Team24_Assignment4_report

106034061 曾靖渝
106062116 黃晨
106062216 馮謙

Environment

1. OS: Mac OS Catalina 10.15.2
2. CPU: 2.7 GHz 四核心Intel Core i7
3. RAM: 8 GB 2133 MHz LPDDR3
4. GPU: Intel Iris Plus Graphics 655 1536 MB

Implementation

1. FileMgr
    a. All the critical sections in the FileMgr are removed.
2. BlockId
    a. Add private variable named “cacheHashCode” to store calculated hash code.
    b. Add private variable named “isNew” to indicate whether it has been accessed.
    c. In hashCode function, use isNew to check if we have calculated hash number. If we haven’t entered yet, assign isNew to false, calculate the hash number, and store it in cacheHashCode.
3. Page
    a. We unplug all the synchronized lock except getVal and setVal.
4. BufferMgr
    a. In BufferMgr, we can make the critical section smaller by moving the “synchronized (bufferPool)” before the statement involving “bufferPool”.
5. BufferPoolMgr
    a. Delete “synchronized” from the function flushAll and available.

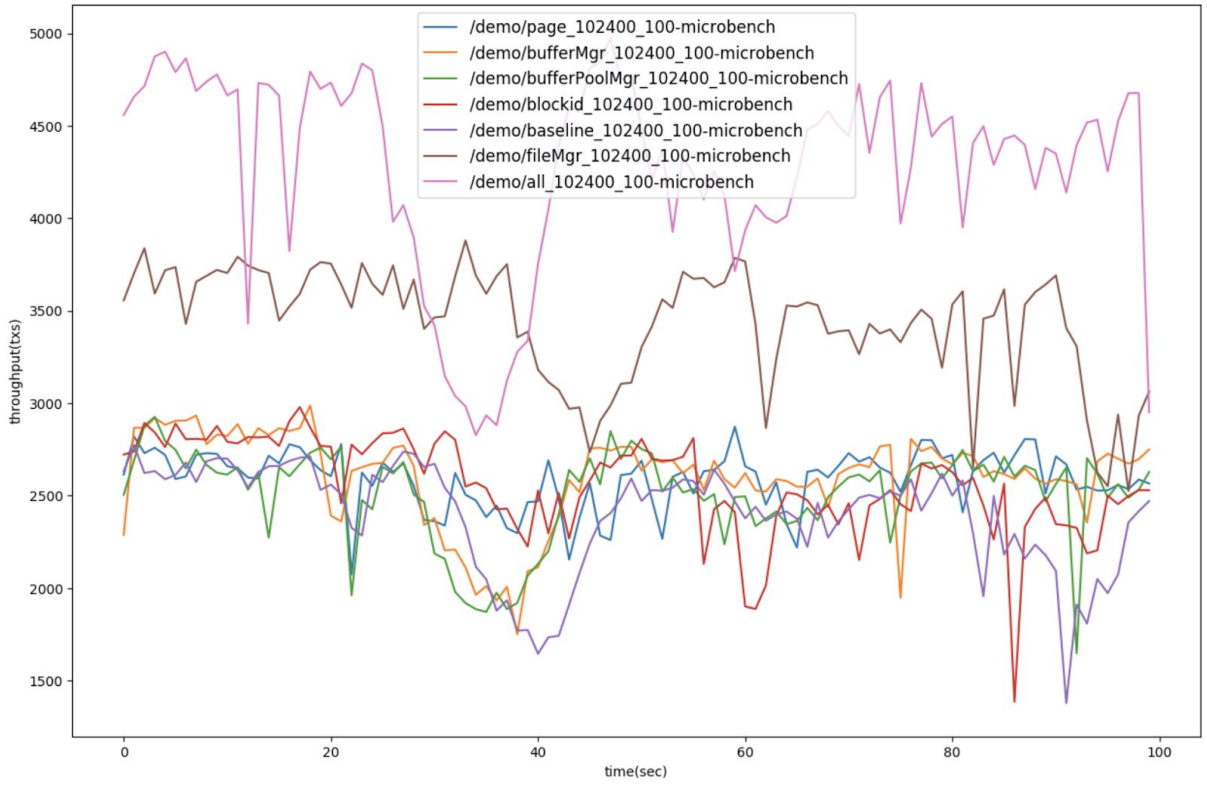
Experiments

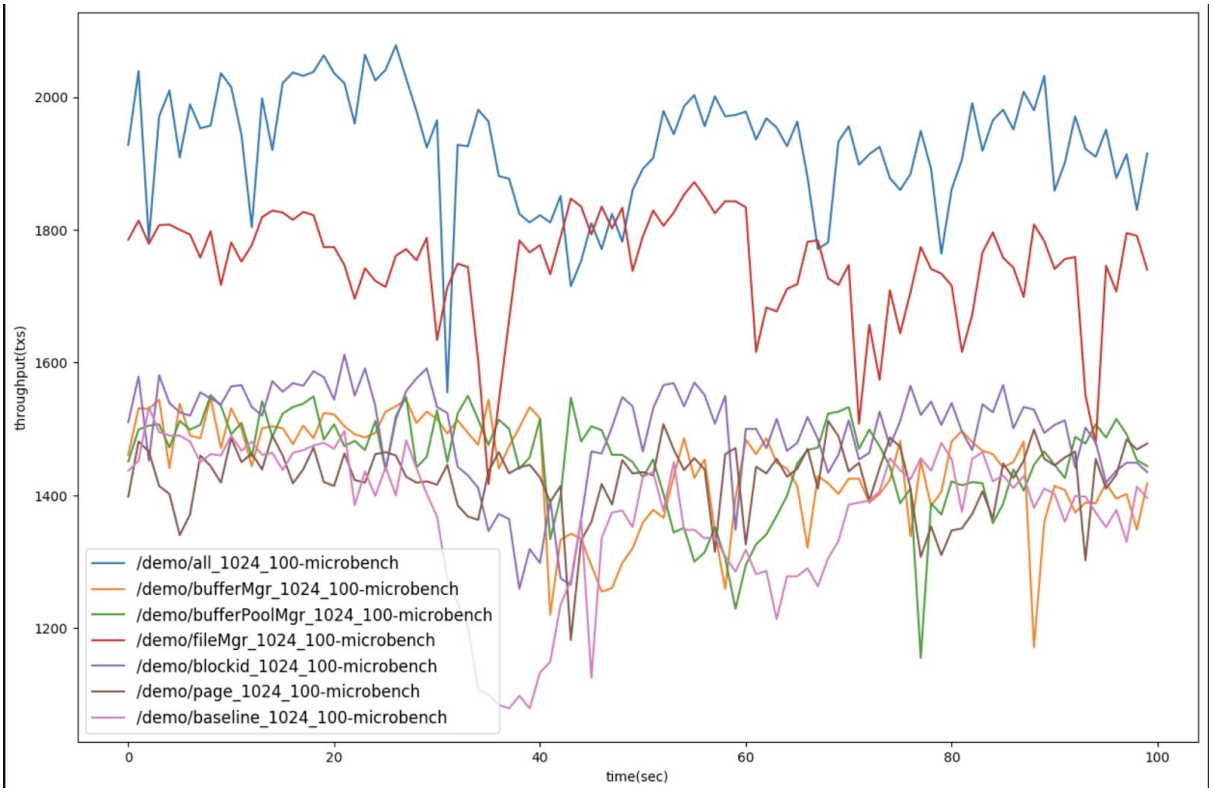
Comparison of BufferPoolsize(Microbenchmark)

```
# The running time for warming up before benchmarking
org.vanilladb.bench.BenchmarkerParameters.WARM_UP_INTERVAL=60000
# The running time for benchmarking
org.vanilladb.bench.BenchmarkerParameters.BENCHMARK_INTERVAL=100000
# The number of remote terminal executors for benchmarking
org.vanilladb.bench.BenchmarkerParameters.NUM_RTES=10
# The IP of the target database server
org.vanilladb.bench.BenchmarkerParameters.SERVER_IP=127.0.0.1
# 1 = JDBC, 2 = Stored Procedures
org.vanilladb.bench.BenchmarkerParameters.CONNECTION_MODE=2
# 1 = Micro, 2 = TPC-C,
org.vanilladb.bench.BenchmarkerParameters.BENCH_TYPE=1
# Whether it enables the built-in profiler on the server
org.vanilladb.bench.BenchmarkerParameters.PROFILING_ON_SERVER=false
# The path to the generated reports
org.vanilladb.bench.StatisticMgr.OUTPUT_DIR=/Users/cengjingyu/Desktop/experiments/microbench
# The granularity for summarizing the performance of benchmarking
org.vanilladb.bench.StatisticMgr.GRANULARITY=1000
# Whether the RTEs display the results of each transaction
org.vanilladb.bench.rte.TransactionExecutor.DISPLAY_RESULT=false


#
# Micro-benchmarks Parameters
#

# The number of items in the testing data set
org.vanilladb.bench.benchmarks.micro.MicrobenchConstants.NUM_ITEMS=100000
# The ratio of read-write transactions during benchmarking
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.RW_TX_RATE=0.5
# The ratio of long-read transactions during benchmarking
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.LONG_READ_TX_RATE=0.0
# The number of read records in a transaction
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.TOTAL_READ_COUNT=10
# The number of hot record in the read set of a transaction
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.LOCAL_HOT_COUNT=1
# The ratio of writes to the total reads of a transaction
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.WRITE_RATIO_IN_RW_TX=0.5
# The conflict rate of a hot record
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.HOT_CONFLICT_RATE=0.08
```

