CS6650 Assignment-1 Tan Wang

Git Repository: https://github.com/TanWang-0914/NEU6650-HW

Client Side Design

Major Classes:

PurchaseApiTest - Main class of client side with main function. This class will do following jobs:

1. Create Api instance and set base path for sending request;
2. Prompt for user input and validate user input against default value;
3. Create blocking queue and data consumer class for storing response records and start data consuming thread.(part2)
4. Create a phase class to keep tracking current phase running.
5. Create store threads based on current phase. If current phase is “EastPhaseStart”, main class will create numStore/4 store thread and change phase to “EastPhaseRunning”; If current phase is “CentralPhaseStart”, main class will create another numStore/4 store thread and change phase to “CentralPhaseRunning”;If current phase is “WestPhaseStart”, main class will create rest numStore/2 store thread and change phase to “WestPhaseRunning”;
6. Track wall time and calculate number of successful requests and throughput.
7. Read from csv file and calculate mean/median/p99/maximum response latency.(part2)

(note:jobs with part2 comment are only executed in part2, code has been comment out in part1)

RequestPerStore - this is the thread class for each store, it contains each store’s information and thread run method

1. For each operating hour, this thread will create numPurchases’s singleRequest thread and start singleRequest thread.
2. Each request are send sequentially, store thread will make sure each singleRequest has finished to start sending next request.
3. If one store thread has send 3*numPurchases and current running phase is “EastPhaseRunning”, store thread will change current phase to “CentralPhaseStart".
4. If one store thread has send 5*numPurchases and current running phase is “CentralPhaseRunning”, store thread will change current phase to “WestPhaseStart".

SingleRequest - this is the thread class for a single request sent to server, each request has its parameters(storeID, custID, date).

1. Each thread will create numbers of purchases item and add them to purchase body, then send request to server, get response, track respond time and add response code, start time, latency to blocking queue.

Phase - This is a class with current running phase of client side, both main thread and store thread had access to this class.

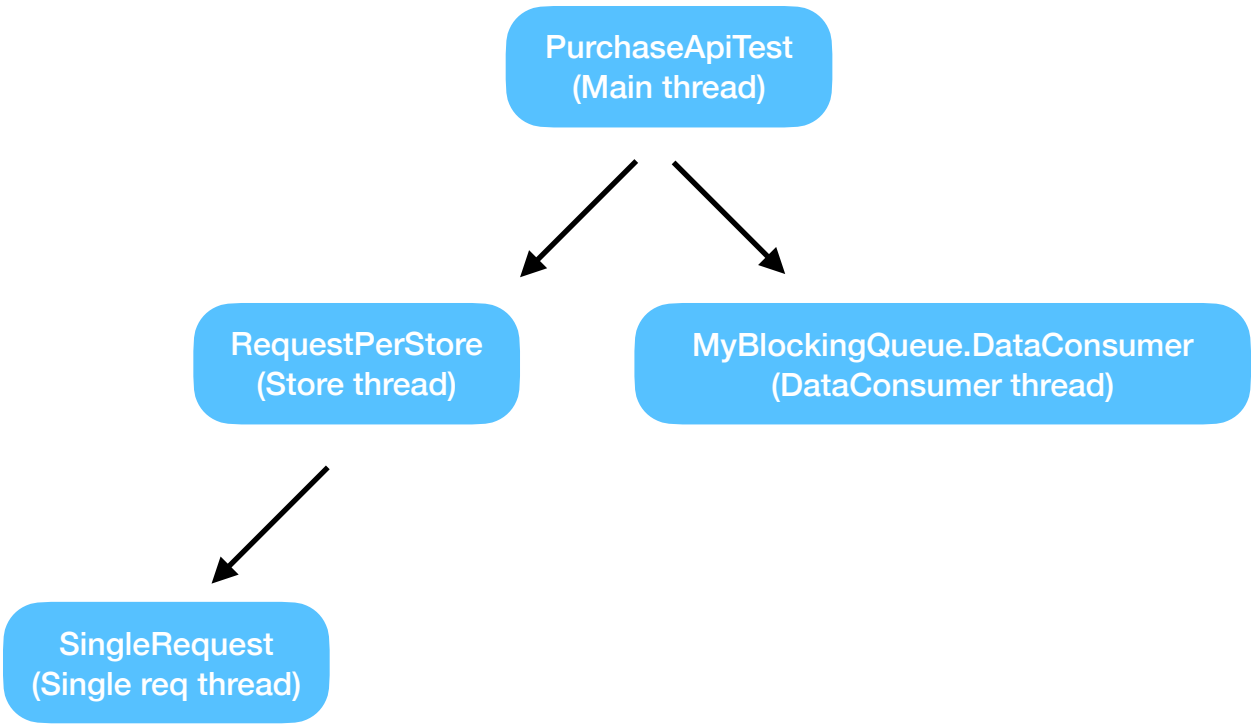
1. This class has two sync method changePhase() and getPhase(), sync methods are used to make sure only one thread can modify and see current phase, it will prevent data racing.
2. There are 6 possible phases in this assignment, which are {“EastPhaseStart”, “EastPhaseRunning”, “CentralPhaseStart”, “CentralPhaseRunning”, “WestPhaseStart”, “WestPhaseRunning”}

