CS6220-Assignment4
Problem3. Hand-on Experience with a key-value system

Bin Xie

Overview

For this assignment, I choose to combine the Problem3.1 Performance Measurement of your favorite Key-Value store and Problem3.2 Performance Comparison of two Key-Value Systems.

For the two types of key-value stores, I choose: Redis (http://redis.io) and RocksDB (http://rocksdb.org). Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. RocksDB is an embeddable persistent key-value store for fast storage.

For the workload benchmarks, I choose the YCSB (Yahoo! Cloud Serving Benchmark). YCSB project is a framework and common set of workloads for evaluating the performance of different “key-value” and “cloud” serving stores. Workloads have two executable phases: the loading phase and the transactions phase. Also, it provides six different core workloads: (A) Update heavy workload (B) Read mostly workload (C) Read only workload (D) Read latest workload (E) Short ranges, and (F) Read-modify-write. I will run all these workloads to understand the performance tradeoffs of these two key-value stores.

For the different scales of datasets, I will compare the key-value system performance when the dataset size is increasing from 1000, 3000, 6000, 10000.

Workloads running process:

To run bulk loading of dataset and all the six core workloads with the consistent database size, I follow the sequence provided by YSCB:

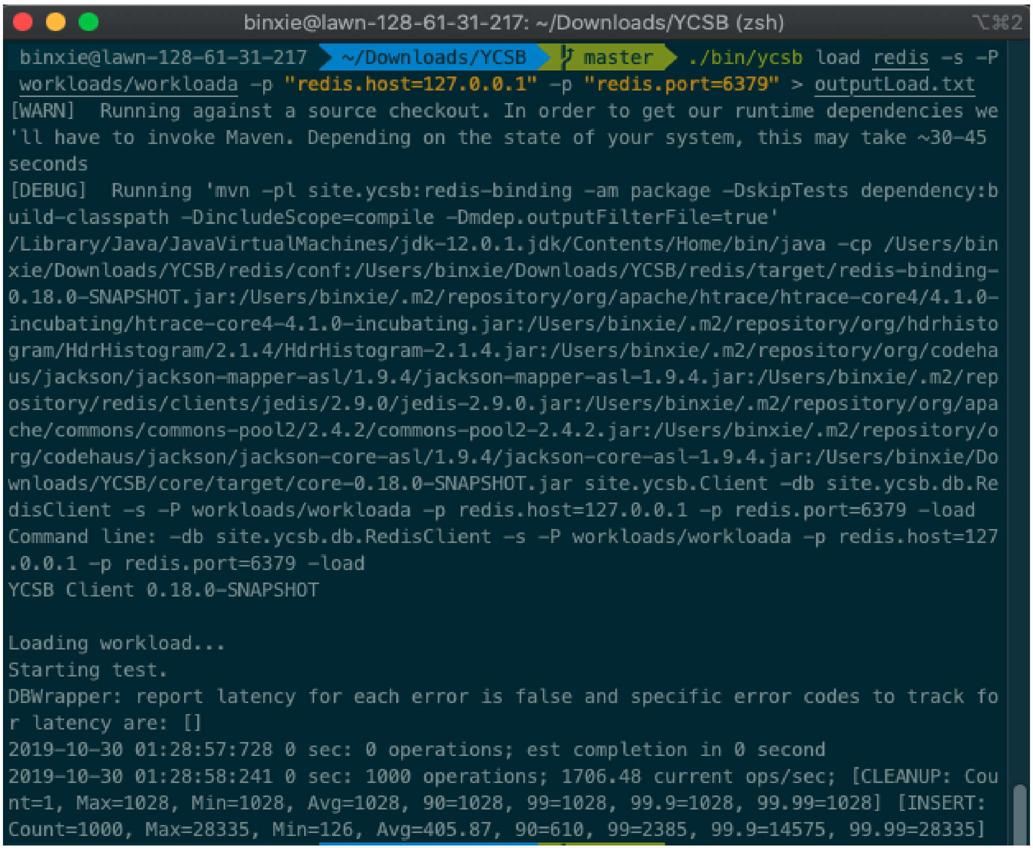
- Load the database, using workload A’s parameter file and the “-load”
- Run workload A for a variety of throughputs.
- Run workload B for a variety of throughputs.
- Run workload C for a variety of throughputs.
- Run workload F for a variety of throughputs.
- Run workload D for a variety of throughputs. This workload inserts records, increasing the size of the database.
- Delete the data in the database.
- Reload the database, using workload E’s parameter file and the “-load”
- Run workload E for a variety of throughputs. This workload inserts records, increasing the size of the database.

Experience with installation, running and measurement

To run these workloads, for YCSB, the environment should include: Java, Maven, Python2.7. To run workloads on Redis, I download the Redis from github, install it and start the server. To run workloads on RocksDB, YCSB already includes the RocksDB in it.

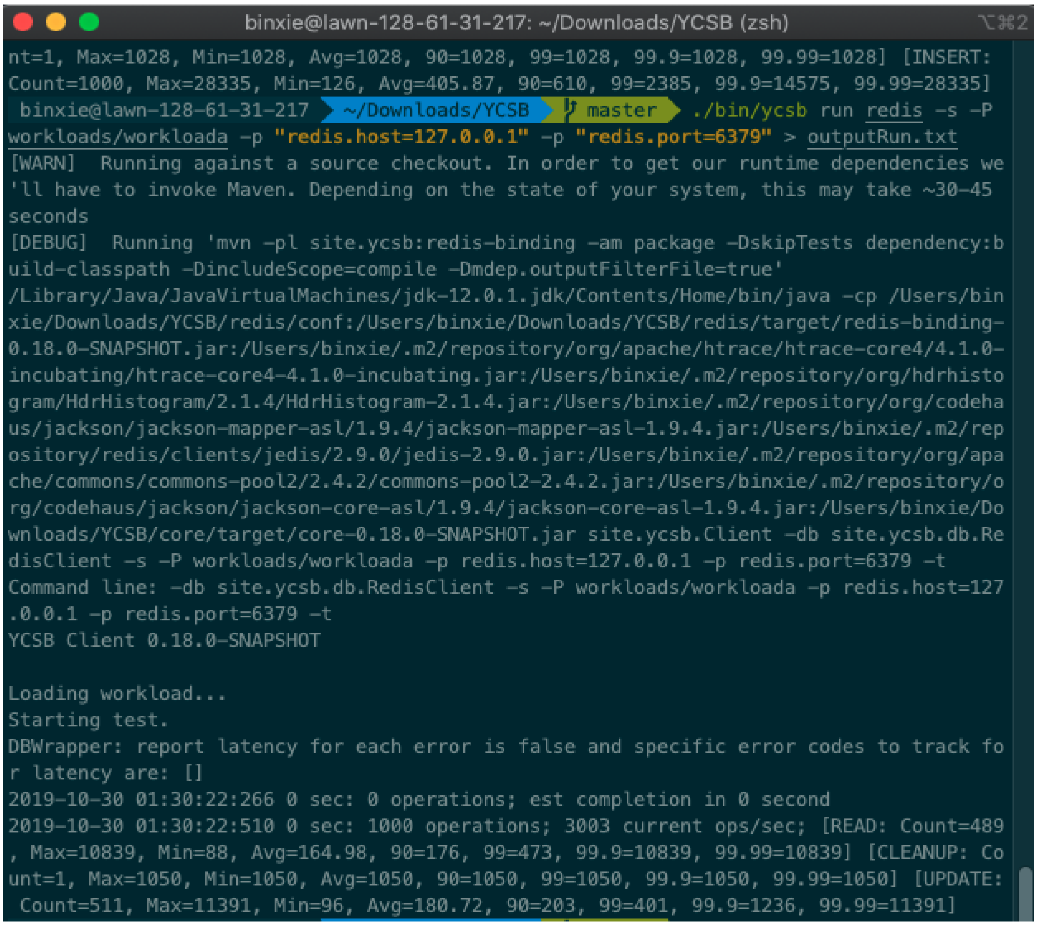
Basically, there are two phases for running: load the data and run the workload test.