1. Result (Time Interval= 100s)
    a. BufferPoolSize=102400:
    Speedup=427874(commits)/238933(commits)=1.790(79%)
    b. BufferPoolSize=1024:
    Speedup=192590(commits)/136930(commits)=1.406(40.6%)

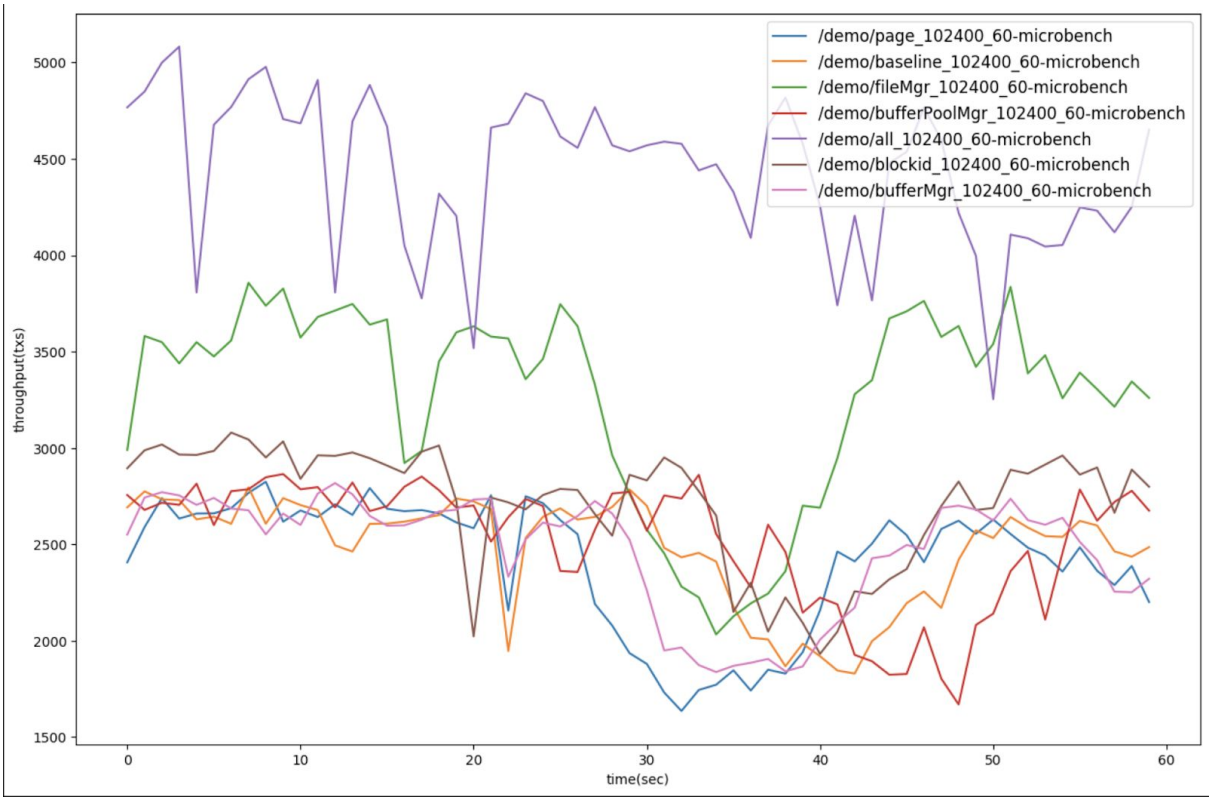
Throughput

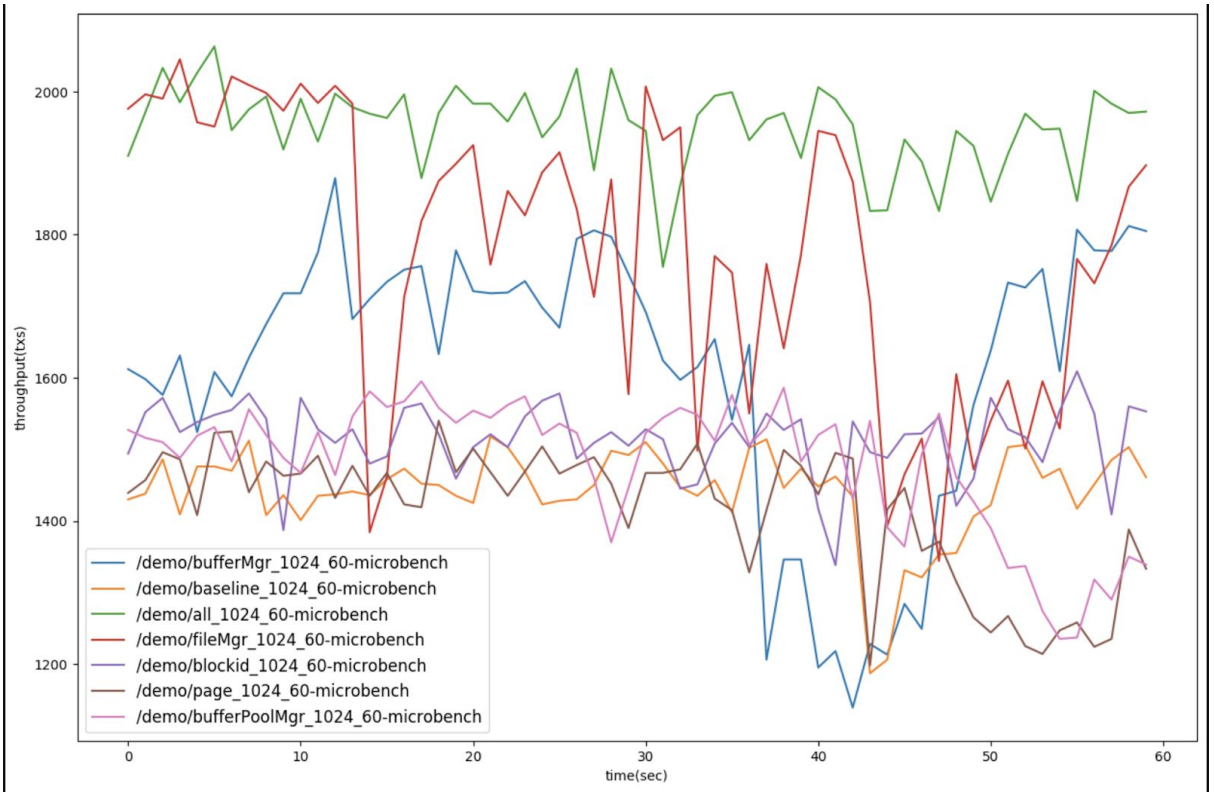




