# Database As5 report

104062203 陳涵宇
104062232 廖子毅
104062234 林士軒

# Implementation 2V2PL

1. 由於 Lock Table 已經先完成 MGL 的部分，我們只要增加 shadowX 即可完成。 所以我們首先修改 Lock Table 讓他可以支持 Shadow X Lock 。
2. 在 Transaction class 中增加三個 function。
    - processWorkspace : 這個 func 功能是將原先將被直接寫入的資料先暫存起來。詳細實作方法是將欲寫入的 BlockId, offset, val, lsn 儲存在一個 concurrentHashMap 中。HashMap 的 Key 為 BlockId, Value 則為 Vector<Workspace> ( Workspace = {offset, val, lsn})。
    - getVal : 查詢目前的 workspace 中是否有需要的資料，有的話則回傳該值，無則回傳 null。查找方式是每次搜尋整個陣列。
    - wireBuff : 將欲修改的資料寫入。此 func 主要方式為：
        1. 將 HashTable 每個 Vector 都提取出來。
        2. 拿取與 BlockID 相對應的 XLock，將 Vector 內的資料依序放入。
3. 修改 RecordPage 的 setVal 部分，將原先 currenBuff.setVal() 的部分刪除。由於此部分會直接將值寫入，會造成其他 tx 在讀取資料時發生錯誤 ( 不符合 shadowX 的行為)。改成將欲修改的資料暫存在 tx 的 workspace 中。也就是說，此部分的更動為將 currentBuff.setVal() 替換成 this.tx.processWorkspace() 將資料暫存而非直接寫入。
4. 增加一個 TwoVersionTwoPhaseLock 的 class 來當作 concurrencyMgr 的介面，並修改各個 concurrencyMgr modifyRecord func 使其取得 sxLock 而不是 xLock。

- 整體實作流程為 :

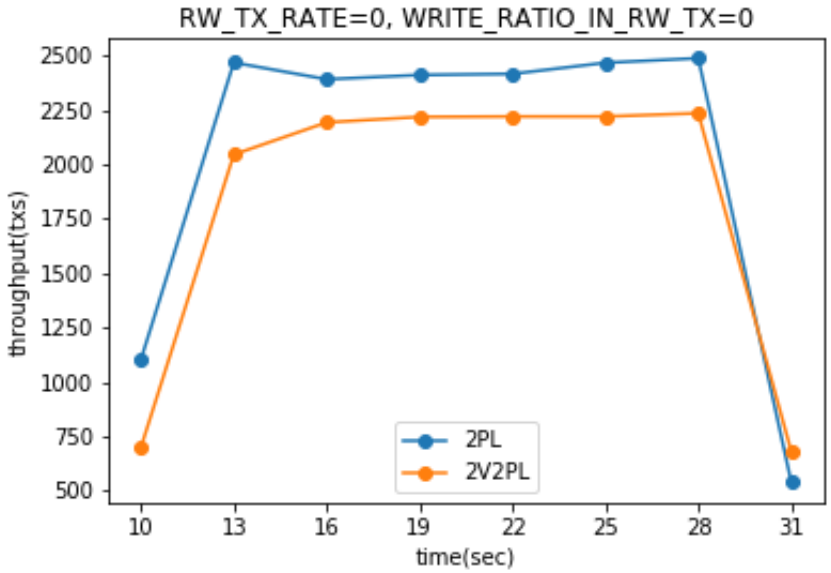
  Lock Table -> 修改 Workspace 使資料在 commit 前寫入 -> 支持 Shadow X Lock 功能。