Execution process Screenshots



```
 binxie@lawn-128-61-31-217  ~/Downloads/YCSB   master  ./bin/ycsb load redis -s -P
 workloads/workloada -p "redis.host=127.0.0.1" -p "redis.port=6379" > outputLoad.txt
[WARN]  Running against a source checkout. In order to get our runtime dependencies we
'll have to invoke Maven. Depending on the state of your system, this may take ~30-45
seconds
[DEBUG]  Running 'mvn -pl site.ycsb:redis-binding -am package -DskipTests dependency:b
uild-classpath -DincludeScope=compile -Dmdep.outputFilterFile=true'
/Library/Java/JavaVirtualMachines/jdk-12.0.1.jdk/Contents/Home/bin/java -cp /Users/bin
xie/Downloads/YCSB/redis/conf:/Users/binxie/Downloads/YCSB/redis/target/redis-binding-
0.18.0-SNAPSHOT.jar:/Users/binxie/.m2/repository/org/apache/htrace/htrace-core4/4.1.0-
incubating/htrace-core4-4.1.0-incubating.jar:/Users/binxie/.m2/repository/org/hdrhisto
gram/HdrHistogram/2.1.4/HdrHistogram-2.1.4.jar:/Users/binxie/.m2/repository/org/codeha
us/jackson/jackson-mapper-asl/1.9.4/jackson-mapper-asl-1.9.4.jar:/Users/binxie/.m2/rep
ository/redis/clients/jedis/2.9.0/jedis-2.9.0.jar:/Users/binxie/.m2/repository/org/apa
che/commons/commons-pool2/2.4.2/commons-pool2-2.4.2.jar:/Users/binxie/.m2/repository/o
rg/codehaus/jackson/jackson-core-asl/1.9.4/jackson-core-asl-1.9.4.jar:/Users/binxie/Do
wnloads/YCSB/core/target/core-0.18.0-SNAPSHOT.jar site.ycsb.Client -db site.ycsb.db.Re
disClient -s -P workloads/workloada -p redis.host=127.0.0.1 -p redis.port=6379 -load
Command line: -db site.ycsb.db.RedisClient -s -P workloads/workloada -p redis.host=127
.0.0.1 -p redis.port=6379 -load
YCSB Client 0.18.0-SNAPSHOT

Loading workload...
Starting test.
DBWrapper: report latency for each error is false and specific error codes to track fo
r latency are: []
2019-10-30 01:28:57:728 0 sec: 0 operations; est completion in 0 second
2019-10-30 01:28:58:241 0 sec: 1000 operations; 1706.48 current ops/sec; [CLEANUP: Cou
nt=1, Max=1028, Min=1028, Avg=1028, 90=1028, 99=1028, 99.9=1028, 99.99=1028] [INSERT:
Count=1000, Max=28335, Min=126, Avg=405.87, 90=610, 99=2385, 99.9=14575, 99.99=28335]
```



