer,
        // wait for it
        if (buff == null) {
            waitingThreads.add(Thread.currentThread());

            while (buff == null && !waitingTooLong(timestamp)) {
                bufferPool.wait(MAX_TIME);
                if (waitingThreads.get(0).equals(Thread.currentThread()))
                    buff = bufferPool.pin(blk);
            }

            waitingThreads.remove(Thread.currentThread());

            // Wake up other waiting threads (after leaving this critical section)
            bufferPool.notifyAll();
        }
    }

    // If it still has no buffer after a long wait,
    // release and re-pin all buffers it has
    if (buff == null) {
        repin();
        buff = pin(blk);
    } else {
        pinningBuffers.put(buff.block(), new PinningBuffer(buff));
    }

    return buff;
```

  (2)pinNew()

```
try {
    Buffer buff;
    long timestamp = System.currentTimeMillis();
    synchronized (bufferPool) {
        // Try to pin a buffer or the pinned buffer for the given BlockId
        buff = bufferPool.pinNew(fileName, fmtr);

        // If there is no such buffer or no available buffer,
        // wait for it
        if (buff == null) {
            waitingThreads.add(Thread.currentThread());

            while (buff == null && !waitingTooLong(timestamp)) {
                bufferPool.wait(MAX_TIME);
                if (waitingThreads.get(0).equals(Thread.currentThread()))
                    buff = bufferPool.pinNew(fileName, fmtr);
            }

            waitingThreads.remove(Thread.currentThread());

            bufferPool.notifyAll();
        }
    }

        // If it still has no buffer after a long wait,
        // release and re-pin all buffers it has
    if (buff == null) {
        repin();
        buff = pinNew(fileName, fmtr);
    } else {
        pinningBuffers.put(buff.block(), new PinningBuffer(buff));
    }

    return buff;
```

(3) unpin()

```
public void unpin(Buffer buff) {
    BlockId blk = buff.block();
    PinningBuffer pinnedBuff = pinningBuffers.get(blk);

    if (pinnedBuff != null) {
        pinnedBuff.pinCount--;
        if (pinnedBuff.pinCount == 0) {
            bufferPool.unpin(buff);
            pinningBuffers.remove(blk);
            synchronized (bufferPool) {
                // it was before (236) if (pinnedBuff != null)
                bufferPool.notifyAll();
            }
        }
    }
}
```

(4) flushAll()

```
public void flushAll() {
    //already synchronized in BufferPoolMgr
    bufferPool.flushAll();
}
```

(5) flushAllMyBuffers()

```
public void flushAllMyBuffers() {
    // Unsynchronized
    for (Buffer buff : buffersToFlush) {
        buff.flush();
    }
}
```

(6) unpinAll()

```
private void unpinAll(Transaction tx) {
    // Copy the set of pinned buffers to avoid ConcurrentModificationException
    Set<PinningBuffer> pinnedBuffs = new HashSet<PinningBuffer>(pinningBuffers.values());
    if (pinnedBuffs != null) {
        for (PinningBuffer pinnedBuff : pinnedBuffs)
            bufferPool.unpin(pinnedBuff.buffer);
    }
    synchronized (bufferPool) {
        // it was synchronized all of the function.
        bufferPool.notifyAll();
    }
}
```

(7) repin()

```
try {
    // Copy the pinned buffers to avoid ConcurrentModificationException
    List<BlockId> blksToBeRepinned = new LinkedList<BlockId>();
    List<Buffer> buffersToBeUnpinned = new LinkedList<Buffer>();

    // Unpin all buffers it has
    for (Entry<BlockId, PinningBuffer> entry : pinningBuffers.entrySet()) {
        blksToBeRepinned.add(entry.getKey());
        buffersToBeUnpinned.add(entry.getValue().buffer);
    }

    // Un-pin all buffers it has
    for (Buffer buf : buffersToBeUnpinned)
        unpin(buf);

    synchronized (bufferPool) {
        // Wait other threads pinning blocks
        bufferPool.wait(MAX_TIME);
    }