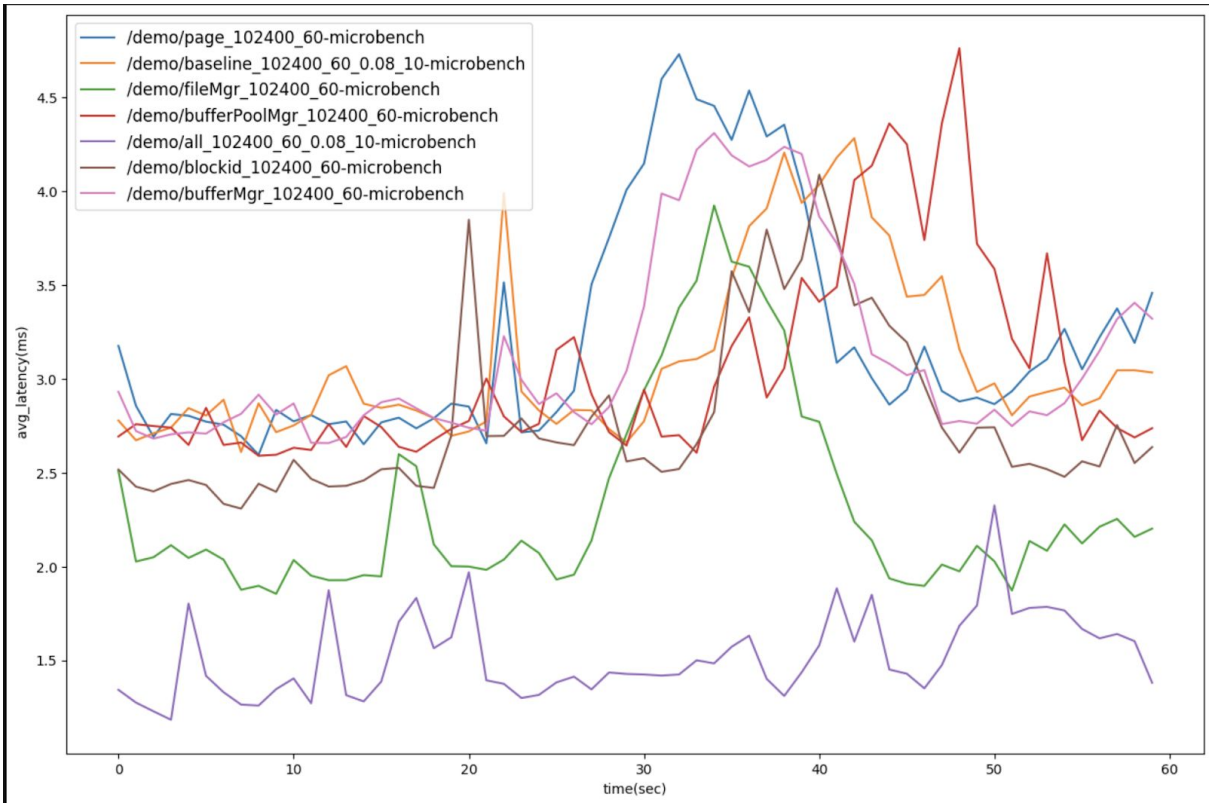
2. Result(Time Interval = 60s)
    a. BufferPoolSize=102400:
    Speedup=265853(commits)/148399(commits)=1.791(79.1%)

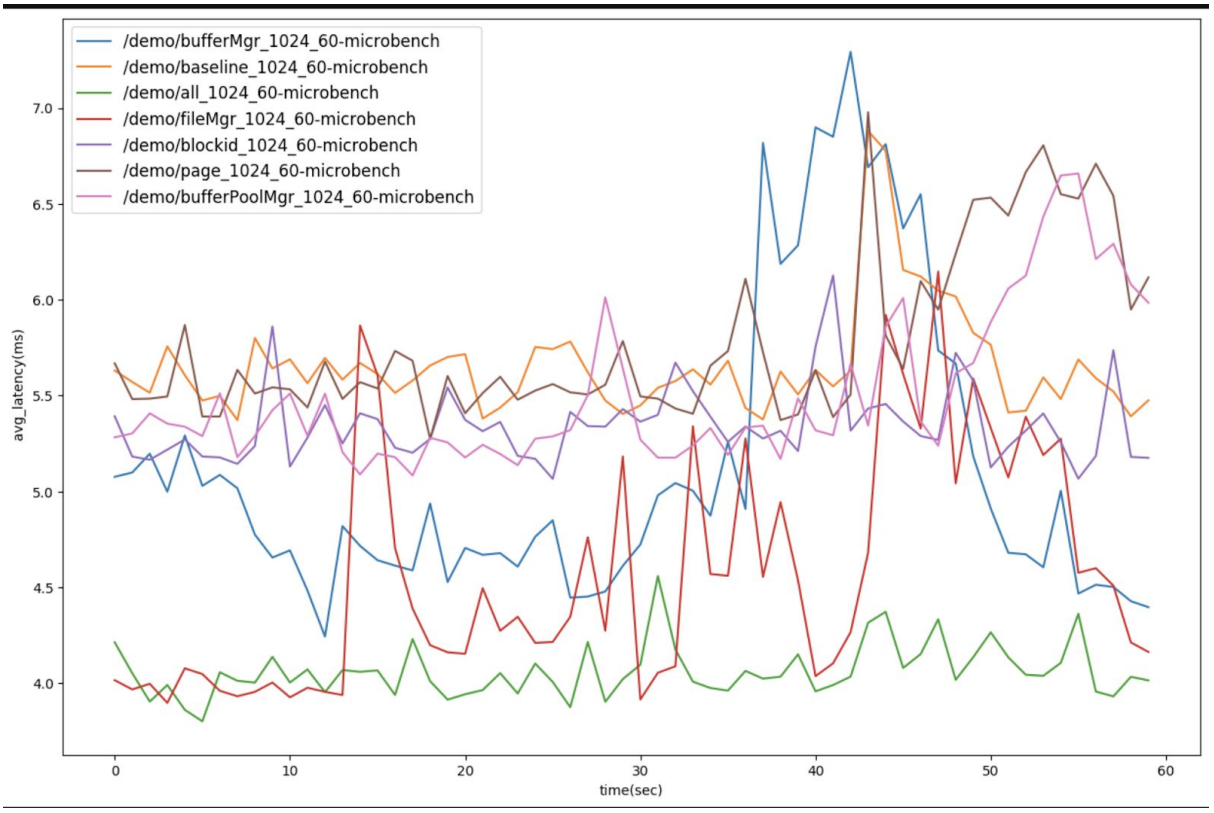
b. BufferPoolSize=1024:
Speedup=117089(commits)/86460(commits)=1.35(35%)