MyBlockingQueue - this class create a blocking queue to store response records, it also create dataConsumer for this blocking queue. (This class is added for part2, class code also exist in part1 but are not called in main class)

DataConsumer - this class is designed to consume data in MyBlockingQueue, it will create a csvFile and csvWriter, and write all response to csvFile. Writer will write 10000 records at a time to reduce numbers of write to disk. (This class is added for part2, class code also exist in part1 but are not called in main class)

ReqCount - this class will keep counting of failed request, it has a sync method incFail(), since all store threads are sending request at the same time, sync method will prevent data racing.

Thread structure:



Client Part 1 Running Result

32 thread
Total Successful request: 17280
Wall Time (sec): 17.21
Throughput: 1003

```
[ec2-user@ip-172-31-89-76 WorkSpace]$ java -jar swagger-java-client-1.0.0-jar-with-dependencies.jar
http://35.175.64.87:8080/HW_1_war/
Enter maximum number of stores to simulate (maxStores):
32
maxStores = 32
Enter number of customers/store (default 1000):
1000
maxCustID = 1000
Enter maximum itemID - default 100000:
100000
maxItemID = 100000
Enter number of purchases per hour: (default 60):
60
numPurchases = 60
Enter number of items for each purchase (range 1-20, default 5):
5
numItemPerPurchase = 5
Enter date - default to 20210101:
20210101
date = 20210101
Enter ipAddress :
0.0.0.0
EastPhaseStart
EastPhaseRunning
CentralPhaseStart
CentralPhaseRunning
WestPhaseStart
WestPhaseRunning
Number of Stores/threads:32
All threads finished
Total successful Request:17280
Total failed Request:0
Time Period:17.214
Throughput:1003.8340885325898
Program finished.
[ec2-user@ip-172-31-89-76 WorkSpace]$
```

64 thread
Total Successful request: 34560
Wall Time (sec):27.44
Throughput: 1259

```
[ec2-user@ip-172-31-89-76 WorkSpace]$ java -jar swagger-java-client-1.0.0-jar-with-dependencies.jar
http://35.175.64.87:8080/HW_1_war/
Enter maximum number of stores to simulate (maxStores):
64
maxStores = 64
Enter number of customers/store (default 1000):
1000
maxCustID = 1000
Enter maximum itemID - default 100000:
100000
maxItemID = 100000
Enter number of purchases per hour: (default 60):
60
numPurchases = 60
Enter number of items for each purchase (range 1-20, default 5):
5
numItemPerPurchase = 5
Enter date - default to 20210101:
20210101
date = 20210101
Enter ipAddress :
0.0.0.0
EastPhaseStart
EastPhaseRunning
CentralPhaseStart
CentralPhaseRunning
WestPhaseStart
WestPhaseRunning
Number of Stores/threads:64
All threads finished
Total successful Request:34560
Total failed Request:0
Time Period:27.448
Throughput:1259.1081317400174
Program finished.
[ec2-user@ip-172-31-89-76 WorkSpace]$
```