    // Re-pin all blocks
    for (BlockId blk : blksToBeRepinned)
        pin(blk);
```

- Never Do It Again
  在 blockID 這個檔案裡，原本每次呼叫 toString 跟 hashCode 這兩個 function 的時候，都會重新去生成一個 string 重新計算 hashCode 可是同樣一個 fileName & blkNum 產生出來的 string 都是一樣的，然後根據同一個 string 生成的 hashcode 也是一樣的，所以我們把這個動作，移到 constructor 去，這樣就避免了重複執行的過程

```java
private String fileName;
private long blkNum;
private String curString;
private int curHashCode;

/**
 * Constructs a block ID for the specified fileName and block number.
 *
 * @param fileName the name of the file
 * @param blkNum   the block number
 */
public BlockId(String fileName, long blkNum) {
    this.fileName = fileName;
    this.blkNum = blkNum;
    this.curString = "[file " + fileName + ", block " + blkNum + "]";
    this.curHashCode = curString.hashCode();
}
```

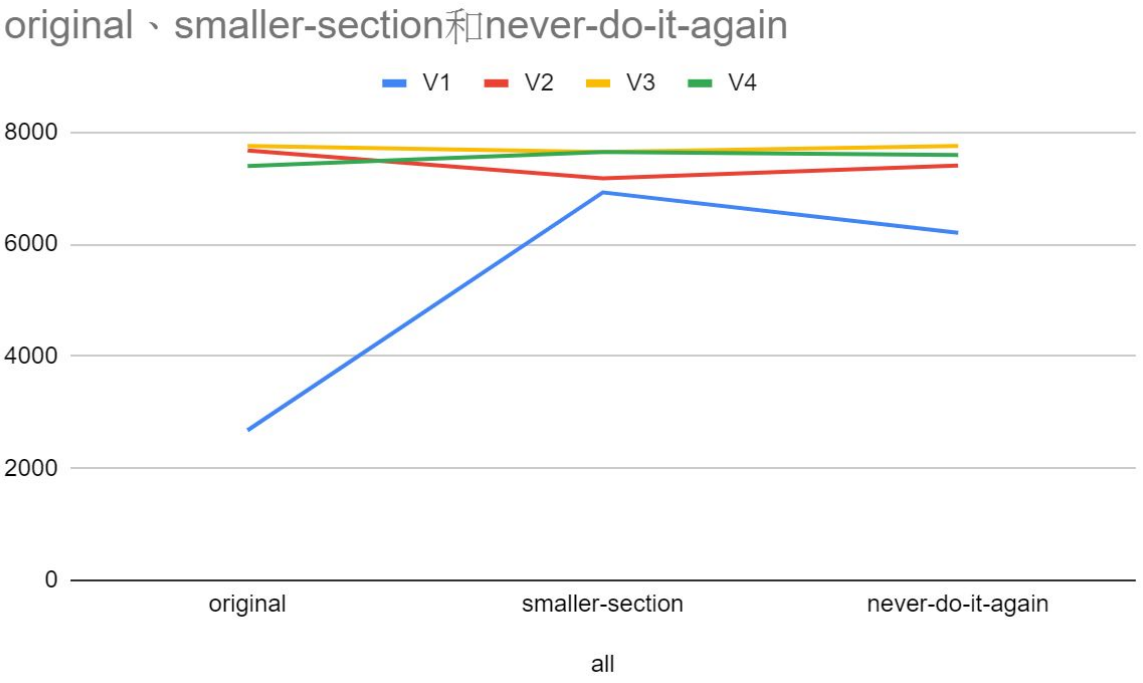
```java
@Override
public String toString() {
    return curString;
}