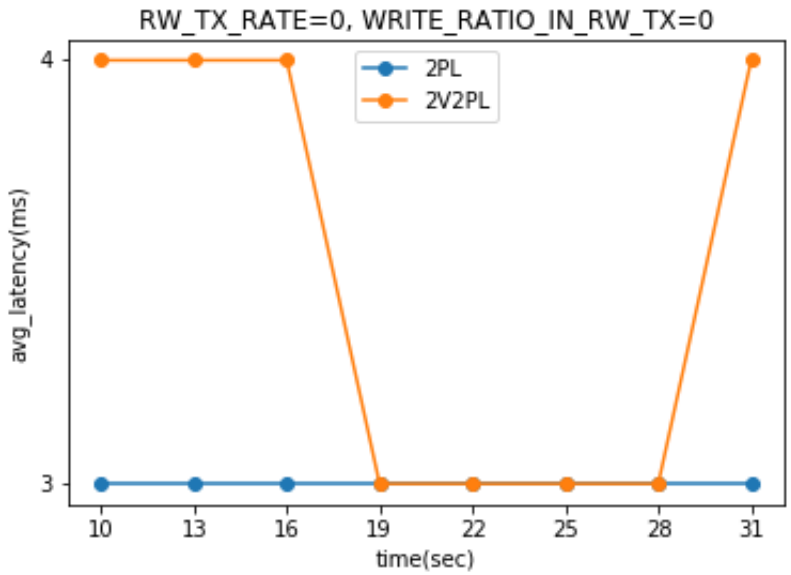
# Experiments

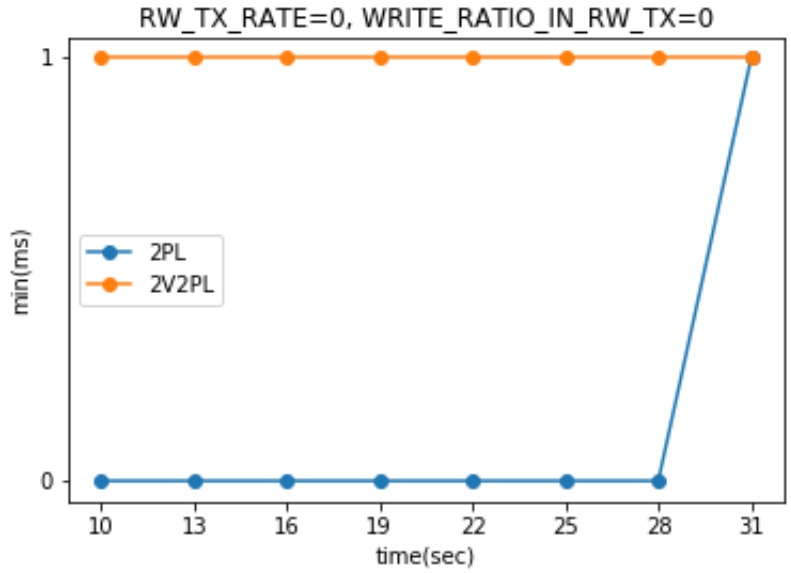
## Environment


## experiment compare

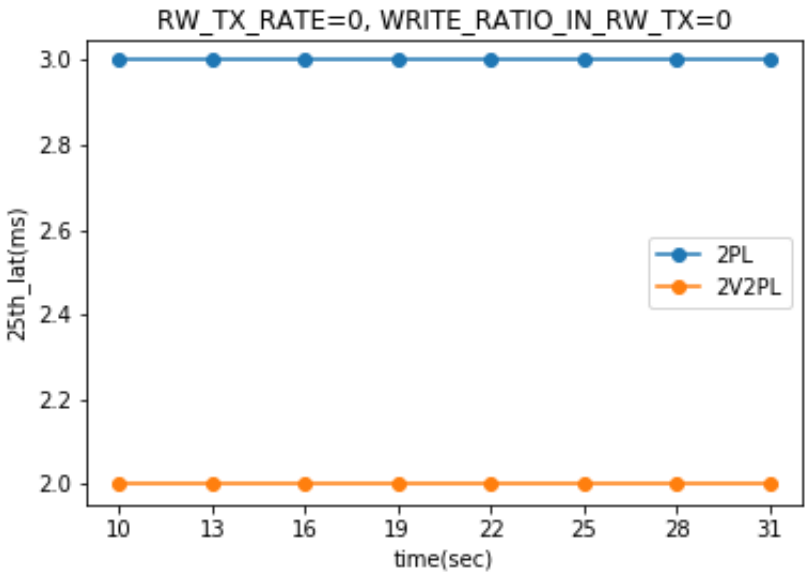
首先是測試極端值 RW_TX_RATE=0, WRITE_RATIO_IN_RW_TX=0 的情況：