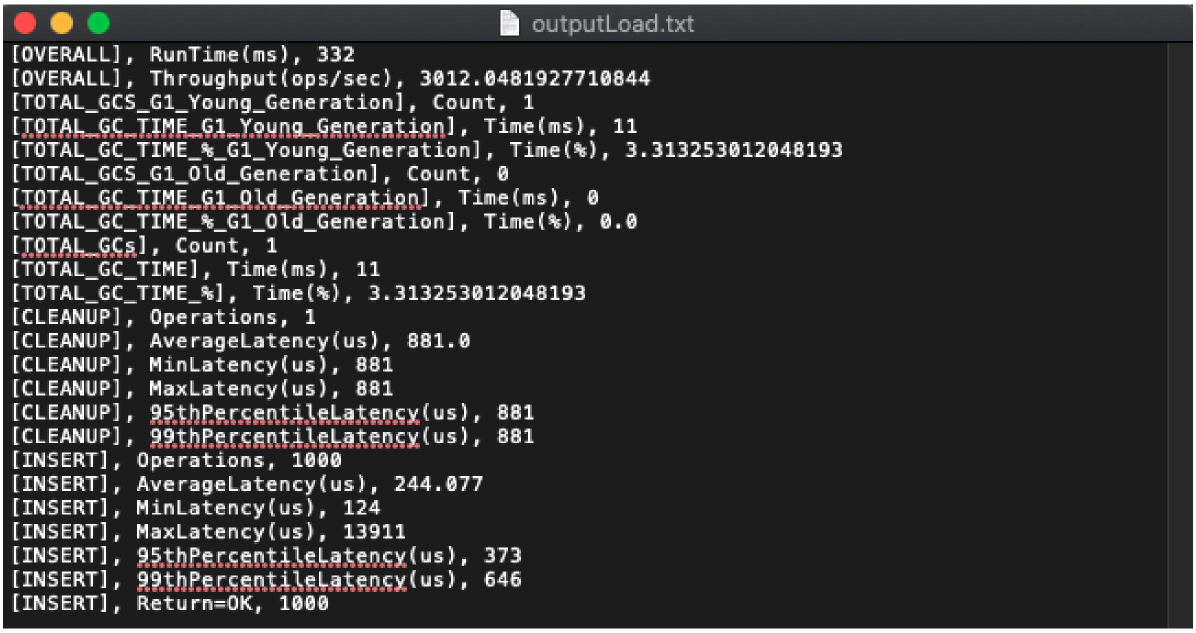
```
nt=1, Max=1028, Min=1028, Avg=1028, 90=1028, 99=1028, 99.9=1028, 99.99=1028] [INSERT:
Count=1000, Max=28335, Min=126, Avg=405.87, 90=610, 99=2385, 99.9=14575, 99.99=28335]
 binxie@lawn-128-61-31-217  ~/Downloads/YCSB   master  ./bin/ycsb run redis -s -P
workloads/workloada -p "redis.host=127.0.0.1" -p "redis.port=6379" > outputRun.txt
[WARN]  Running against a source checkout. In order to get our runtime dependencies we
'll have to invoke Maven. Depending on the state of your system, this may take ~30-45
seconds
[DEBUG]  Running 'mvn -pl site.ycsb:redis-binding -am package -DskipTests dependency:b
uild-classpath -DincludeScope=compile -Dmdep.outputFilterFile=true'
/Library/Java/JavaVirtualMachines/jdk-12.0.1.jdk/Contents/Home/bin/java -cp /Users/bin
xie/Downloads/YCSB/redis/conf:/Users/binxie/Downloads/YCSB/redis/target/redis-binding-
0.18.0-SNAPSHOT.jar:/Users/binxie/.m2/repository/org/apache/htrace/htrace-core4/4.1.0-
incubating/htrace-core4-4.1.0-incubating.jar:/Users/binxie/.m2/repository/org/hdrhisto
gram/HdrHistogram/2.1.4/HdrHistogram-2.1.4.jar:/Users/binxie/.m2/repository/org/codeha
us/jackson/jackson-mapper-asl/1.9.4/jackson-mapper-asl-1.9.4.jar:/Users/binxie/.m2/rep
ository/redis/clients/jedis/2.9.0/jedis-2.9.0.jar:/Users/binxie/.m2/repository/org/apa
che/commons/commons-pool2/2.4.2/commons-pool2-2.4.2.jar:/Users/binxie/.m2/repository/o
rg/codehaus/jackson/jackson-core-asl/1.9.4/jackson-core-asl-1.9.4.jar:/Users/binxie/Do
wnloads/YCSB/core/target/core-0.18.0-SNAPSHOT.jar site.ycsb.Client -db site.ycsb.db.Re
disClient -s -P workloads/workloada -p redis.host=127.0.0.1 -p redis.port=6379 -t
Command line: -db site.ycsb.db.RedisClient -s -P workloads/workloada -p redis.host=127
.0.0.1 -p redis.port=6379 -t
YCSB Client 0.18.0-SNAPSHOT

Loading workload...
Starting test.
DBWrapper: report latency for each error is false and specific error codes to track fo
r latency are: []
2019-10-30 01:30:22:266 0 sec: 0 operations; est completion in 0 second
2019-10-30 01:30:22:510 0 sec: 1000 operations; 3003 current ops/sec; [READ: Count=489
, Max=10839, Min=88, Avg=164.98, 90=176, 99=473, 99.9=10839, 99.99=10839] [CLEANUP: Co
unt=1, Max=1050, Min=1050, Avg=1050, 90=1050, 99=1050, 99.9=1050, 99.99=1050] [UPDATE:
 Count=511, Max=11391, Min=96, Avg=180.72, 90=203, 99=401, 99.9=1236, 99.99=11391]
```

Bulk loading of dataset

The bulk loading of dataset defines the data needs to be inserted into key-value store.

Here is the sample output of results for bulk loading of data by Redis with data size 1000.



```
[OVERALL], RunTime(ms), 332
[OVERALL], Throughput(ops/sec), 3012.0481927710844
[TOTAL_GCS_G1_Young_Generation], Count, 1
[TOTAL_GC_TIME_G1_Young_Generation], Time(ms), 11
[TOTAL_GC_TIME_%_G1_Young_Generation], Time(%), 3.313253012048193
[TOTAL_GCS_G1_Old_Generation], Count, 0
[TOTAL_GC_TIME_G1_Old_Generation], Time(ms), 0
[TOTAL_GC_TIME_%_G1_Old_Generation], Time(%), 0.0
[TOTAL_GCs], Count, 1
[TOTAL_GC_TIME], Time(ms), 11
[TOTAL_GC_TIME_%], Time(%), 3.313253012048193
[CLEANUP], Operations, 1
[CLEANUP], AverageLatency(us), 881.0
[CLEANUP], MinLatency(us), 881
[CLEANUP], MaxLatency(us), 881
[CLEANUP], 95thPercentileLatency(us), 881
[CLEANUP], 99thPercentileLatency(us), 881
[INSERT], Operations, 1000
[INSERT], AverageLatency(us), 244.077
[INSERT], MinLatency(us), 124
[INSERT], MaxLatency(us), 13911
[INSERT], 95thPercentileLatency(us), 373
[INSERT], 99thPercentileLatency(us), 646
[INSERT], Return=OK, 1000
```

For the convenience, I just use the overall runtime and throughput to evaluate the performance. The following workload experiments follow the same rule, too.

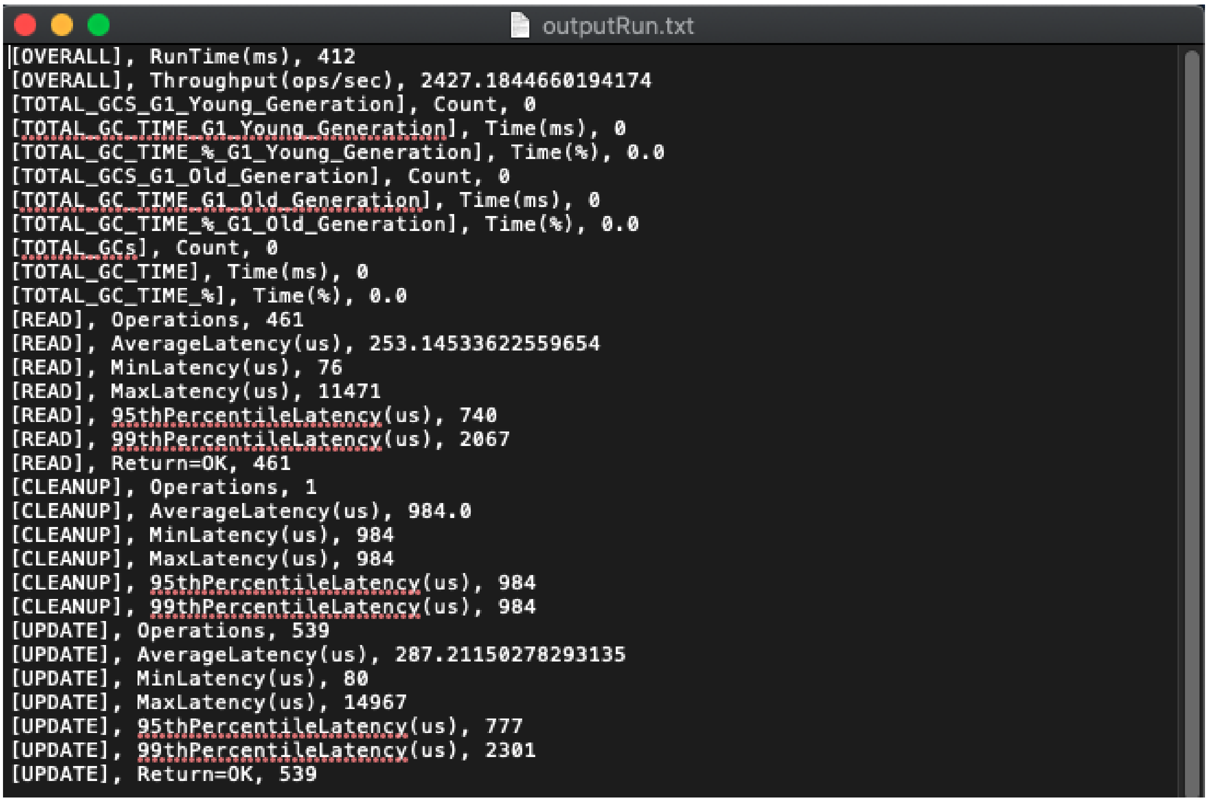
The overall runtime and throughput results for Redis and RocksDB with different data sizes are shown in the following table:

Stores	Size	Runtime(ms)	Throughput(ops/s)
Redis	1000	332	3012.05
	3000	930	3225.80
	6000	1156	5190.31
	10000	1703	5871.99
RocksDB	1000	860	1162.79
	3000	487	6160.16
	6000	634	9463.72
	10000	1131	8841.73

A. Update heavy workload

This workload has a mix of 50/50 reads and writes.

Here is the sample output of results for update heavy workload by Redis with data size 1000.



```
[OVERALL], RunTime(ms), 412
[OVERALL], Throughput(ops/sec), 2427.1844660194174
[TOTAL_GCS_G1_Young_Generation], Count, 0
[TOTAL_GC_TIME_G1_Young_Generation], Time(ms), 0
[TOTAL_GC_TIME_%_G1_Young_Generation], Time(%), 0.0
[TOTAL_GCS_G1_Old_Generation], Count, 0
[TOTAL_GC_TIME_G1_Old_Generation], Time(ms), 0
[TOTAL_GC_TIME_%_G1_Old_Generation], Time(%), 0.0
[TOTAL_GCs], Count, 0
[TOTAL_GC_TIME], Time(ms), 0
[TOTAL_GC_TIME_%], Time(%), 0.0
[READ], Operations, 461
[READ], AverageLatency(us), 253.14533622559654
[READ], MinLatency(us), 76
[READ], MaxLatency(us), 11471
[READ], 95thPercentileLatency(us), 740
[READ], 99thPercentileLatency(us), 2067
[READ], Return=OK, 461
[CLEANUP], Operations, 1
[CLEANUP], AverageLatency(us), 984.0
[CLEANUP], MinLatency(us), 984
[CLEANUP], MaxLatency(us), 984
[CLEANUP], 95thPercentileLatency(us), 984
[CLEANUP], 99thPercentileLatency(us), 984
[UPDATE], Operations, 539
[UPDATE], AverageLatency(us), 287.21150278293135
[UPDATE], MinLatency(us), 80
[UPDATE], MaxLatency(us), 14967
[UPDATE], 95thPercentileLatency(us), 777
[UPDATE], 99thPercentileLatency(us), 2301
[UPDATE], Return=OK, 539
```

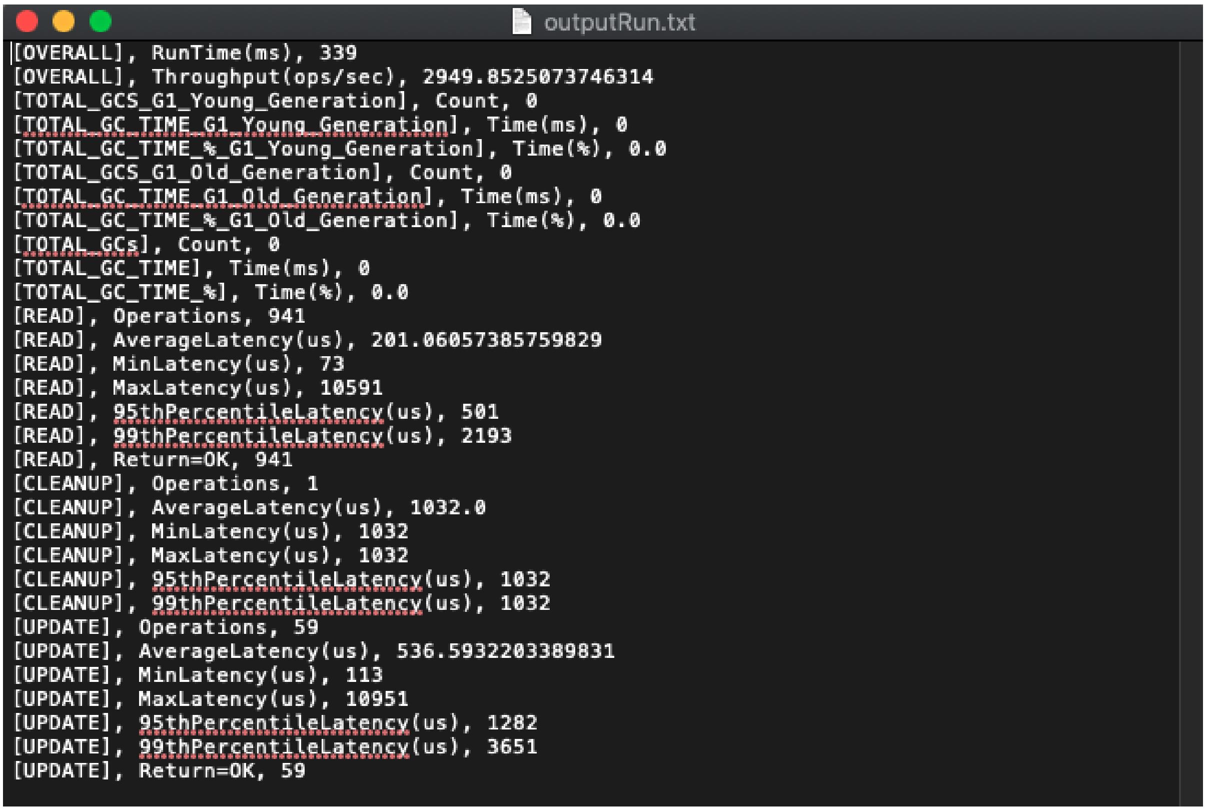
The overall runtime and throughput results for Redis and RocksDB with different data sizes are shown in the following table:

Stores	Size	Runtime(ms)	Throughput(ops/s)
Redis	1000	412	2427.18
	3000	599	5008.34
	6000	941	6376.20
	10000	1151	8688.10
RocksDB	1000	587	1703.58
	3000	515	5825.24
	6000	514	11673.15
	10000	923	10834.24

B. Read mostly workload

This workload has a 95/5 reads/write mix.