@Override
public int hashCode() {
    return curHashCode;
}
```

Experiments:

- Experiment Environment:
  Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz, 12 GB RAM, 64位元作業系統, x64型處理器

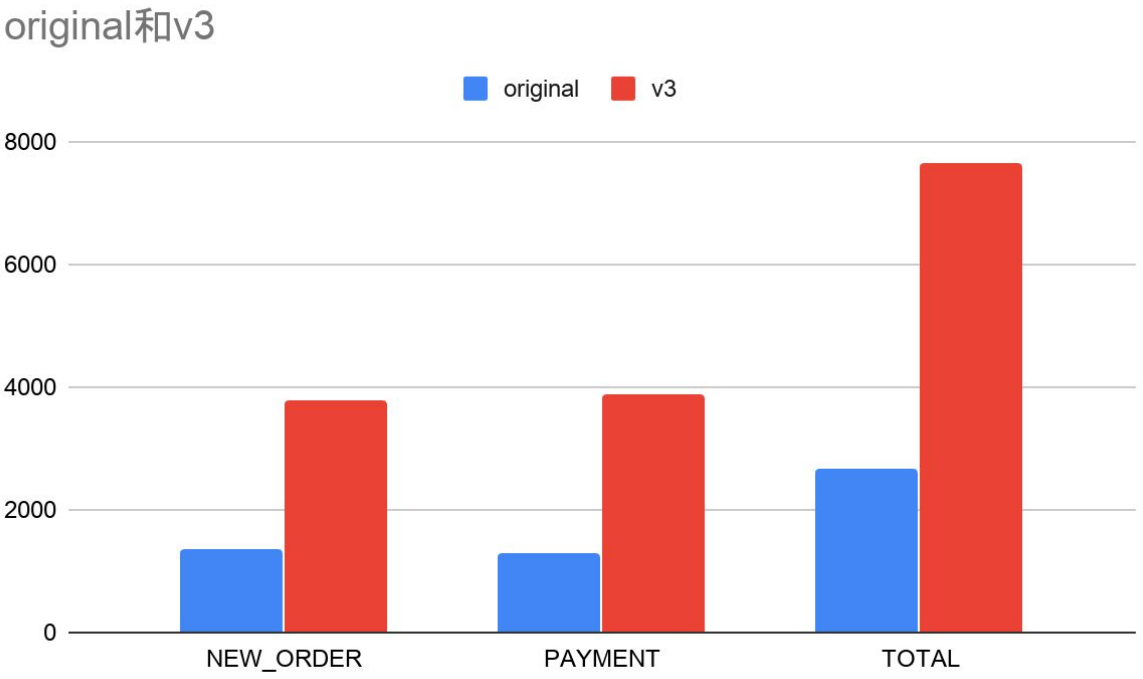
- Benchmarks and Parameters:

	RTE	Write Tx Rate	Conflict Rate	Buffer Pool Size
V1	1	0.2	0.001	102400
V2	2	0.2	0.001	102400
V3	2	0.5	0.001	102400
V4	2	0.5	0.1	102400



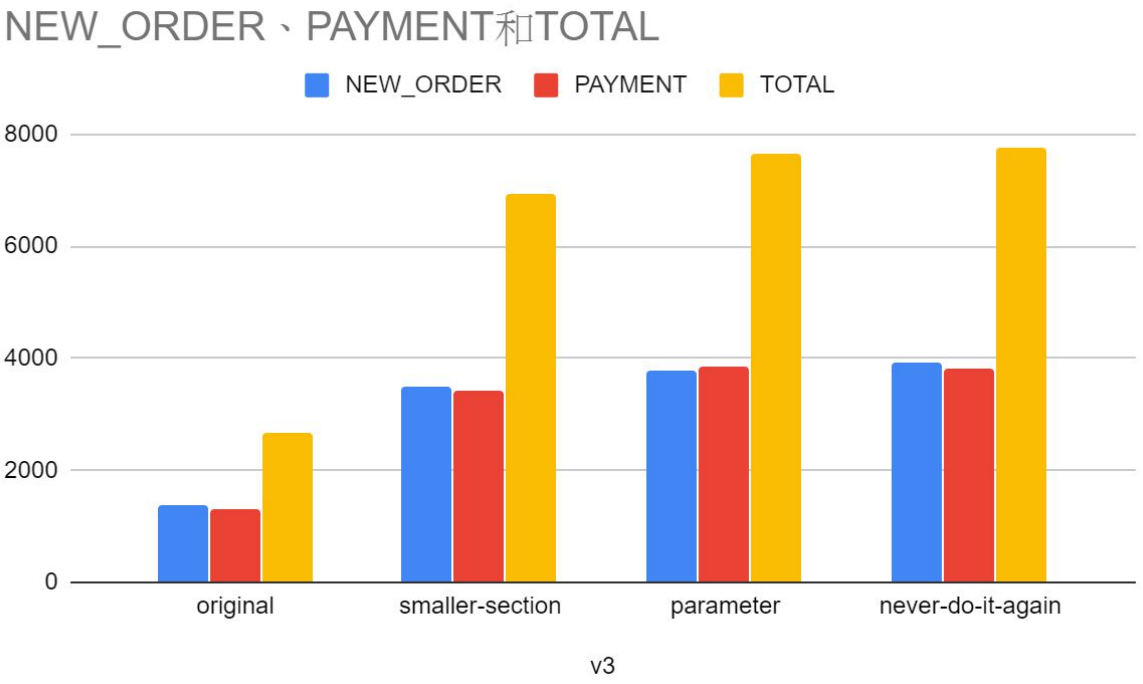
從各方面來看 V3 的效能整體來說是最好的。

- Overall Improvement:

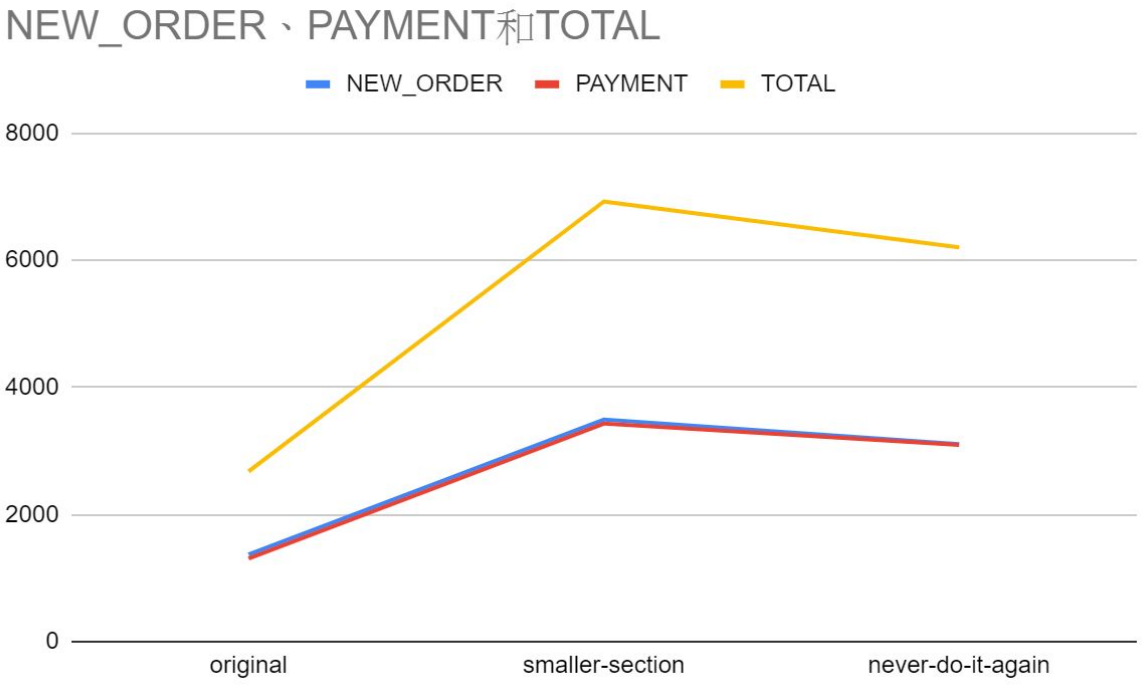


由圖可見，V3在經過 critical section 的縮短以及避免重複做同樣的運算(never-do-it-again)，再加上parameter 數值的修改，在各方面的效能都比原來的版本有所提升。

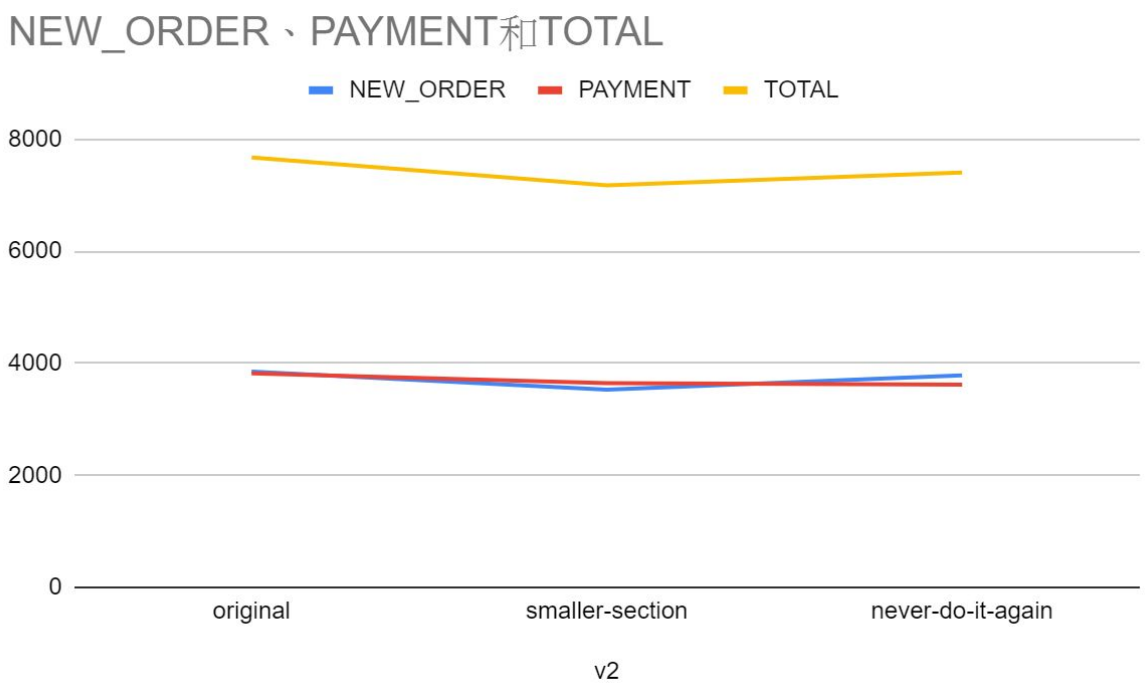
- Improvement by Each Optimization:



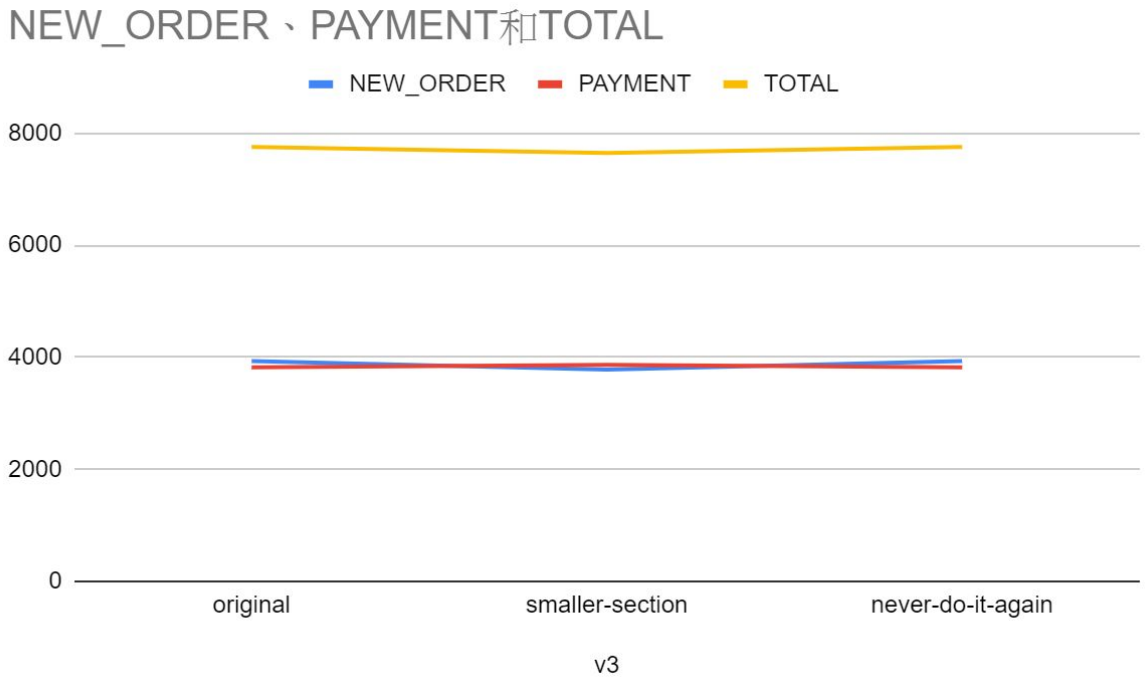