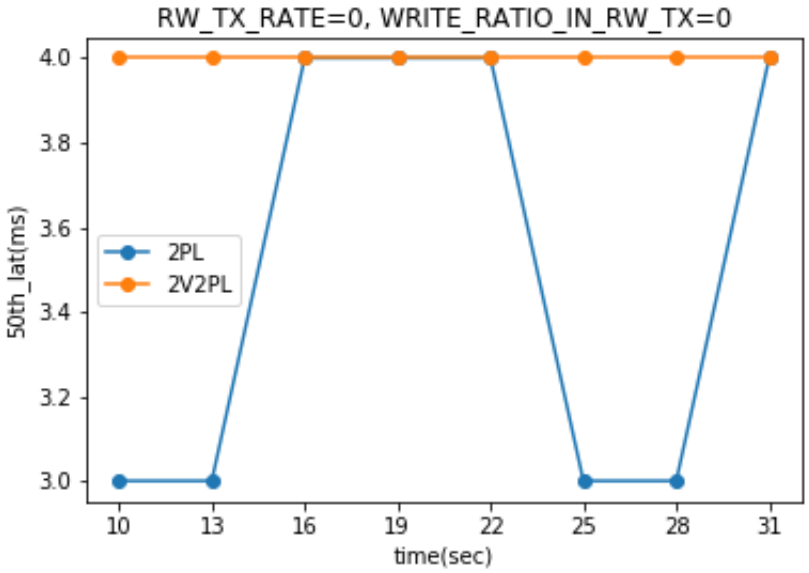


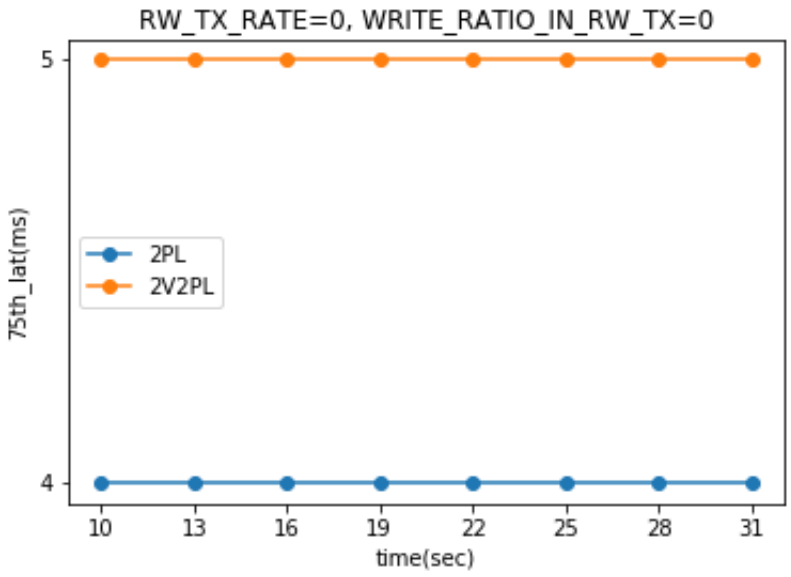

RW_TX_RATE=0, WRITE_RATIO_IN_RW_TX=0
2PL
2V2PL
avg_latency(ms)
4
3
10
13
16
19
22
25
28
31
time(sec)


RW_TX_RATE=0, WRITE_RATIO_IN_RW_TX=0
2PL
2V2PL
min(ms)
1
0
10
13
16
19
22
25
28
31
time(sec)

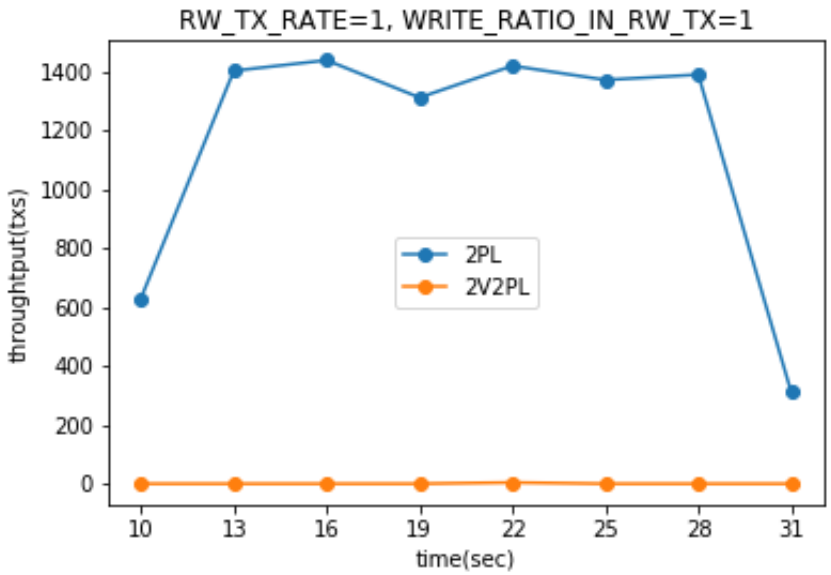

RW_TX_RATE=0, WRITE_RATIO_IN_RW_TX=0
2PL
2V2PL
max(ms)
80
70
60
50
40
30
20
10
10
13
16
19
22
25
28
31
time(sec)