Latency

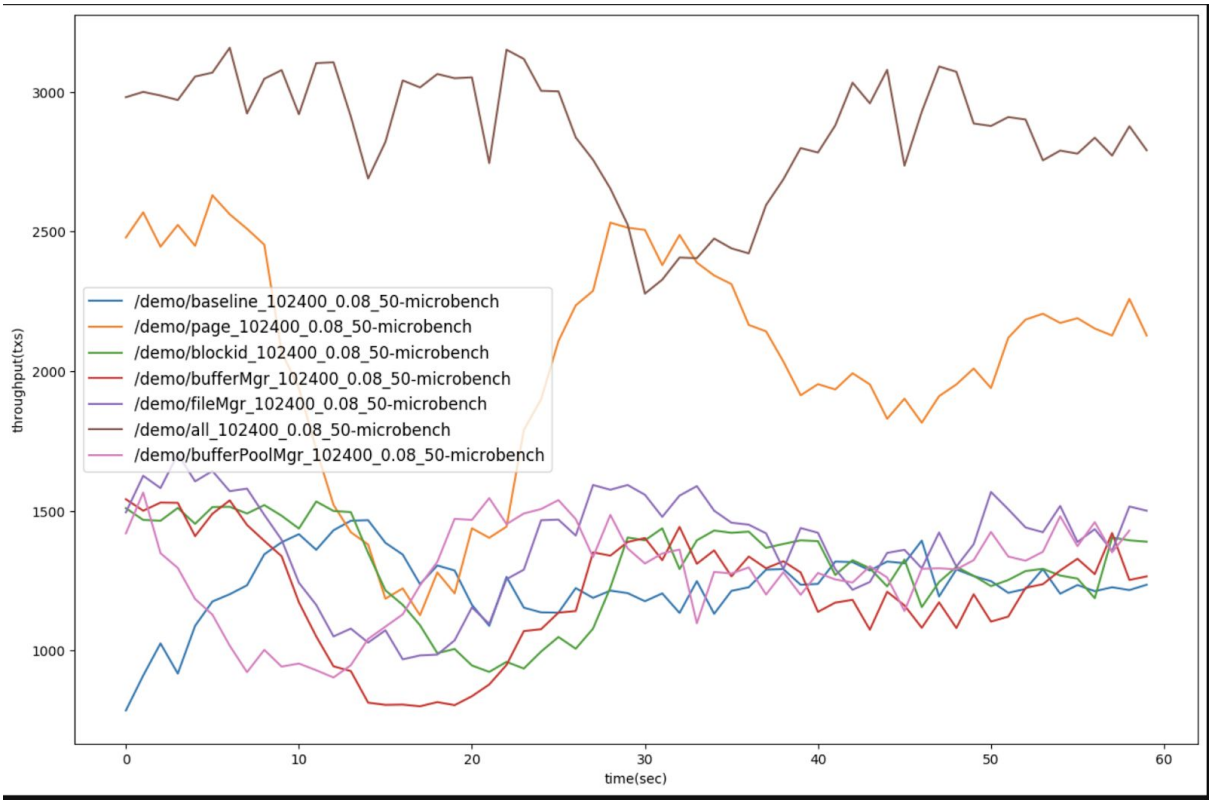

/demo/page_102400_60-microbench
/demo/baseline_102400_60_0.08_10-microbench
/demo/fileMgr_102400_60-microbench
/demo/bufferPoolMgr_102400_60-microbench
/demo/all_102400_60_0.08_10-microbench
/demo/blockid_102400_60-microbench
/demo/bufferMgr_102400_60-microbench
avg_latency(ms)
time(sec)

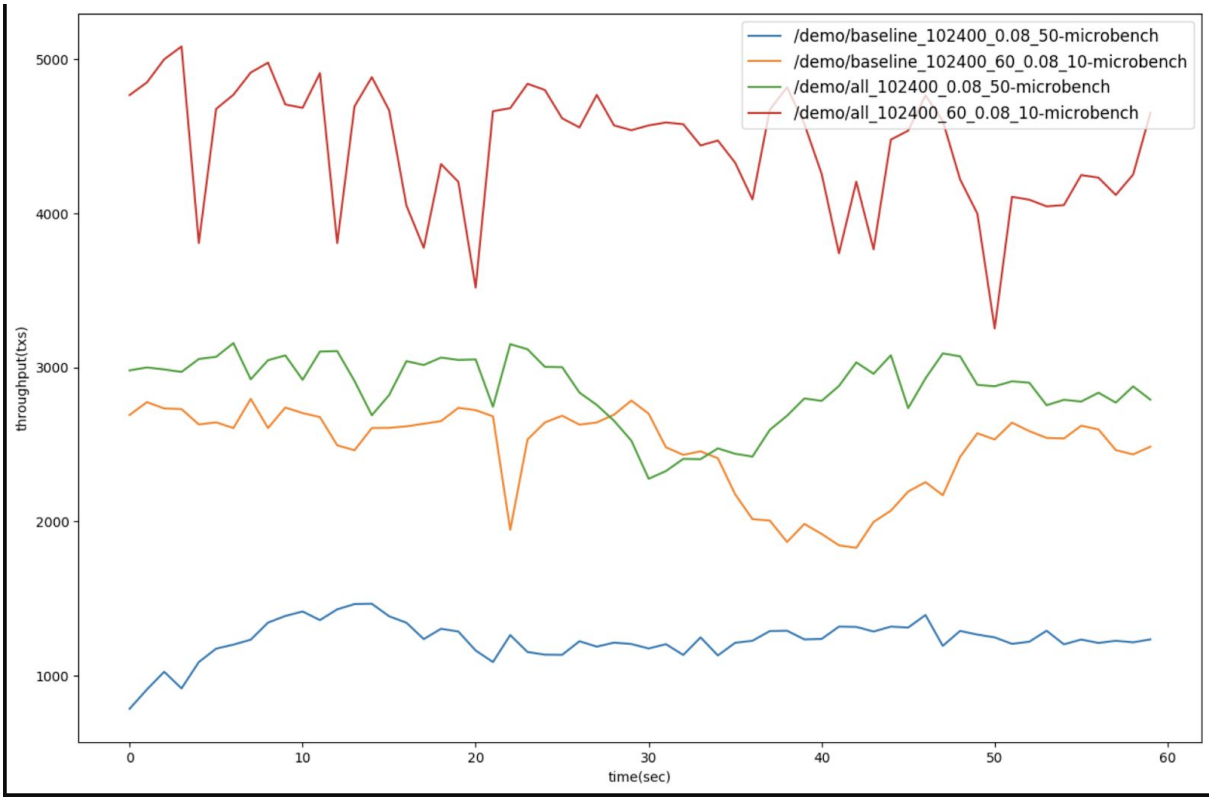

/demo/bufferMgr_1024_60-microbench
/demo/baseline_1024_60-microbench
/demo/all_1024_60-microbench
/demo/fileMgr_1024_60-microbench
/demo/blockid_1024_60-microbench
/demo/page_1024_60-microbench
/demo/bufferPoolMgr_1024_60-microbench
avg_latency(ms)
time(sec)

Comparison of NUM_RTES(Microbenchmark)