v1



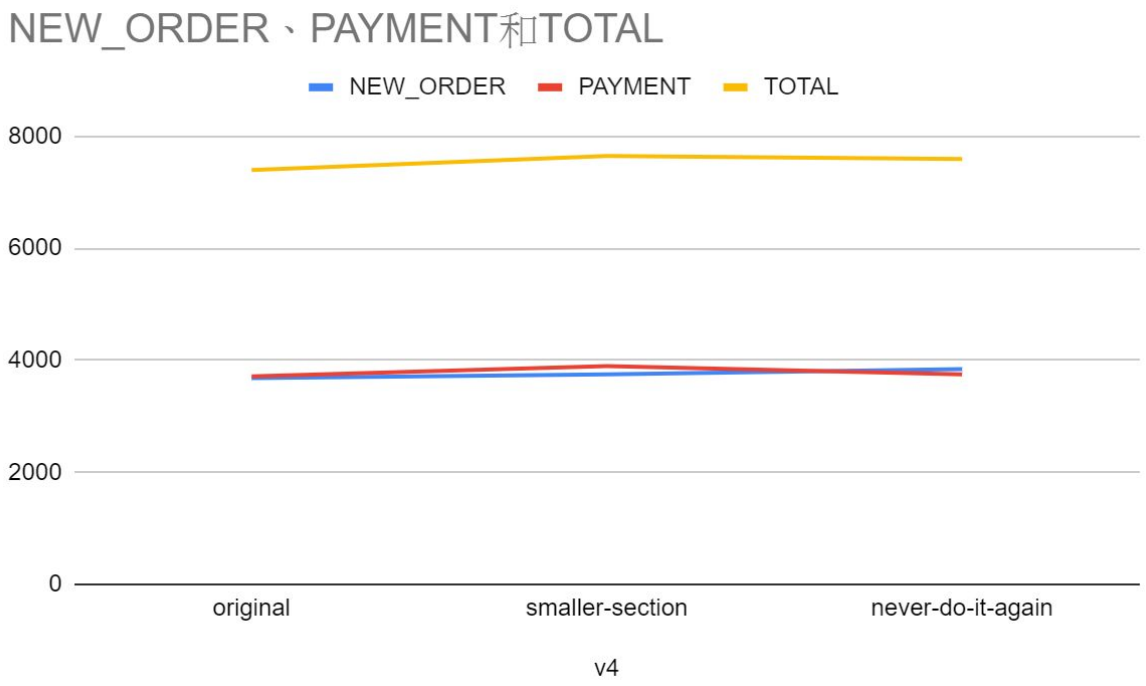
v2


NEW_ORDER、PAYMENT和TOTAL
NEW_ORDER
PAYMENT
TOTAL
8000
6000
4000
2000
0
original
smaller-section
never-do-it-again
v2

v3


NEW_ORDER、PAYMENT和TOTAL
NEW_ORDER
PAYMENT
TOTAL
8000
6000
4000
2000
0
original
smaller-section
never-do-it-again
v3

v4



附上數據 (可以看上面的折線圖就好)

v1

RTE = 1
Write Tx Rate = 0.2
Conflict Rate = 0.001
Buffer Pool Size = 102400

original---------
NEW_ORDER - committed: 1372, aborted: 0, avg latency: 40 ms
PAYMENT - committed: 1306, aborted: 0, avg latency: 3 ms
TOTAL - committed: 2678, aborted: 0, avg latency: 22 ms

smaller-section----------
NEW_ORDER - committed: 3494, aborted: 0, avg latency: 14 ms
PAYMENT - committed: 3430, aborted: 0, avg latency: 2 ms