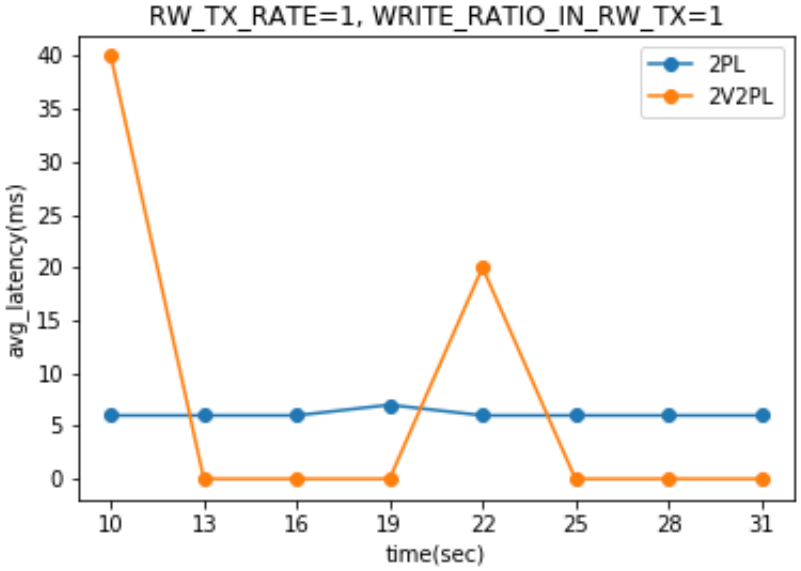


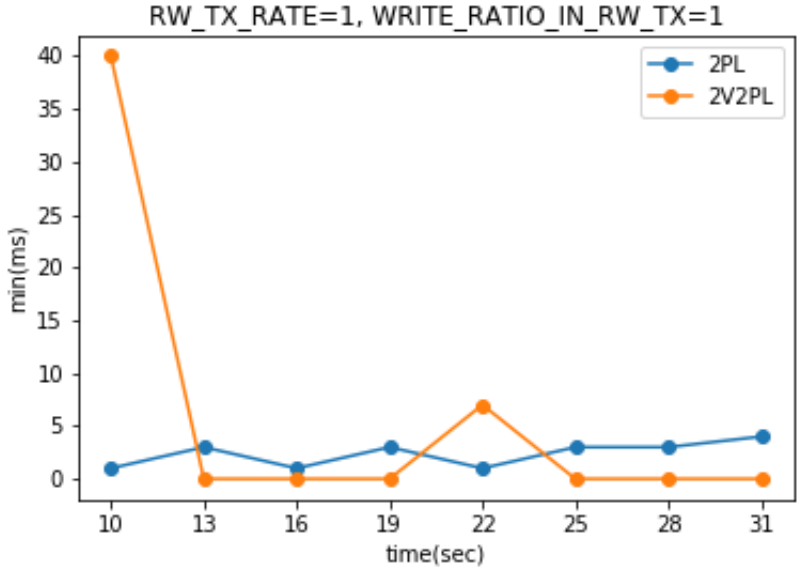



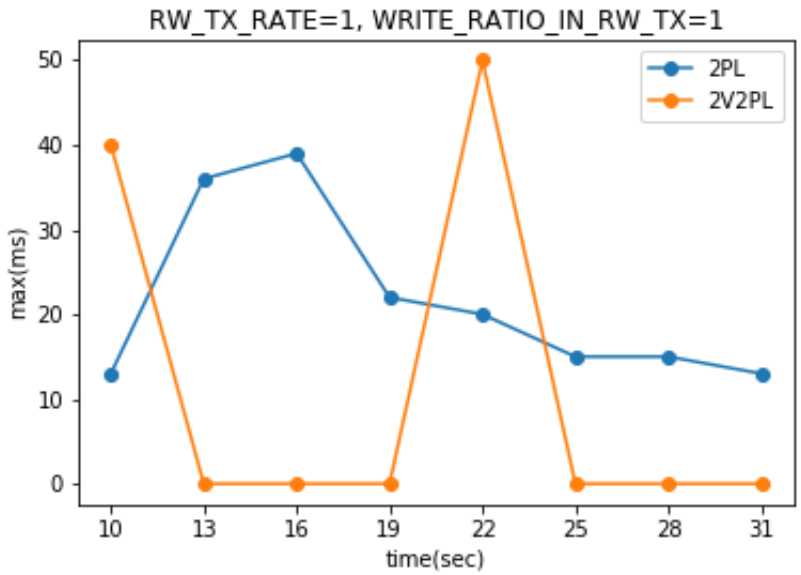




另一種極端值 RW_TX_RATE=1, WRITE_RATIO_IN_RW_TX=1：

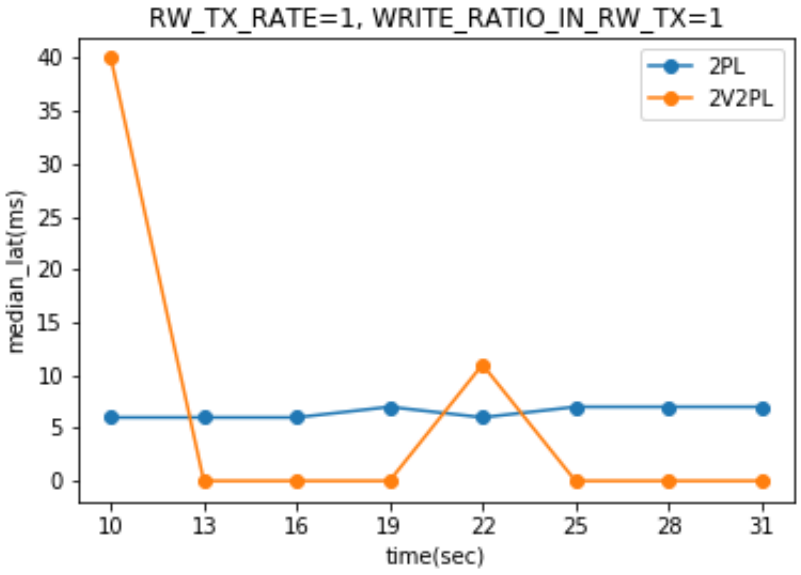

RW_TX_RATE=1, WRITE_RATIO_IN_RW_TX=1