Here is the sample output of results for read mostly workload by Redis with data size 1000.



outputRun.txt

```
[OVERALL], RunTime(ms), 339
[OVERALL], Throughput(ops/sec), 2949.8525073746314
[TOTAL_GCS_G1_Young_Generation], Count, 0
[TOTAL_GC_TIME_G1_Young_Generation], Time(ms), 0
[TOTAL_GC_TIME_%_G1_Young_Generation], Time(%), 0.0
[TOTAL_GCS_G1_Old_Generation], Count, 0
[TOTAL_GC_TIME_G1_Old_Generation], Time(ms), 0
[TOTAL_GC_TIME_%_G1_Old_Generation], Time(%), 0.0
[TOTAL_GCs], Count, 0
[TOTAL_GC_TIME], Time(ms), 0
[TOTAL_GC_TIME_%], Time(%), 0.0
[READ], Operations, 941
[READ], AverageLatency(us), 201.06057385759829
[READ], MinLatency(us), 73
[READ], MaxLatency(us), 10591
[READ], 95thPercentileLatency(us), 501
[READ], 99thPercentileLatency(us), 2193
[READ], Return=OK, 941
[CLEANUP], Operations, 1
[CLEANUP], AverageLatency(us), 1032.0
[CLEANUP], MinLatency(us), 1032
[CLEANUP], MaxLatency(us), 1032
[CLEANUP], 95thPercentileLatency(us), 1032
[CLEANUP], 99thPercentileLatency(us), 1032
[UPDATE], Operations, 59
[UPDATE], AverageLatency(us), 536.5932203389831
[UPDATE], MinLatency(us), 113
[UPDATE], MaxLatency(us), 10951
[UPDATE], 95thPercentileLatency(us), 1282
[UPDATE], 99thPercentileLatency(us), 3651
[UPDATE], Return=OK, 59
```

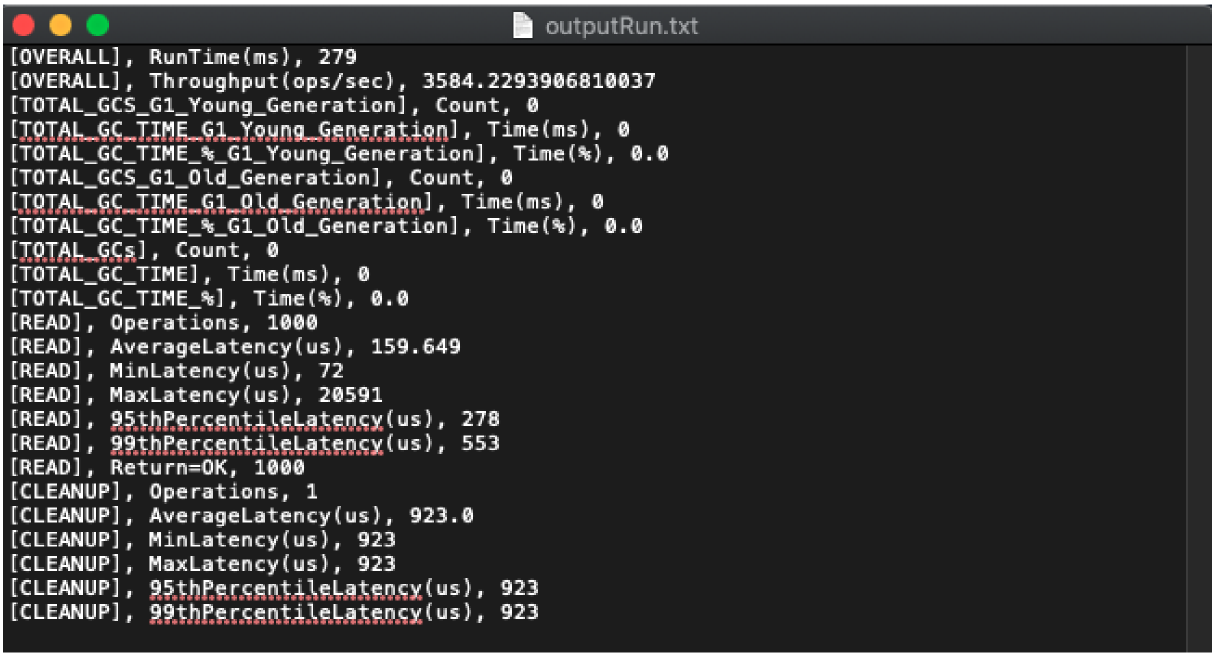
The overall runtime and throughput results for Redis and RocksDB with different data sizes are shown in the following table:

Stores	Size	Runtime(ms)	Throughput(ops/s)
Redis	1000	339	2949.85
	3000	470	6382.98
	6000	1058	5671.08
	10000	906	11037.53
RocksDB	1000	284	3521.12
	3000	367	8174.39
	6000	708	8474.58
	10000	748	13368.98

C. Read only workload

This workload is 100% read.

Here is the sample output of results for read only workload by Redis with data size 1000.



```
[OVERALL], RunTime(ms), 279
[OVERALL], Throughput(ops/sec), 3584.2293906810037
[TOTAL_GCS_G1_Young_Generation], Count, 0
[TOTAL_GC_TIME_G1_Young_Generation], Time(ms), 0
[TOTAL_GC_TIME_%_G1_Young_Generation], Time(%), 0.0
[TOTAL_GCS_G1_Old_Generation], Count, 0
[TOTAL_GC_TIME_G1_Old_Generation], Time(ms), 0
[TOTAL_GC_TIME_%_G1_Old_Generation], Time(%), 0.0
[TOTAL_GCs], Count, 0
[TOTAL_GC_TIME], Time(ms), 0
[TOTAL_GC_TIME_%], Time(%), 0.0
[READ], Operations, 1000
[READ], AverageLatency(us), 159.649
[READ], MinLatency(us), 72
[READ], MaxLatency(us), 20591
[READ], 95thPercentileLatency(us), 278
[READ], 99thPercentileLatency(us), 553
[READ], Return=OK, 1000
[CLEANUP], Operations, 1
[CLEANUP], AverageLatency(us), 923.0
[CLEANUP], MinLatency(us), 923
[CLEANUP], MaxLatency(us), 923
[CLEANUP], 95thPercentileLatency(us), 923
[CLEANUP], 99thPercentileLatency(us), 923
```