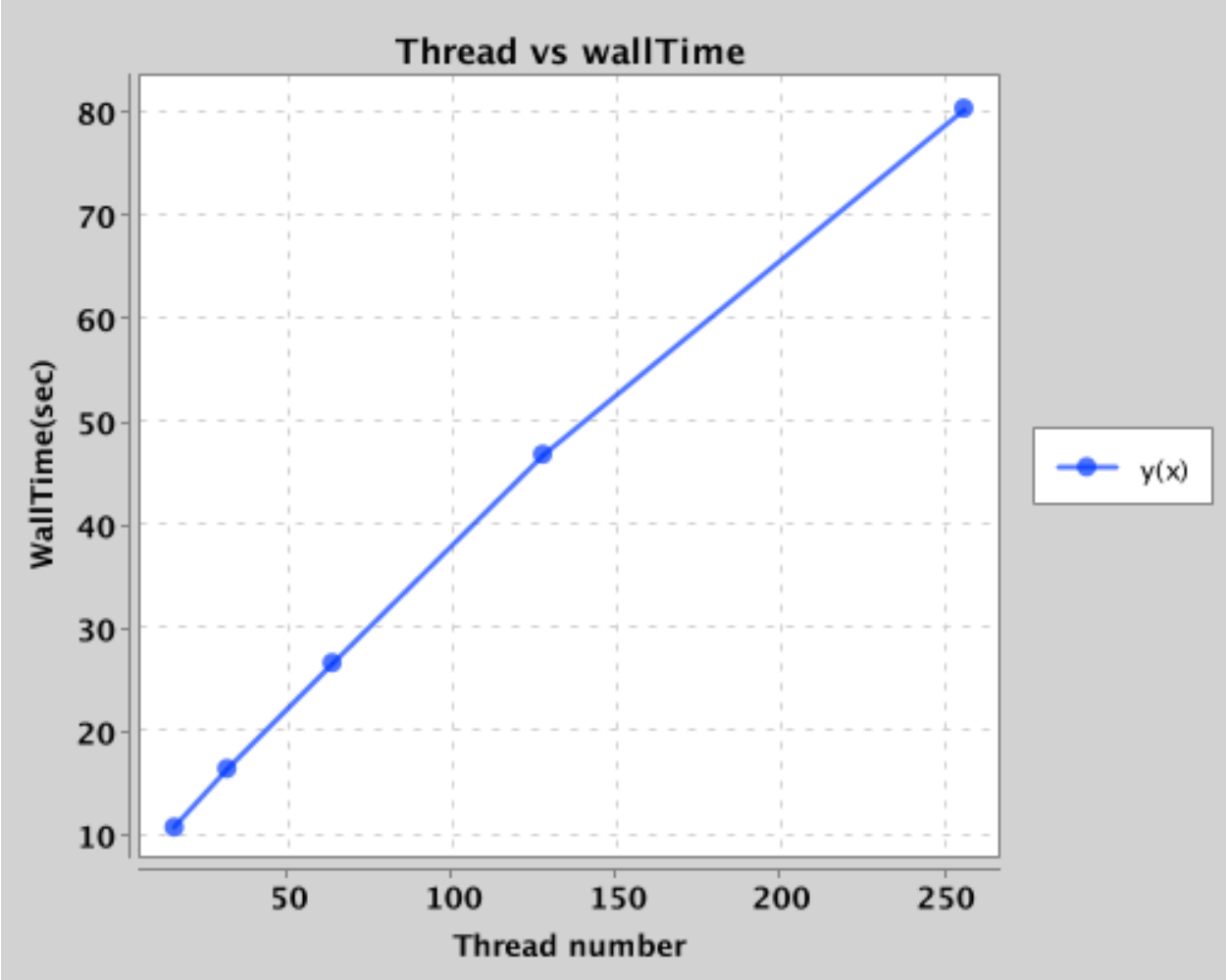
128 thread
Total Successful request: 69120
Wall Time (sec): 46.96
Throughput: 1471

```
[ec2-user@ip-172-31-89-76 WorkSpace]$ java -jar swagger-java-client-1.0.0-jar-with-dependencies.jar
http://52.91.146.150:8080/HW_1_war/
Enter maximum number of stores to simulate (maxStores):
128
maxStores = 128
Enter number of customers/store (default 1000):
1000
maxCustID = 1000
Enter maximum itemID - default 100000:
100000
maxItemID = 100000
Enter number of purchases per hour: (default 60):
60
numPurchases = 60
Enter number of items for each purchase (range 1-20, default 5):
5
numItemPerPurchase = 5
Enter date - default to 20210101:
20210101
date = 20210101
Enter ipAddress :
0.0.0.0
EastPhaseStart
EastPhaseRunning
CentralPhaseStart
CentralPhaseRunning
WestPhaseStart
WestPhaseRunning
Number of Stores/threads:128
All threads finished
Total successful Request:69120
Total failed Request:0
Time Period:46.96
Throughput:1471.8909710391822
Program finished.
[ec2-user@ip-172-31-89-76 WorkSpace]$
```

256 thread
Total Successful request: 138240
Wall Time (sec): 80.46
Throughput: 1717

```
[ec2-user@ip-172-31-89-76 WorkSpace]$ java -jar swagger-java-client-1.0.0-jar-with-dependencies.jar
http://52.91.146.150:8080/HW_1_war/
Enter maximum number of stores to simulate (maxStores):
256
maxStores = 256
Enter number of customers/store (default 1000):
1000
maxCustID = 1000
Enter maximum itemID - default 100000:
100000
maxItemID = 100000
Enter number of purchases per hour: (default 60):
60
numPurchases = 60
Enter number of items for each purchase (range 1-20, default 5):
5
numItemPerPurchase = 5
Enter date - default to 20210101:
20210101
date = 20210101
Enter ipAddress :
0.0.0.0
EastPhaseStart
EastPhaseRunning
CentralPhaseStart
CentralPhaseRunning
WestPhaseStart
WestPhaseRunning
Number of Stores/threads:256
All threads finished
Total successful Request:138240
Total failed Request:0
Time Period:80.469
Throughput:1717.928643328487
Program finished.
[ec2-user@ip-172-31-89-76 WorkSpace]$
```

Client Part1 Thread-WallTime Plot



From this graph we can see that there is a positive correlation between thread number and wall time. With more thread running, the wall time of program will increase too. Since we only have one server, our throughput are reaching an upper bound and will increase slower and slower, wall time will increase at a constant speed gradually.