throughtput(txs)
time(sec)
2PL
2V2PL
0
200
400
600
800
1000
1200
1400
10
13
16
19
22
25
28
31

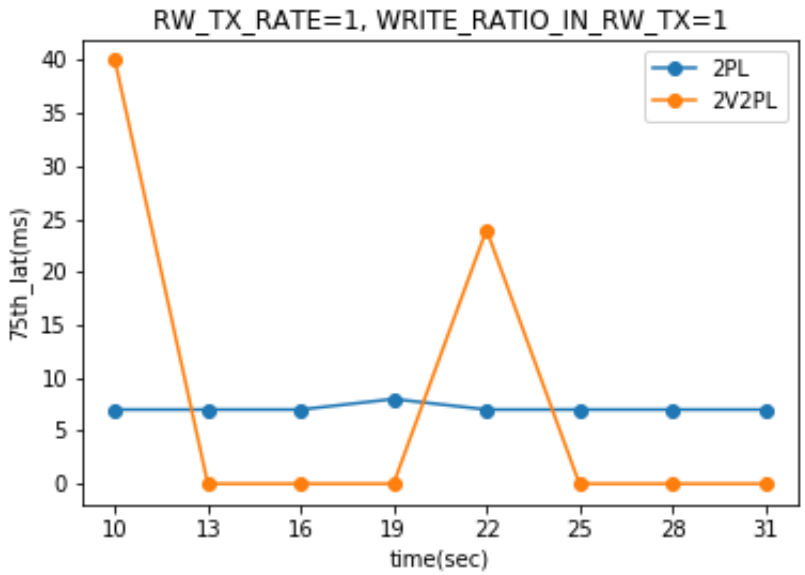

RW_TX_RATE=1, WRITE_RATIO_IN_RW_TX=1
avg_latency(ms)
time(sec)
2PL
2V2PL
0
5
10
15
20
25
30
35
40
10
13
16
19
22
25
28
31