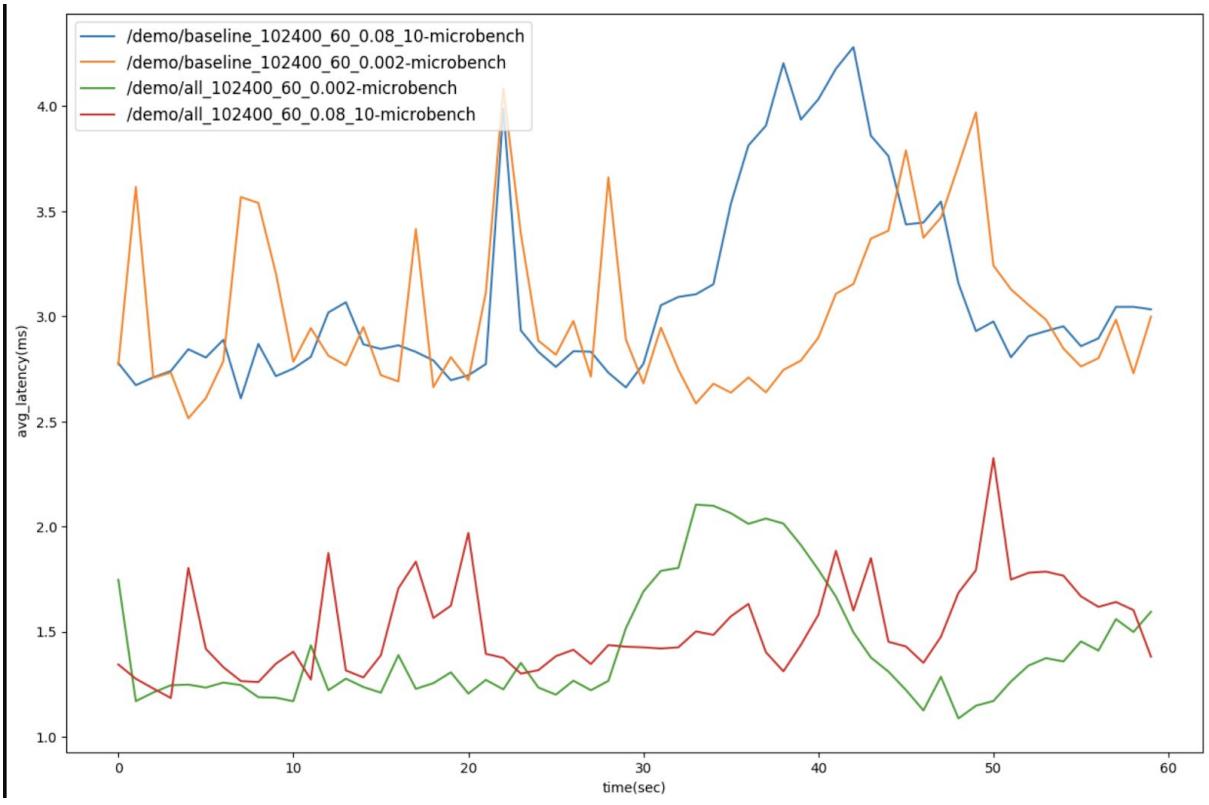
1. Result (BufferPoolSize=102400)
    a. NUM_RTES=10 :
    c. **Speedup=265853(commits)/148399(commits)=1.791(79.1%)**
    b. NUM_RTES=50 :
    Speedup=171426(commits)/73962(commits)=2.317(131.7%)

Throughput




/demo/baseline_102400_0.08_50-microbench
/demo/baseline_102400_60_0.08_10-microbench
/demo/all_102400_0.08_50-microbench
/demo/all_102400_60_0.08_10-microbench
throughput(txs)
time(sec)

Latency

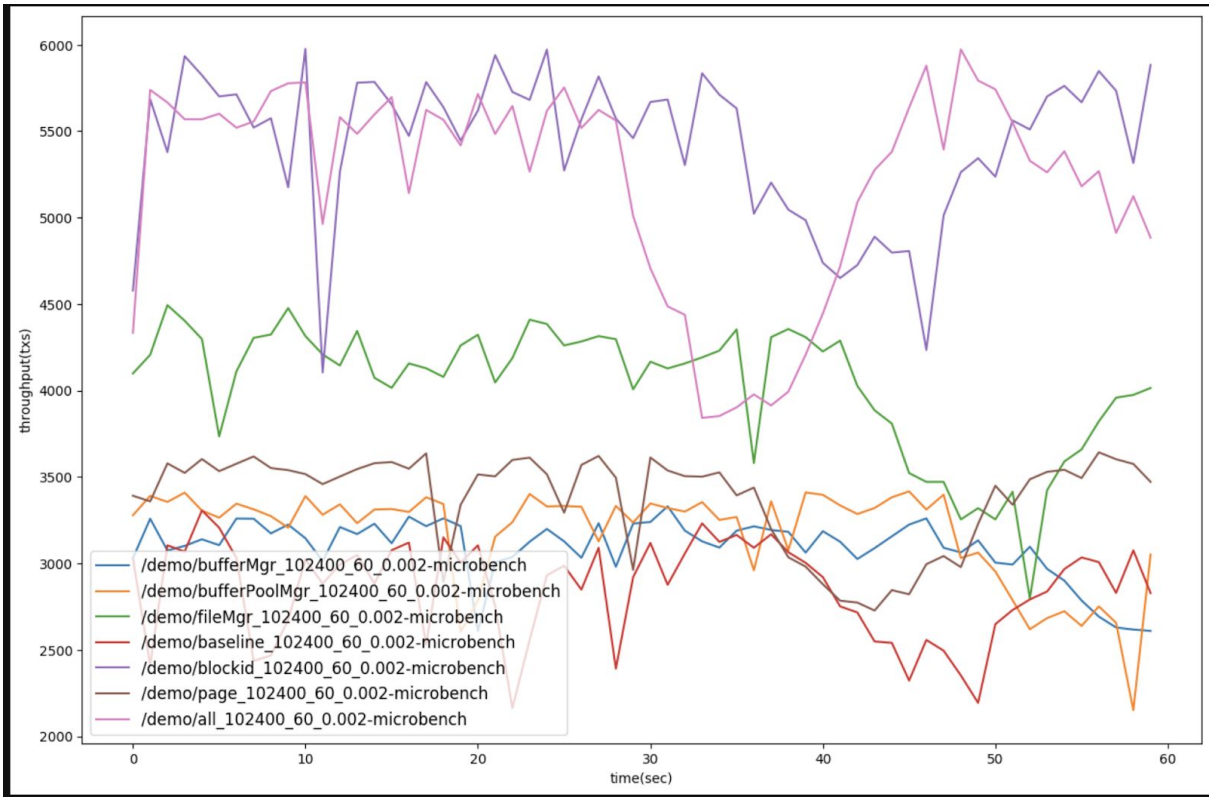

/demo/baseline_102400_60_0.08_10-microbench
/demo/baseline_102400_60_0.002-microbench
/demo/all_102400_60_0.002-microbench
/demo/all_102400_60_0.08_10-microbench
avg_latency(ms)
time(sec)