Client Part 2 Running Result

(mean/median/p99/max/throughput are captured in output window)

32 thread

Total Successful request: 17280
Wall Time (sec): 17.63
Throughput: 979

```
[ec2-user@ip-172-31-89-76 WorkSpace]$ java -jar swagger-java-client-part2-1.0.0-jar-with-dependencies.jar
http://52.91.146.150:8080/HW_1_war/
Enter maximum number of stores to simulate (maxStores):
32
maxStores = 32
Enter number of customers/store (default 1000):
1000
maxCustID = 1000
Enter maximum itemID - default 100000:
100000
maxItemID = 100000
Enter number of purchases per hour: (default 60):
60
numPurchases = 60
Enter number of items for each purchase (range 1-20, default 5):
5
numItemPerPurchase = 5
Enter date - default to 20210101:
20210101
date = 20210101
Enter ipAddress :
0.0.0.0
EastPhaseStart
EastPhaseRunning
CentralPhaseStart
CentralPhaseRunning
WestPhaseStart
WestPhaseRunning
Number of Stores/threads:32
All threads finished
Total successful Request:17280
Total failed Request:0
Time Period:17.637
Throughput:979.7584623235243
Mean Response Time:3
Median Response Time:2
P99 Response Time:19
Max Response Time:844
consumer thread finished.
Program finished.
[ec2-user@ip-172-31-89-76 WorkSpace]$
```

64 thread

Total Successful request: 34560
Wall Time (sec): 28.31
Throughput: 1220

```
[ec2-user@ip-172-31-89-76 WorkSpace]$ java -jar swagger-java-client-part2-1.0.0-jar-with-dependencies.jar
http://52.91.146.150:8080/HW_1_war/
Enter maximum number of stores to simulate (maxStores):
64
maxStores = 64
Enter number of customers/store (default 1000):
1000
maxCustID = 1000
Enter maximum itemID - default 100000:
100000
maxItemID = 100000
Enter number of purchases per hour: (default 60):
60
numPurchases = 60
Enter number of items for each purchase (range 1-20, default 5):
5
numItemPerPurchase = 5
Enter date - default to 20210101:
20210101
date = 20210101
Enter ipAddress :
0.0.0.0
EastPhaseStart
EastPhaseRunning
CentralPhaseStart
CentralPhaseRunning
WestPhaseStart
WestPhaseRunning
Number of Stores/threads:64
All threads finished
Total successful Request:34560
Total failed Request:0
Time Period:28.316
Throughput:1220.5113716626643
Mean Response Time:4
Median Response Time:3
P99 Response Time:19
Max Response Time:866
consumer thread finished.
Program finished.
[ec2-user@ip-172-31-89-76 WorkSpace]$
```

128 thread

Total Successful request: 69120
Wall Time (sec): 48.14
Throughput: 1435

```
[ec2-user@ip-172-31-89-76 WorkSpace]$ java -jar swagger-java-client-part2-1.0.0-jar-with-dependencies.jar
http://52.91.146.150:8080/HW_1_war/
Enter maximum number of stores to simulate (maxStores):
128
maxStores = 128
Enter number of customers/store (default 1000):
1000
maxCustID = 1000
Enter maximum itemID - default 100000:
100000
maxItemID = 100000
Enter number of purchases per hour: (default 60):
60
numPurchases = 60
Enter number of items for each purchase (range 1-20, default 5):
5
numItemPerPurchase = 5
Enter date - default to 20210101:
20210101
date = 20210101
Enter ipAddress :
0.0.0.0
EastPhaseStart
EastPhaseRunning
CentralPhaseStart
CentralPhaseRunning
WestPhaseStart
WestPhaseRunning
Number of Stores/threads:128
All threads finished
Total successful Request:69120
Total failed Request:0
Time Period:48.149
Throughput:1435.5438326860371
Mean Response Time:7
Median Response Time:3
P99 Response Time:39
Max Response Time:859
consumer thread finished.
Program finished.
[ec2-user@ip-172-31-89-76 WorkSpace]$
```