RW_TX_RATE=1, WRITE_RATIO_IN_RW_TX=1
min(ms)
time(sec)
2PL
2V2PL
0
5
10
15
20
25
30
35
40
10
13
16
19
22
25
28
31

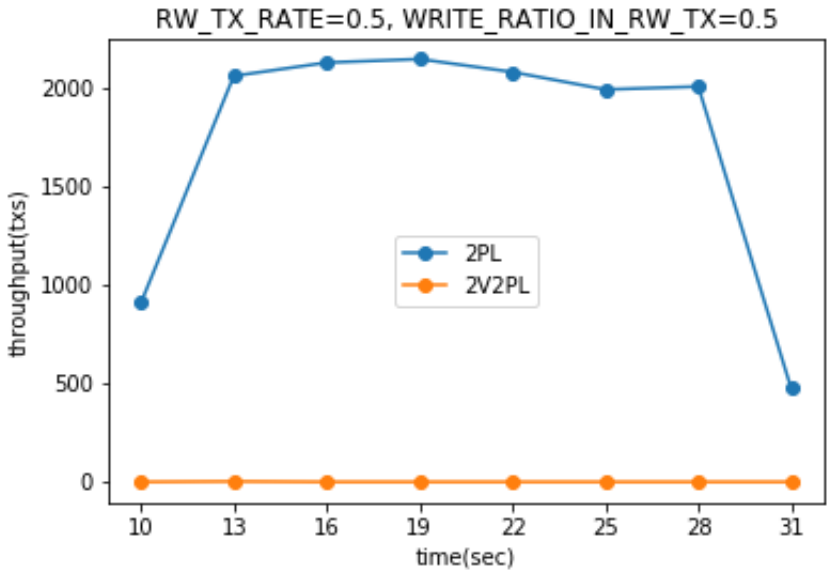

RW_TX_RATE=1, WRITE_RATIO_IN_RW_TX=1
2PL
2V2PL
max(ms)
time(sec)
0
10
20
30
40
50
10
13
16
19
22
25
28
31

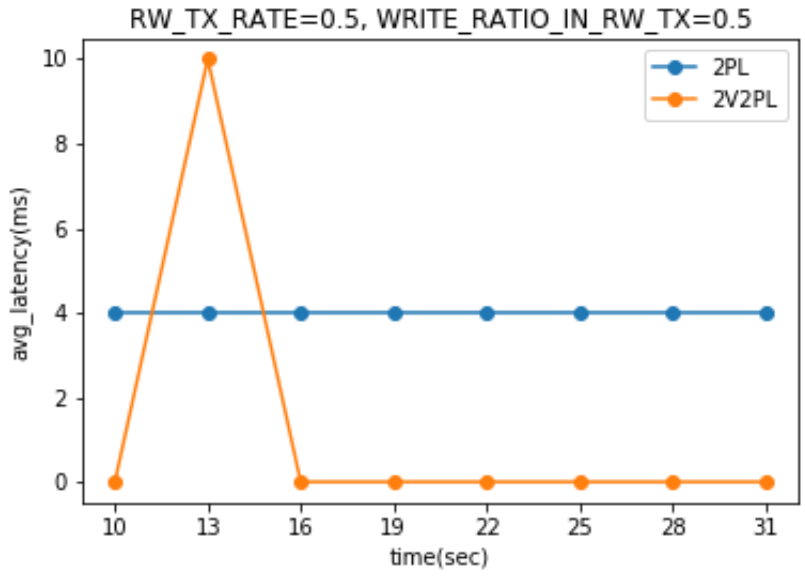

RW_TX_RATE=1, WRITE_RATIO_IN_RW_TX=1
2PL
2V2PL
25th_lat(ms)
time(sec)
0
5
10
15
20
25
30
35
40
10
13
16
19
22
25
28
31