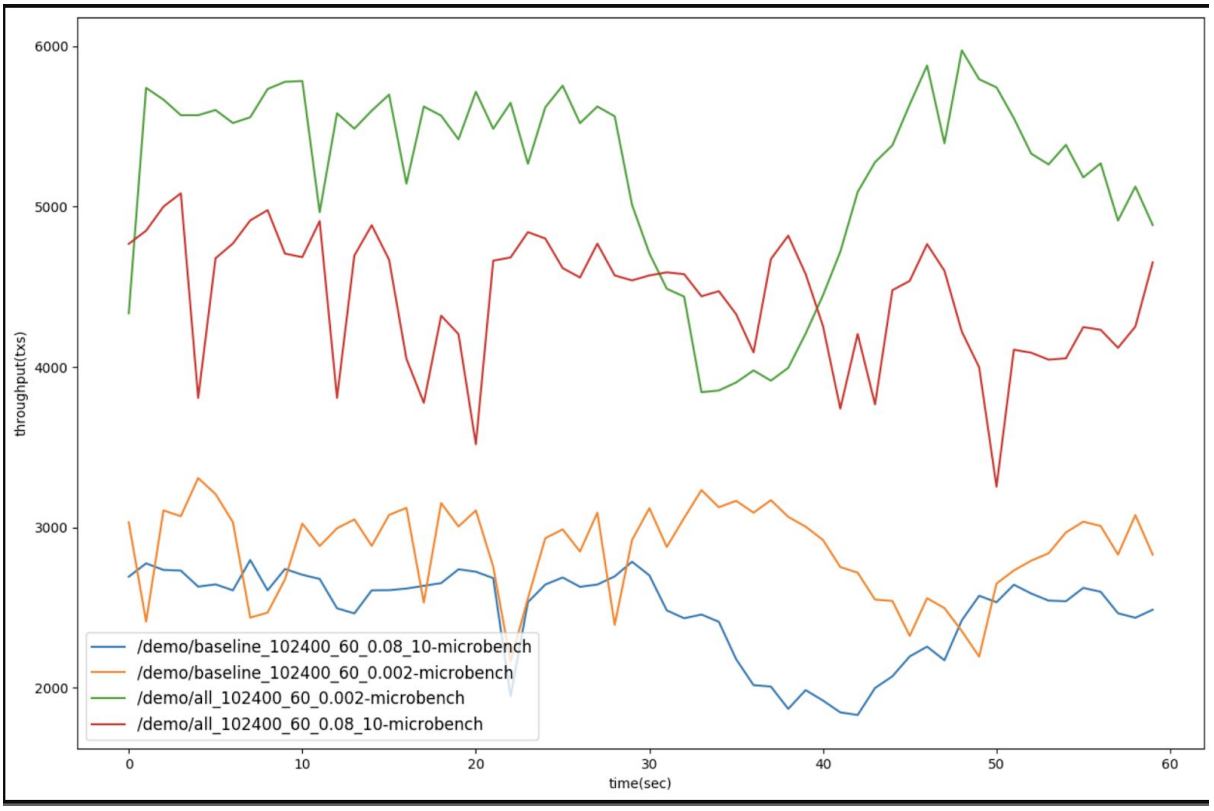
Comparison of HOT_CONFLICT_RATE(Microbenchmark)

```
# The running time for warming up before benchmarking
org.vanilladb.bench.BenchmarkerParameters.WARM_UP_INTERVAL=60000
# The running time for benchmarking
org.vanilladb.bench.BenchmarkerParameters.BENCHMARK_INTERVAL=60000
# The number of remote terminal executors for benchmarking
org.vanilladb.bench.BenchmarkerParameters.NUM_RTES=10
# The IP of the target database server
org.vanilladb.bench.BenchmarkerParameters.SERVER_IP=127.0.0.1
# 1 = JDBC, 2 = Stored Procedures
org.vanilladb.bench.BenchmarkerParameters.CONNECTION_MODE=2
# 1 = Micro, 2 = TPC-C,
org.vanilladb.bench.BenchmarkerParameters.BENCH_TYPE=1
# Whether it enables the built-in profiler on the server
org.vanilladb.bench.BenchmarkerParameters.PROFILING_ON_SERVER=false
# The path to the generated reports
org.vanilladb.bench.StatisticMgr.OUTPUT_DIR=/Users/cengjingyu/Desktop/experiments/microbench
# The granularity for summarizing the performance of benchmarking
org.vanilladb.bench.StatisticMgr.GRANULARITY=1000
# Whether the RTEs display the results of each transaction
org.vanilladb.bench.rte.TransactionExecutor.DISPLAY_RESULT=false


#
# Micro-benchmarks Parameters
#

# The number of items in the testing data set
org.vanilladb.bench.benchmarks.micro.MicrobenchConstants.NUM_ITEMS=100000
# The ratio of read-write transactions during benchmarking
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.RW_TX_RATE=0.5
# The ratio of long-read transactions during benchmarking
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.LONG_READ_TX_RATE=0.0
# The number of read records in a transaction
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.TOTAL_READ_COUNT=10
# The number of hot record in the read set of a transaction
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.LOCAL_HOT_COUNT=1
# The ratio of writes to the total reads of a transaction
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.WRITE_RATIO_IN_RW_TX=0.5
# The conflict rate of a hot record
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.HOT_CONFLICT_RATE=0.002
```

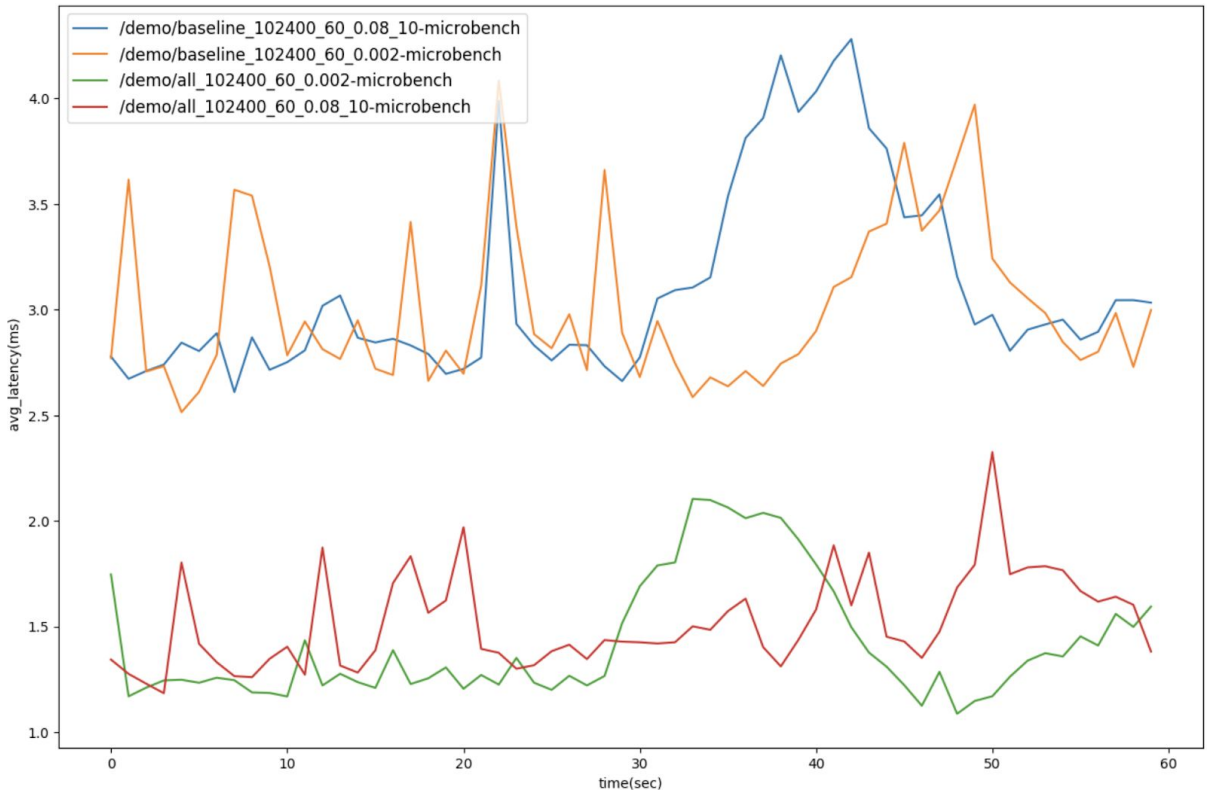
1. Result (BufferPoolSize=102400)
    a. HOT_CONFLICT_RATE=0.08 :
    d. **Speedup=265853(commits)/148399(commits)=1.791(79.1%)**
    b. HOT_CONFLICT_RATE=0.002:
    Speedup=312686(commits)/171295(commits)=1.825(82.5%)

Throughput


throughput(txs)
time(sec)
/demo/bufferMgr_102400_60_0.002-microbench
/demo/bufferPoolMgr_102400_60_0.002-microbench
/demo/fileMgr_102400_60_0.002-microbench
/demo/baseline_102400_60_0.002-microbench
/demo/blockid_102400_60_0.002-microbench
/demo/page_102400_60_0.002-microbench
/demo/all_102400_60_0.002-microbench


throughput(txs)
time(sec)
/demo/baseline_102400_60_0.08_10-microbench
/demo/baseline_102400_60_0.002-microbench
/demo/all_102400_60_0.002-microbench
/demo/all_102400_60_0.08_10-microbench

Latency


/demo/baseline_102400_60_0.08_10-microbench
/demo/baseline_102400_60_0.002-microbench
/demo/all_102400_60_0.002-microbench
/demo/all_102400_60_0.08_10-microbench
avg_latency(ms)
time(sec)
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
10
20
30
40
50
60