TOTAL - committed: 6924, aborted: 0, avg latency: 9 ms

never-do-it-again----------
NEW_ORDER - committed: 3108, aborted: 0, avg latency: 16 ms
PAYMENT - committed: 3096, aborted: 0, avg latency: 2 ms
TOTAL - committed: 6204, aborted: 0, avg latency: 10 ms

v2

RTE = 2
Write Tx Rate = 0.2
Conflict Rate = 0.001
Buffer Pool Size = 102400

original---------
NEW_ORDER - committed: 3854, aborted: 0, avg latency: 21 ms
PAYMENT - committed: 3819, aborted: 0, avg latency: 9 ms
TOTAL - committed: 7673, aborted: 0, avg latency: 16 ms

smaller-section----------
NEW_ORDER - committed: 3529, aborted: 0, avg latency: 23 ms
PAYMENT - committed: 3647, aborted: 0, avg latency: 10 ms
TOTAL - committed: 7176, aborted: 0, avg latency: 17 ms

never-do-it-again----------
NEW_ORDER - committed: 3784, aborted: 0, avg latency: 22 ms
PAYMENT - committed: 3621, aborted: 1, avg latency: 9 ms
TOTAL - committed: 7405, aborted: 1, avg latency: 16 ms

v3

RTE = 2
Write Tx Rate = 0.5
Conflict Rate = 0.001
Buffer Pool Size = 102400

original---------
NEW_ORDER - committed: 3934, aborted: 0, avg latency: 21 ms
PAYMENT - committed: 3822, aborted: 0, avg latency: 9 ms
TOTAL - committed: 7756, aborted: 0, avg latency: 15 ms

smaller-section----------
NEW_ORDER - committed: 3782, aborted: 0, avg latency: 21 ms
PAYMENT - committed: 3865, aborted: 0, avg latency: 9 ms
TOTAL - committed: 7647, aborted: 0, avg latency: 16 ms

never-do-it-again----------
NEW_ORDER - committed: 3934, aborted: 0, avg latency: 21 ms
PAYMENT - committed: 3822, aborted: 0, avg latency: 9 ms
TOTAL - committed: 7756, aborted: 0, avg latency: 15 ms

v4

RTE = 2
Write Tx Rate = 0.5
Conflict Rate = 0.1
Buffer Pool Size = 102400

original---------
NEW_ORDER - committed: 3682, aborted: 0, avg latency: 22 ms
PAYMENT - committed: 3715, aborted: 0, avg latency: 9 ms
TOTAL - committed: 7397, aborted: 0, avg latency: 16 ms

smaller-section----------
NEW_ORDER - committed: 3749, aborted: 0, avg latency: 21 ms
PAYMENT - committed: 3897, aborted: 0, avg latency: 9 ms
TOTAL - committed: 7646, aborted: 0, avg latency: 16 ms

never-do-it-again----------
NEW_ORDER - committed: 3845, aborted: 0, avg latency: 21 ms
PAYMENT - committed: 3747, aborted: 0, avg latency: 9 ms
TOTAL - committed: 7592, aborted: 0, avg latency: 16 ms