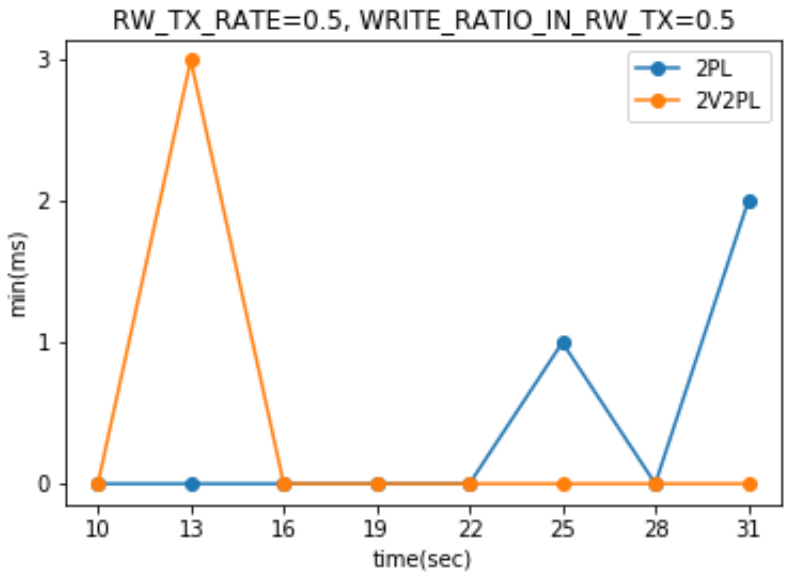

RW_TX_RATE=1, WRITE_RATIO_IN_RW_TX=1
2PL
2V2PL
median_lat(ms)
time(sec)
0
5
10
15
20
25
30
35
40
10
13
16
19
22
25
28
31