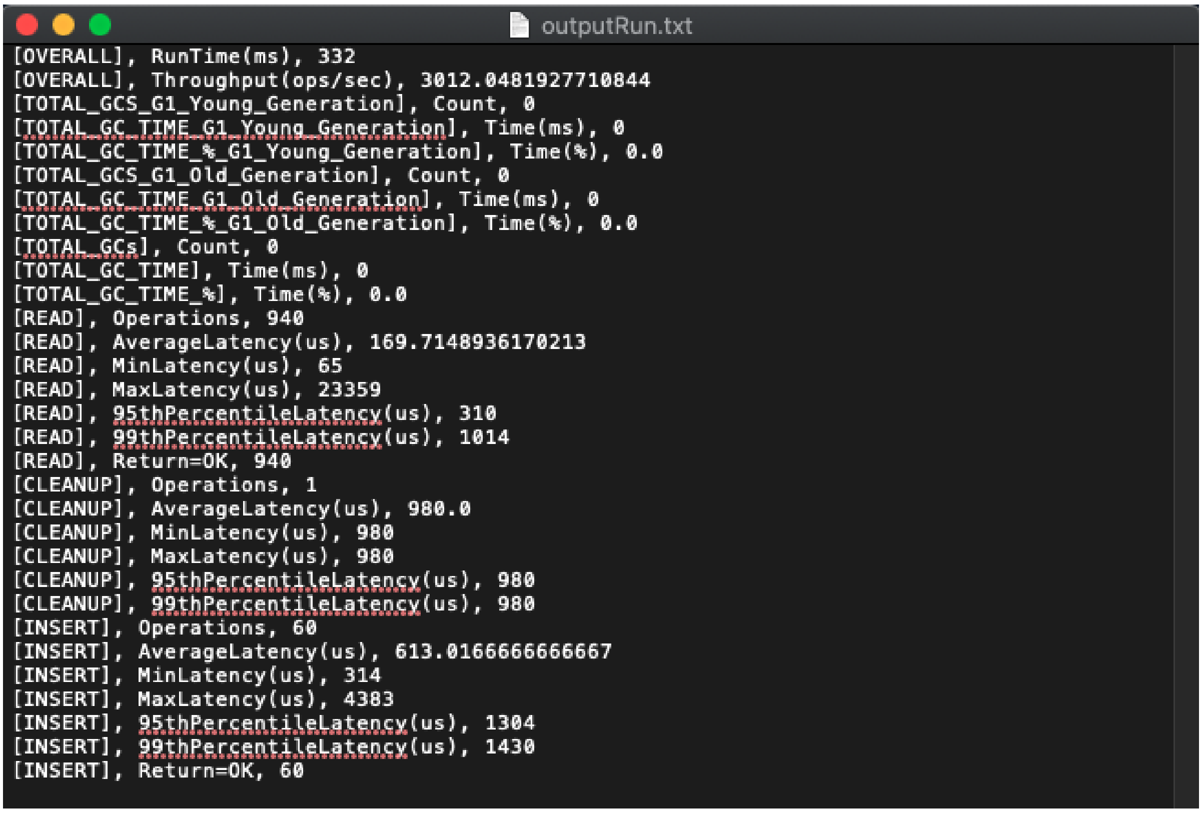
The overall runtime and throughput results for Redis and RocksDB with different data sizes are shown in the following table:

Stores	Size	Runtime(ms)	Throughput(ops/s)
Redis	1000	279	3584.23
	3000	657	4566.21
	6000	746	8042.90
	10000	2048	4882.81
RocksDB	1000	648	1543.21
	3000	397	7556.68
	6000	450	13333.33
	10000	557	17953.32

D. Read latest workload

In this workload, new records are inserted, and the most recently inserted records are the most popular.

Here is the sample output of results for read latest workload by Redis with data size 1000.



```
[OVERALL], RunTime(ms), 332
[OVERALL], Throughput(ops/sec), 3012.0481927710844
[TOTAL_GCS_G1_Young_Generation], Count, 0
[TOTAL_GC_TIME_G1_Young_Generation], Time(ms), 0
[TOTAL_GC_TIME_%_G1_Young_Generation], Time(%), 0.0
[TOTAL_GCS_G1_Old_Generation], Count, 0
[TOTAL_GC_TIME_G1_Old_Generation], Time(ms), 0
[TOTAL_GC_TIME_%_G1_Old_Generation], Time(%), 0.0
[TOTAL_GCs], Count, 0
[TOTAL_GC_TIME], Time(ms), 0
[TOTAL_GC_TIME_%], Time(%), 0.0
[READ], Operations, 940
[READ], AverageLatency(us), 169.7148936170213
[READ], MinLatency(us), 65
[READ], MaxLatency(us), 23359
[READ], 95thPercentileLatency(us), 310
[READ], 99thPercentileLatency(us), 1014
[READ], Return=OK, 940
[CLEANUP], Operations, 1
[CLEANUP], AverageLatency(us), 980.0
[CLEANUP], MinLatency(us), 980
[CLEANUP], MaxLatency(us), 980
[CLEANUP], 95thPercentileLatency(us), 980
[CLEANUP], 99thPercentileLatency(us), 980
[INSERT], Operations, 60
[INSERT], AverageLatency(us), 613.0166666666667
[INSERT], MinLatency(us), 314
[INSERT], MaxLatency(us), 4383
[INSERT], 95thPercentileLatency(us), 1304
[INSERT], 99thPercentileLatency(us), 1430
[INSERT], Return=OK, 60
```