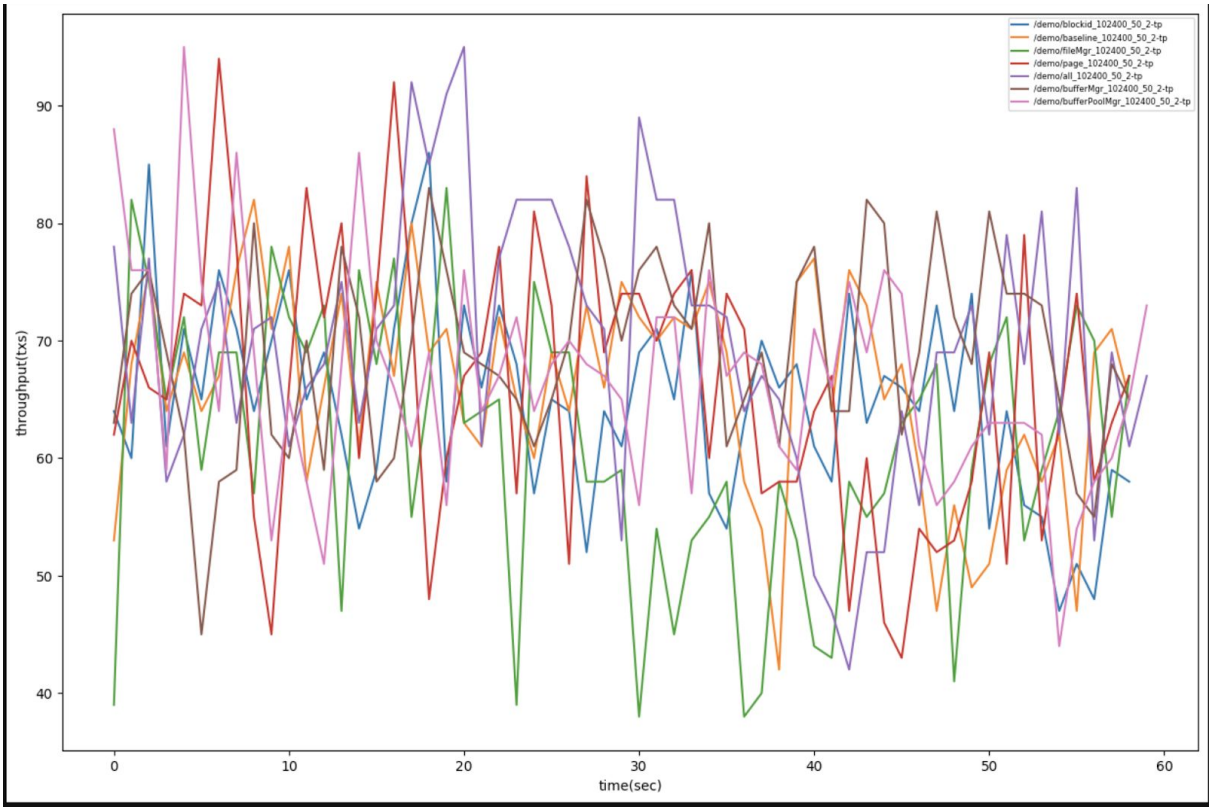
Comparison of NUM_WAREHOUSES(TPCC)

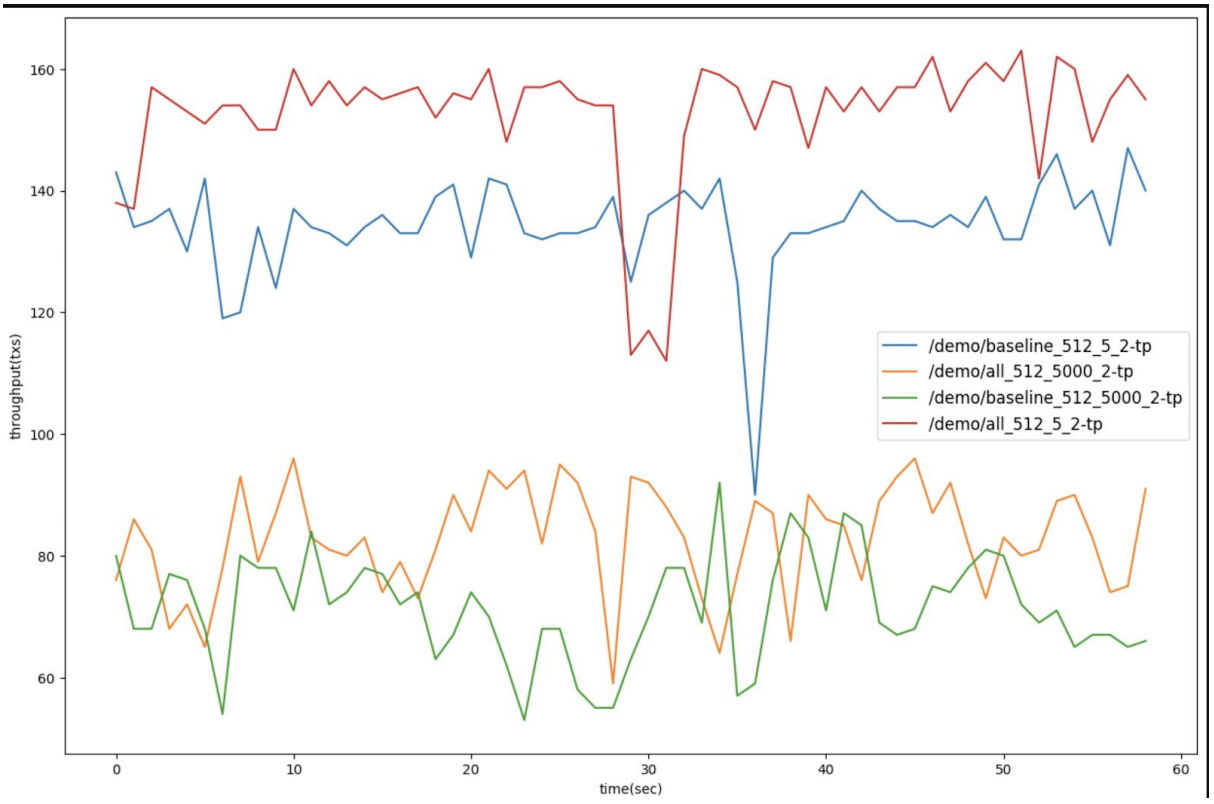
```
#
# Basic Parameters
#

# The running time for warming up before benchmarking
org.vanilladb.bench.BenchmarkerParameters.WARM_UP_INTERVAL=60000
# The running time for benchmarking
org.vanilladb.bench.BenchmarkerParameters.BENCHMARK_INTERVAL=60000
# The number of remote terminal executors for benchmarking
org.vanilladb.bench.BenchmarkerParameters.NUM_RTES=2
# The IP of the target database server
org.vanilladb.bench.BenchmarkerParameters.SERVER_IP=127.0.0.1
# 1 = JDBC, 2 = Stored Procedures
org.vanilladb.bench.BenchmarkerParameters.CONNECTION_MODE=2
# 1 = Micro, 2 = TPC-C,
org.vanilladb.bench.BenchmarkerParameters.BENCH_TYPE=2
# Whether it enables the built-in profiler on the server
org.vanilladb.bench.BenchmarkerParameters.PROFILING_ON_SERVER=false
# The path to the generated reports
org.vanilladb.bench.StatisticMgr.OUTPUT_DIR=/Users/cengjingyu/Desktop/experiments/TPCC
# The granularity for summarizing the performance of benchmarking
org.vanilladb.bench.StatisticMgr.GRANULARITY=1000
# Whether the RTEs display the results of each transaction
org.vanilladb.bench.rte.TransactionExecutor.DISPLAY_RESULT=false
```

```
#
# TPC-C Parameters
#

# The number of warehouses in the testing data set
org.vanilladb.bench.benchmarks.tpcc.TpccConstants.NUM_WAREHOUSES=5
# The total number of frequency
org.vanilladb.bench.benchmarks.tpcc.TpccConstants.FREQUENCY_TOTAL=100
# The frequency of new-order transactions
org.vanilladb.bench.benchmarks.tpcc.TpccConstants.FREQUENCY_NEW_ORDER=100
# The frequency of payment transactions
org.vanilladb.bench.benchmarks.tpcc.TpccConstants.FREQUENCY_PAYMENT=0
# The frequency of order-status transactions
# XXX: Not implemented
org.vanilladb.bench.benchmarks.tpcc.TpccConstants.FREQUENCY_ORDER_STATUS=0
# The frequency of delivery transactions
# XXX: Not implemented
org.vanilladb.bench.benchmarks.tpcc.TpccConstants.FREQUENCY_DELIVERY=0
# The frequency of stock-level transactions
# XXX: Not implemented
org.vanilladb.bench.benchmarks.tpcc.TpccConstants.FREQUENCY_STOCK_LEVEL=0
# Whether it enables the thinking and keying time defined in TPC-C specification
org.vanilladb.bench.benchmarks.tpcc.rte.TpccTxExecutor.ENABLE_THINK_AND_KEYING_TIME=fals
```