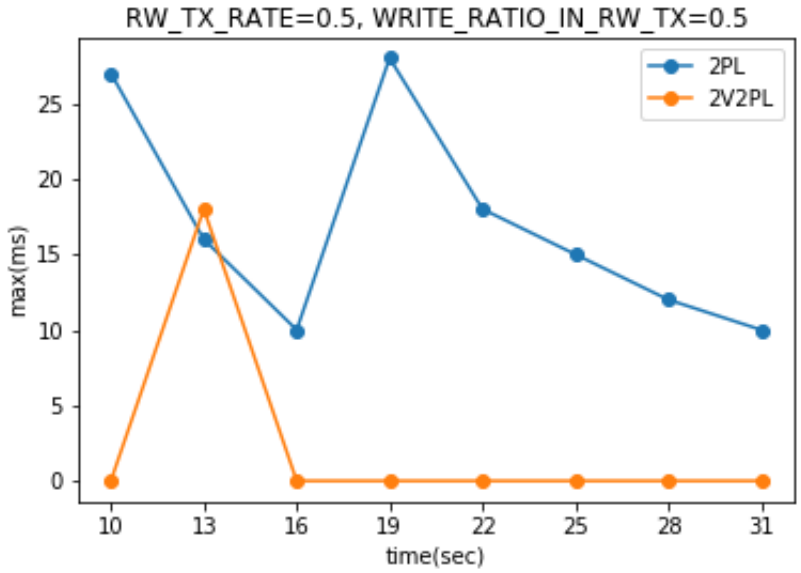


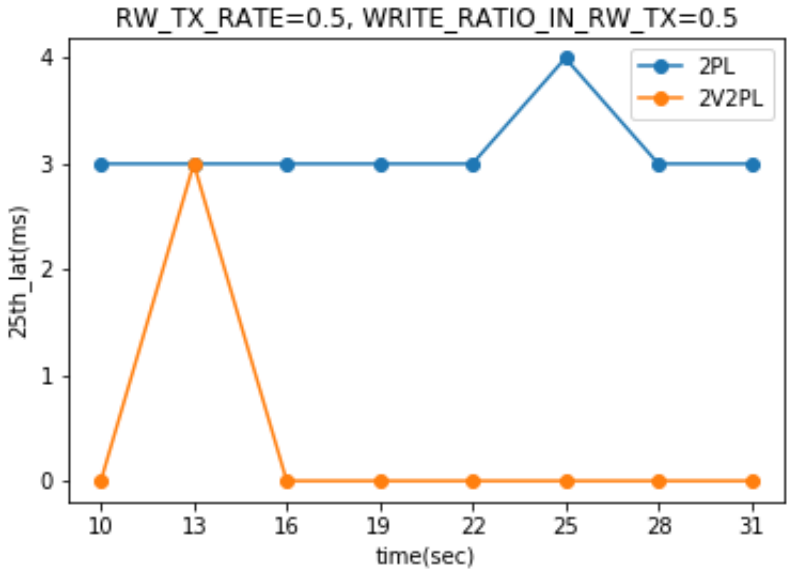

最後一種是 origin 的數據也是 median 的值 RW_TX_RATE=0.5, WRITE_RATIO_IN_RW_TX=0.5：

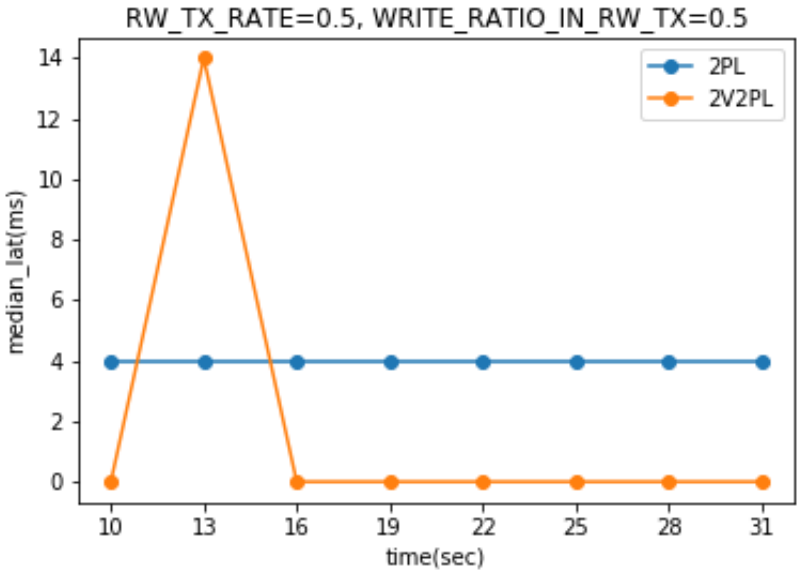


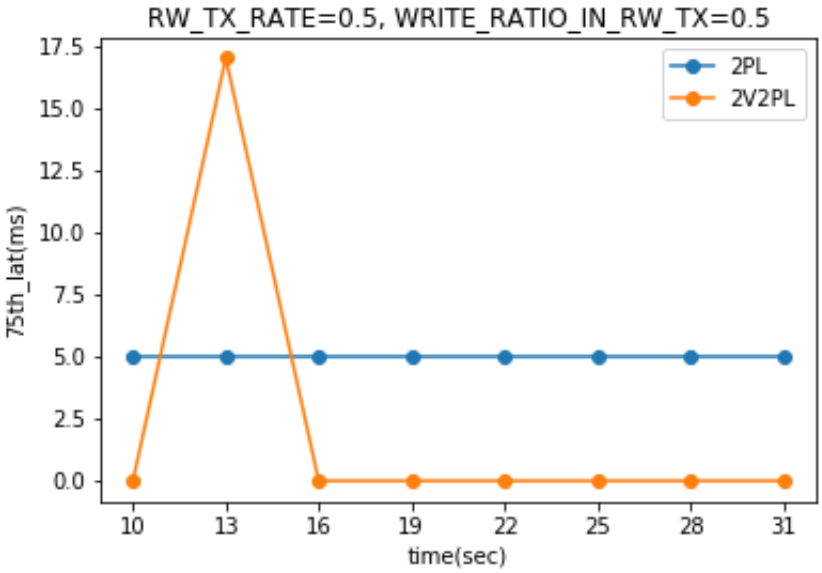




RW_TX_RATE=0.5, WRITE_RATIO_IN_RW_TX=0.5
2PL
2V2PL
min(ms)
0
1
2
3
10
13
16
19
22
25
28
31
time(sec)


RW_TX_RATE=0.5, WRITE_RATIO_IN_RW_TX=0.5
2PL
2V2PL
max(ms)
0
5
10
15
20
25
10
13
16
19
22
25
28
31
time(sec)


RW_TX_RATE=0.5, WRITE_RATIO_IN_RW_TX=0.5
2PL
2V2PL
25th_lat(ms)
0
1
2
3
4
10
13
16
19
22
25
28
31
time(sec)




---

藉由上述的三種 properties 設置，可以發現到 其實最後兩種在實作上面的 performance 是相差無幾的，也就是說雖然我們用了，理想上 performance 比較好的 2V2PL ，但是，實作上比較不好，或者整個系統的 bottleneck 根本不在這裡，所以才不會有 performance 上面的增進。