256 thread

Total Successful request: 138240
Wall Time (sec): 81.10
Throughput: 1704

```
[ec2-user@ip-172-31-89-76 WorkSpace]$ java -jar swagger-java-client-part2-1.0.0-jar-with-dependencies.jar
http://52.91.146.150:8080/HW_1_war/
Enter maximum number of stores to simulate (maxStores):
256
maxStores = 256
Enter number of customers/store (default 1000):
1000
maxCustID = 1000
Enter maximum itemID - default 100000:
100000
maxItemID = 100000
Enter number of purchases per hour: (default 60):
60
numPurchases = 60
Enter number of items for each purchase (range 1-20, default 5):
5
numItemPerPurchase = 5
Enter date - default to 20210101:
20210101
date = 20210101
Enter ipAddress :
0.0.0.0
EastPhaseStart
EastPhaseRunning
CentralPhaseStart
CentralPhaseRunning
WestPhaseStart
WestPhaseRunning
Number of Stores/threads:256
All threads finished
Total successful Request:138240
Total failed Request:0
Time Period:81.103
Throughput:1704.4992170449923
Mean Response Time:10
Median Response Time:5
P99 Response Time:91
Max Response Time:964
consumer thread finished.
Program finished.
[ec2-user@ip-172-31-89-76 WorkSpace]$
```

The Main difference between part 1 and part 2 is that part 2 need to write all response records to a local file, to reduce the increment of runtime for writing, I used a blocking queue to store records while consumer thread keep taking record from queue.

And also to reduce the times of writing to disk, I will use a list to store records ready to be write and then write 10000 records at a time.

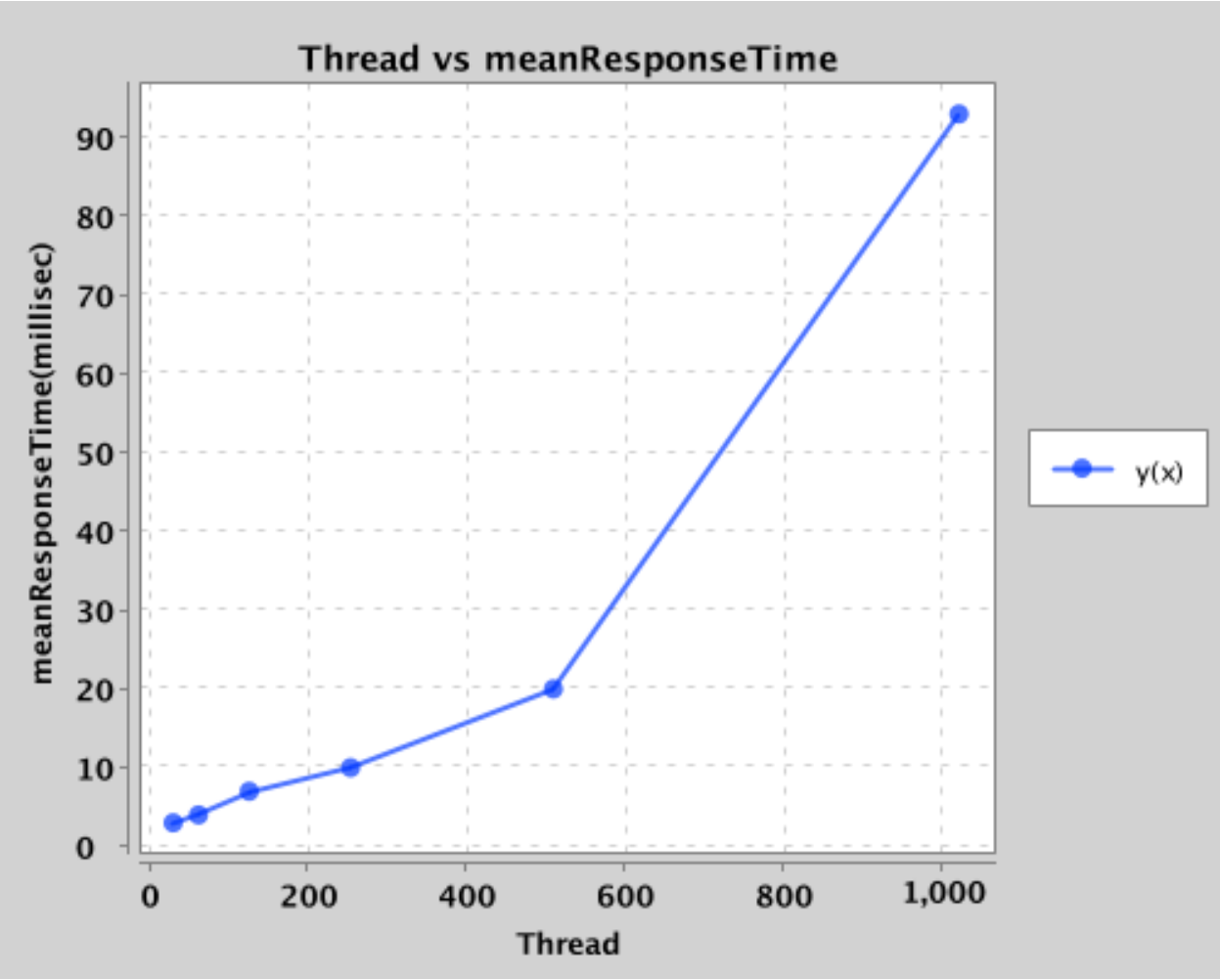
The difference of runtime is not much, comparing to the wall time, when we have less thread, the difference in percentage will be relatively larger. When we have more thread, the difference in wall time will be insignificant.