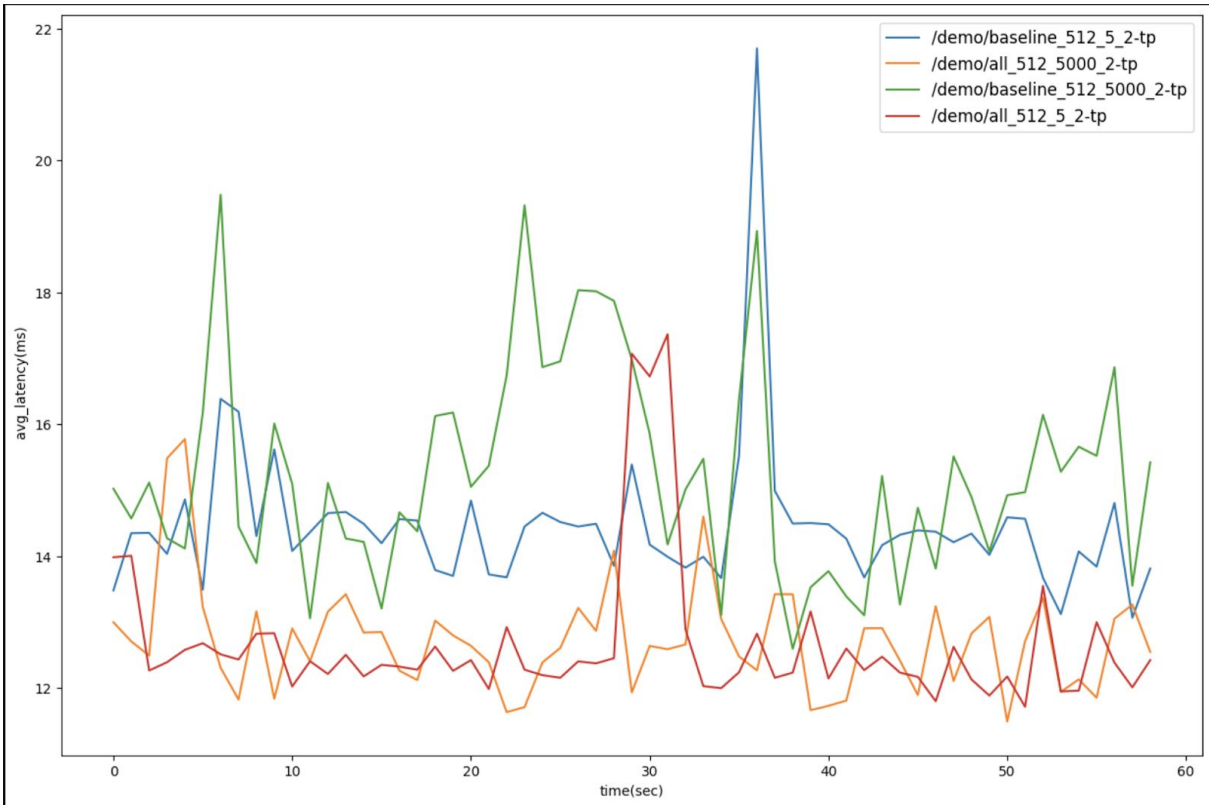
1. Result (BufferPoolSize=512)
    a. NUM_WAREHOUSES=5 :
    Speedup=9124(commits)/8095(commits)=1.127(12.7%)
    b. NUM_WAREHOUSES=5000:
    Speedup=4981(commits)/4272(commits)=1.165(16.5%)

Throughput


/demo/blockid_102400_50_2-tp
/demo/baseline_102400_50_2-tp
/demo/fileMgr_102400_50_2-tp
/demo/page_102400_50_2-tp
/demo/all_102400_50_2-tp
/demo/bufferMgr_102400_50_2-tp
/demo/bufferPoolMgr_102400_50_2-tp
throughput(txs)
time(sec)


/demo/baseline_512_5_2-tp
/demo/all_512_5000_2-tp
/demo/baseline_512_5000_2-tp
/demo/all_512_5_2-tp
throughput(txs)
time(sec)

Latency



Analysis

Smaller Critical Section

Using smaller critical section can improve the performance of DBMS. We try to reduce the regions of critical sections. After tracing codes, we found that there are several critical sections in files.(BufferMgr, BufferPoolMgr, Page, FileMgr). To being with, in order to realize multi-threading, we can rescind different levels' "synchronize" parameters. It mainly can be divided into three parts. Buffer level, Page level and FileMgr level. Initially, all levels' functions are with "synchronize" parameters. We try to reduce the size of critical sections, the performance became better, due to maintaining data consistency, we can modify many functions. Then, we try to decrease the size of Page, FileMgr levels, since we maintains data consistancy of threads in buffer level, we can repeal most parts of each level. Finally the performance becomes much better.

Never Do It Again

When using pinningbuffer hashmap, hashcode() method will be called to calculate the hash value of the blockID. In origin version, the value has to be calculated whenever it is called. We add a variable to store the calculated value when the method is first called. So afterward, it can instantly return the stored value.

Size of BufferPool

In our experiments, we choose two buffer pool size to implement, which is 1024 and 102400. According to the result, when using larger buffer pool size, our optimization has more significant improvement on performance. We guess it is because our optimization is done by decreasing critical section and reducing duplicated calculation, but not replacement strategy. So the improvement on performance when using smaller buffer pool will be constrained by page swapping.

Num of RTEs

We compare two experiments, which has 10 RTEs, 50 RTEs respectively. Result shows that the optimization is better when having 50 RTEs. We think it is because when more clients are accessing a DBMS concurrently, our optimization on concurrency (smaller critical section) will be more effective.

Rate of Conflicts

To find out the influence of conflict rate on optimization performance, we compare two conflict rate, 0.002 and 0.08. The result shows that with conflict rate 0.002, the performance (throughput and latency) and the improvement ratio is a better than with conflict rate 0.08. We believe the reason is that the smaller the conflict rate is, the less possible that clients will access the same block. Accessing the same block would be blocked since all buffer method are synchronized. So smaller conflict rate leads to better performance.

TPC-C

When using TPC-C benchmarking, we first want to use the same bufferpool size like micro benchmark. However, since TPC-C database is large, it would run out of memory. So we finally choose 512 as our bufferpool size. Unfortunately, due to the small size of bufferpool, many transaction is aborted by deadlock. In both two experiment with warehouse number 5 and 5000, the performance and improvement are bad compared to micro benchmarking.