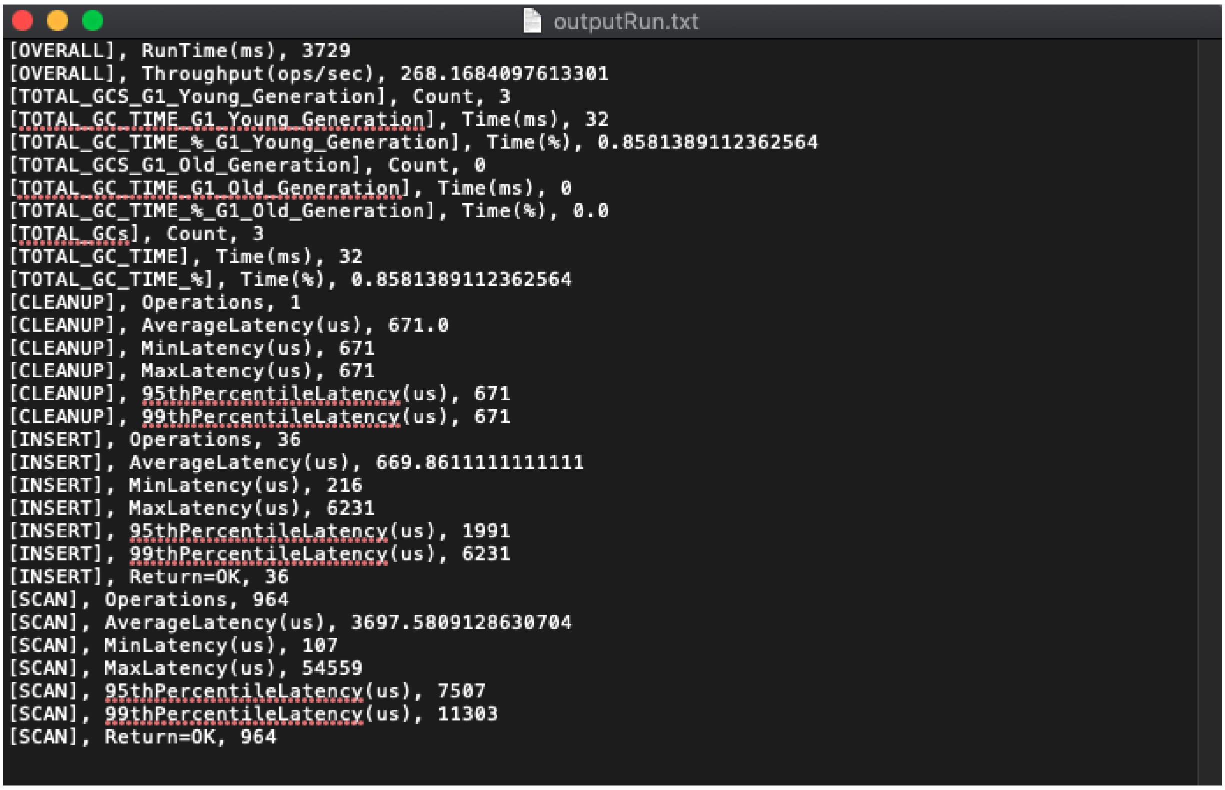
The overall runtime and throughput results for Redis and RocksDB with different data sizes are shown in the following table:

Stores	Size	Runtime(ms)	Throughput(ops/s)
Redis	1000	332	3012.05
	3000	512	5859.38
	6000	880	6818.18
	10000	1049	9532.89
RocksDB	1000	398	2512.56
	3000	412	7281.55
	6000	418	14354.07
	10000	466	21459.23

E. Short ranges

In this workload, short ranges of records are queried, instead of individual records.

Here is the sample output of results for short ranges by Redis with data size 1000.



```
[OVERALL], RunTime(ms), 3729
[OVERALL], Throughput(ops/sec), 268.1684097613301
[TOTAL_GCS_G1_Young_Generation], Count, 3
[TOTAL_GC_TIME_G1_Young_Generation], Time(ms), 32
[TOTAL_GC_TIME_%_G1_Young_Generation], Time(%), 0.8581389112362564
[TOTAL_GCS_G1_Old_Generation], Count, 0
[TOTAL_GC_TIME_G1_Old_Generation], Time(ms), 0
[TOTAL_GC_TIME_%_G1_Old_Generation], Time(%), 0.0
[TOTAL_GCs], Count, 3
[TOTAL_GC_TIME], Time(ms), 32
[TOTAL_GC_TIME_%], Time(%), 0.8581389112362564
[CLEANUP], Operations, 1
[CLEANUP], AverageLatency(us), 671.0
[CLEANUP], MinLatency(us), 671
[CLEANUP], MaxLatency(us), 671
[CLEANUP], 95thPercentileLatency(us), 671
[CLEANUP], 99thPercentileLatency(us), 671
[INSERT], Operations, 36
[INSERT], AverageLatency(us), 669.8611111111111
[INSERT], MinLatency(us), 216
[INSERT], MaxLatency(us), 6231
[INSERT], 95thPercentileLatency(us), 1991
[INSERT], 99thPercentileLatency(us), 6231
[INSERT], Return=OK, 36
[SCAN], Operations, 964
[SCAN], AverageLatency(us), 3697.5809128630704
[SCAN], MinLatency(us), 107
[SCAN], MaxLatency(us), 54559
[SCAN], 95thPercentileLatency(us), 7507
[SCAN], 99thPercentileLatency(us), 11303
[SCAN], Return=OK, 964
```

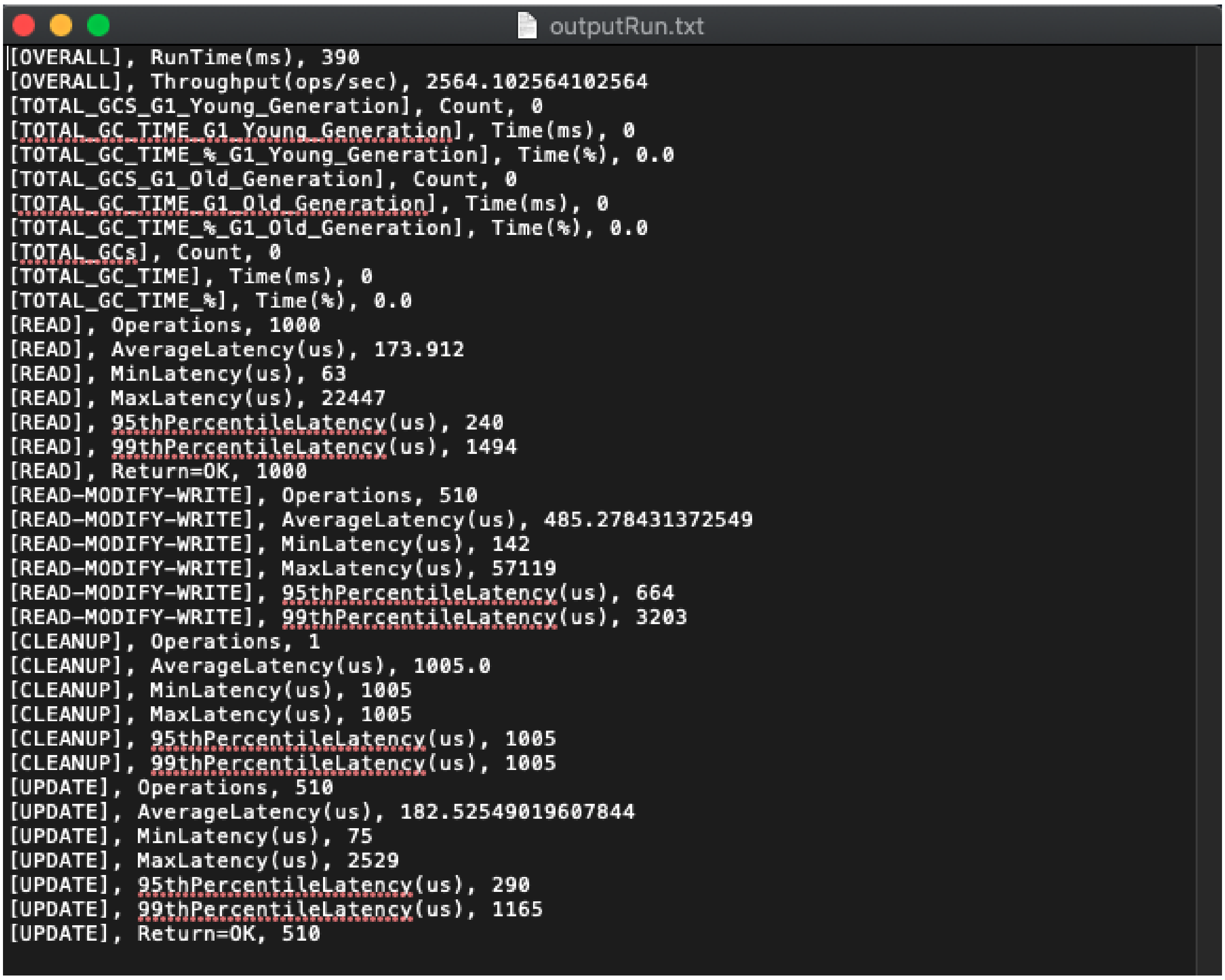
The overall runtime and throughput results for Redis and RocksDB with different data sizes are shown in the following table:

Stores	Size	Runtime(ms)	Throughput(ops/s)
Redis	1000	3729	268.17
	3000	8150	368.10
	6000	13982	429.12
	10000	22955	435.63
RocksDB	1000	652	1533.74
	3000	990	3030.30
	6000	980	6122.45
	10000	1819	5497.53

F. Read-modify-write

In this workload, the client will read a record, modify it, and write back the changes.

Here is the sample output of results for read-modify-write by Redis with data size 1000.



```
[OVERALL], RunTime(ms), 390
[OVERALL], Throughput(ops/sec), 2564.102564102564
[TOTAL_GCS_G1_Young_Generation], Count, 0
[TOTAL_GC_TIME_G1_Young_Generation], Time(ms), 0
[TOTAL_GC_TIME_%_G1_Young_Generation], Time(%), 0.0
[TOTAL_GCS_G1_Old_Generation], Count, 0
[TOTAL_GC_TIME_G1_Old_Generation], Time(ms), 0
[TOTAL_GC_TIME_%_G1_Old_Generation], Time(%), 0.0
[TOTAL_GCs], Count, 0
[TOTAL_GC_TIME], Time(ms), 0
[TOTAL_GC_TIME_%], Time(%), 0.0
[READ], Operations, 1000
[READ], AverageLatency(us), 173.912
[READ], MinLatency(us), 63
[READ], MaxLatency(us), 22447
[READ], 95thPercentileLatency(us), 240
[READ], 99thPercentileLatency(us), 1494
[READ], Return=OK, 1000
[READ-MODIFY-WRITE], Operations, 510
[READ-MODIFY-WRITE], AverageLatency(us), 485.278431372549
[READ-MODIFY-WRITE], MinLatency(us), 142
[READ-MODIFY-WRITE], MaxLatency(us), 57119
[READ-MODIFY-WRITE], 95thPercentileLatency(us), 664
[READ-MODIFY-WRITE], 99thPercentileLatency(us), 3203
[CLEANUP], Operations, 1
[CLEANUP], AverageLatency(us), 1005.0
[CLEANUP], MinLatency(us), 1005
[CLEANUP], MaxLatency(us), 1005
[CLEANUP], 95thPercentileLatency(us), 1005
[CLEANUP], 99thPercentileLatency(us), 1005
[UPDATE], Operations, 510
[UPDATE], AverageLatency(us), 182.52549019607844
[UPDATE], MinLatency(us), 75
[UPDATE], MaxLatency(us), 2529
[UPDATE], 95thPercentileLatency(us), 290
[UPDATE], 99thPercentileLatency(us), 1165
[UPDATE], Return=OK, 510
```

The overall runtime and throughput results for Redis and RocksDB with different data sizes are shown in the following table:

Stores	Size	Runtime(ms)	Throughput(ops/s)
Redis	1000	390	2564.10
	3000	708	4237.29
	6000	902	6651.88
	10000	1349	7412.90
RocksDB	1000	676	1479.29
	3000	611	4909.98
	6000	698	8595.99
	10000	626	15974.44

Analysis and Conclusion

- For the bulk loading of data test, RocksDB has larger throughput than Redis when the size of dataset increases. However, RocksDB performs poorly when the size of dataset is 1000. Also, its throughput reaches the peek at 9463 when the size of dataset is 6000. It looks like that the RocksDB reaches its bottleneck when the size of dataset is over 6000 while Redis can still increase the throughput even if the size of dataset is over 10000.
- For the update heavy workload test, when the size of dataset is small, like 3000, Redis and RocksDB have the similar throughput. However, when the dataset grows larger, RocksDB performs better than Redis. However, it seems that Redis doesn't reach its bottleneck while RocksDB reaches its bottleneck.
- For the read mostly workload test, both RocksDB and Redis don't reach the bottleneck when there are 10000 records in the data. However, RocksDB performs the read operations more quickly than the Redis.
- For the read only workload test, RocksDB performs much better than Redis when the size of dataset increases. When there are 10000 records, RocksDB has 18000 ops/s throughput while Redis only has 4800 ops/s throughput. It seems that RocksDB is good at read operations.
- For the read latest workload test, RocksDB has much larger throughput than Redis again. Both RocksDB and Redis don't reach to their bottlenecks.
- For the short ranges test, RocksDB spends much less time and performs much larger throughput than Redis from small size of data to large size of data. It indicates that RocksDB has better query operations support than Redis.
- For the read-modify-write test, RocksDB and Redis have similar throughput when the size of dataset is from 1000 to 6000. When the size of dataset increases to 10000, RocksDB performs twice throughput than Redis.

From the above observations, we can conclude that RocksDB has better scalability than Redis. For the large dataset, RocksDB increase its throughput more quikly than Redis and performs better. Also, RocksDB supports the query operations with good performance while Redis performs badly for the query operations.

- Compare all the seven workload tests, we can find that with the same size of dataset, for Redis, the increasing sequence of time cost for these workload tests is: **read only workload < read lastest workload < read mostly workload < read-modify-write < update heavy workload < short ranges.** For RocksDB, the increasing sequence of time cost for these workload tests is: **read only workload < read lastest workload < read mostly workload < read-modify-write < update heavy workload < short ranges.**

From the above observations, we can conclude that in both Redis and RocksDB, the time cost of operation has the relationship that: **READ < UPDATE < INSERT < QUERY.**