In cases with 32 threads, the difference are quite close to 5%, in the case of more threads, the difference are within 5%.

The wall time difference between part 1 and part are as follow:

32 thread
Part1-17.21s part2 - 17.63s actual diff 0.42s percentage 5.2%
64 thread
Part1-27.44s part2 - 28.31s actual diff 0.87s percentage 3.1%
128 thread
Part1-46.96s part2 - 48.14s actual diff 1.18s percentage 2.5%
256 thread
Part1-80.46s part2 - 81.10s actual diff 0.64s percentage 0.8%

Client Part2 Throughtput-meanResponseTime Plot



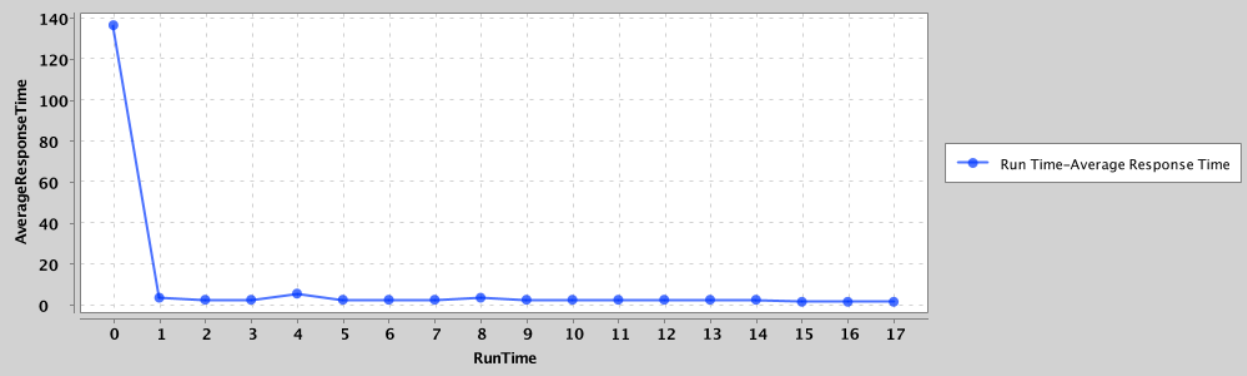
From this graph, throughput is the number of requests processed in a time, in this case, it is number-of-request/second. If we have more requests sent to server, server will work in a heavier load and each request might need a longer time to get response. But as throughput will increase less and less and reach upper bound, mean response time will increase faster than thread number due to the long wait at server.

Bonus Charting- plot average latencies over the whole duration of a test run.

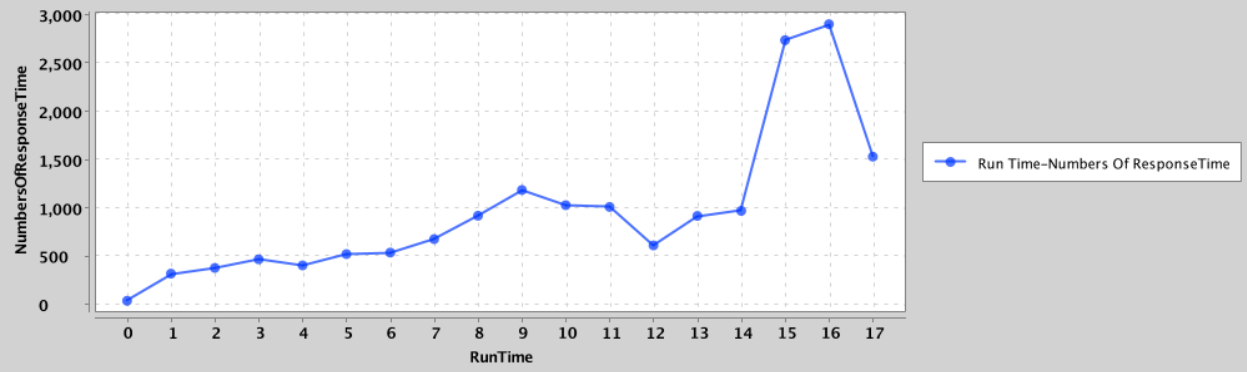
Based on response records, we knew the sent-time of each record, compare the sent time with the start run time, we got the second after program start. Put each Response in buckets and calculate the Runtime vs Average response time:

32 thread

Runtime vs Average Response Latency

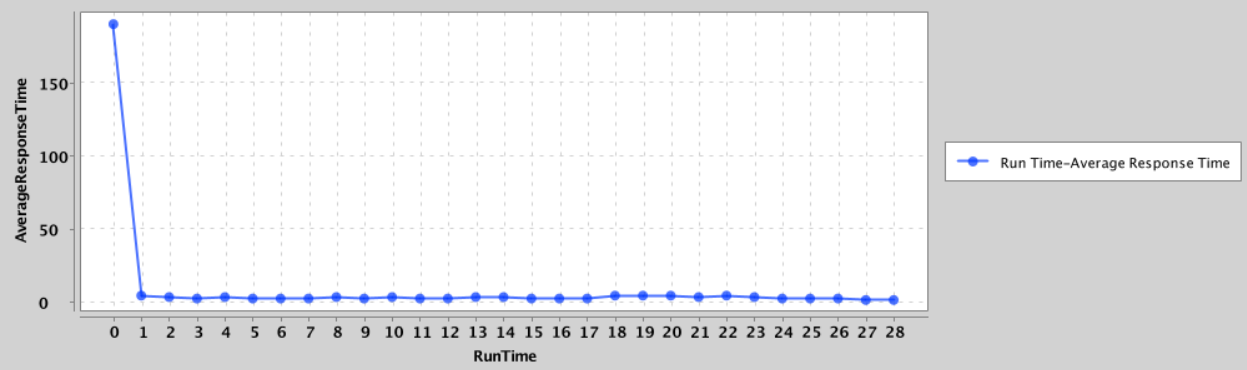


RunTime vs Number of Response

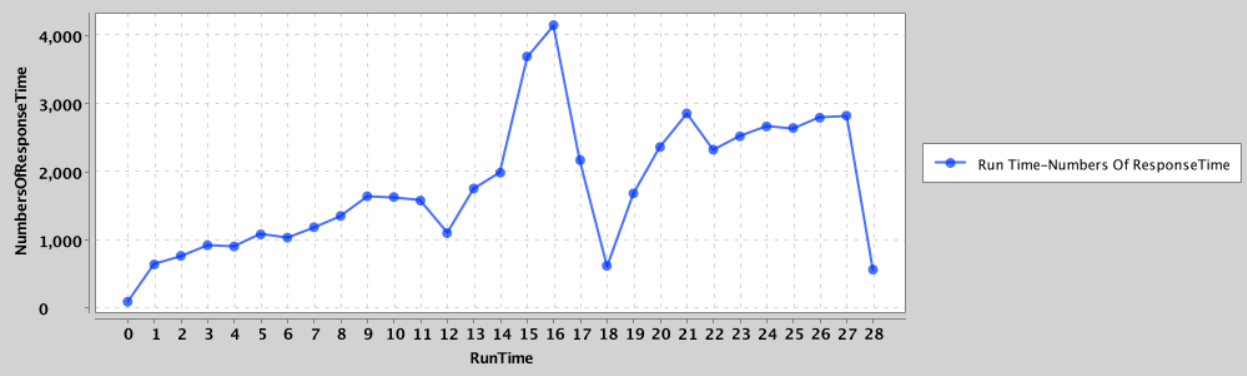


64 thread

Runtime vs Average Response Latency

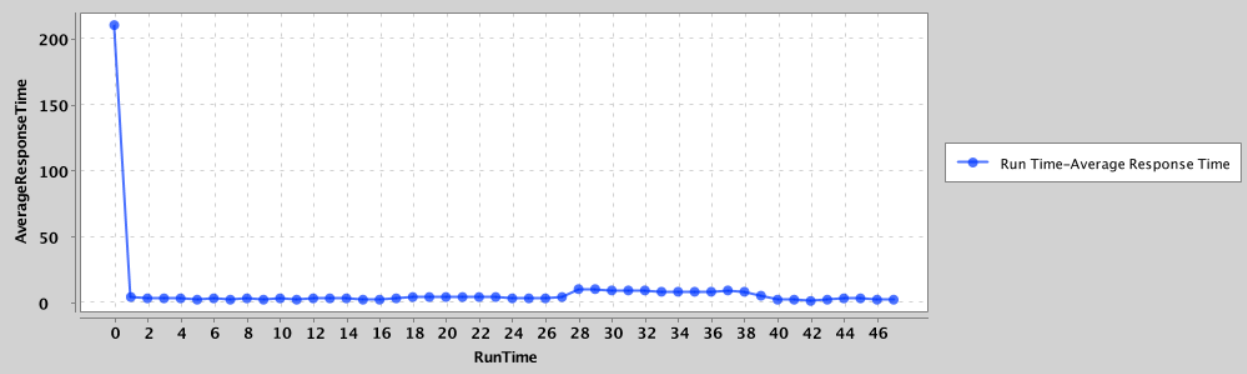


RunTime vs Number of Response

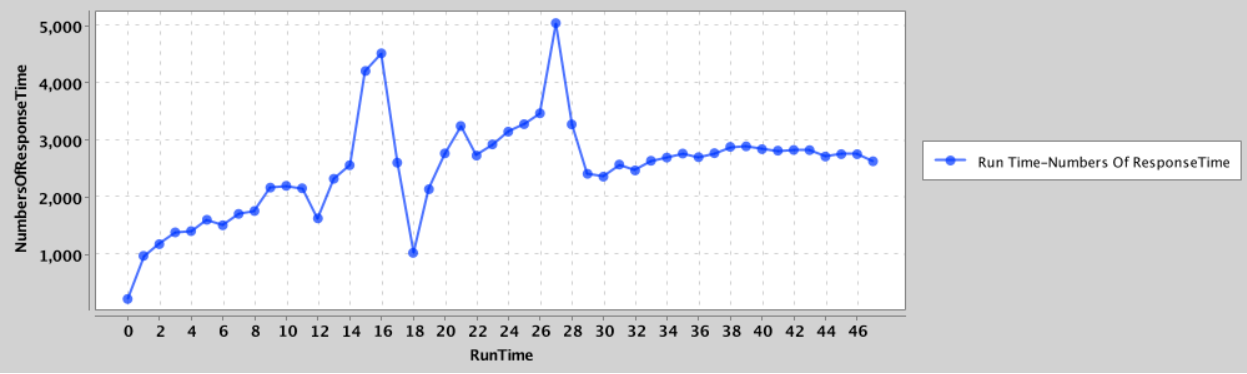


128 thread

Runtime vs Average Response Latency

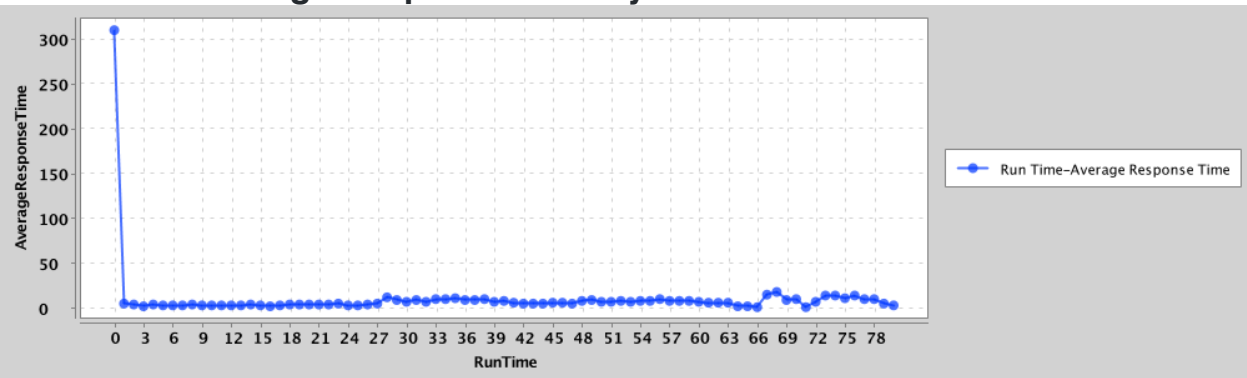


RunTime vs Number of Response

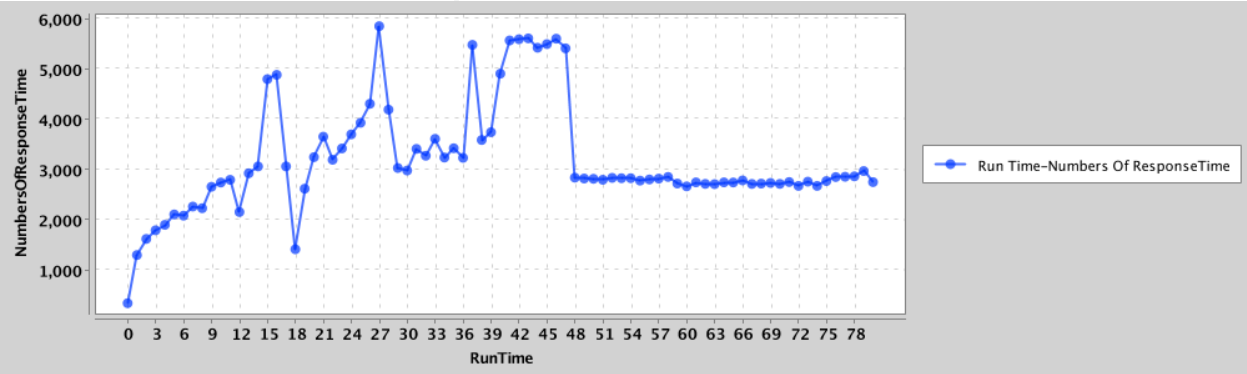


256 thread

Runtime vs Average Response Latency



RunTime vs Number of Response



From these charting, we can see that the first request to server took longer time to receive response than sequential requests. From my point of view, the first request to the server need to create connection between client and server and also ISP router need to find the correct route to sent request to the server's IP address. These works might cause longer time. After the first connection, the rest request should get a response much faster, as all p99 response time are within 50 millisecond.