CS6650 Assignment1 Yue Sun

GitHub Repo: https://github.ccs.neu.edu/yuesun/cs6650

Client Design:

- For the Part1, the major class is **StoreClientP1**.

  The ***main()*** method demonstrates how multiple stores(multi-thread) generate and send synthetic purchase requests to a server in the cloud.

  A **StoreClientP1** is a runnable object representing a single store, and the ***run()*** method shows its behavior from open to close -- sends purchases 60 times per hour for 9 hours.

  There are server static class fields: 3 **CountDownLatch waitForCentral(1)**, **waitForWest(1)**, and **waitForComplete(maxThread)** as the phase barriers, and two integer variables **totalSuccess** and **totalFailure** that count the total successful and unsuccessful requests generated by all stores.

  The ***run()*** will call the ***sendPurchaseOrder()*** method for 60 * 9 times in a for loop, while after sending the 180th request and the 300th request, count down the **waitForCentral** and **waitForWest** respectively. After sending all requests, count down **waitForComplete** and call the synchronized methods ***reportSuccessCount()*** and ***reportFailureCount()*** to add the counts of a single store in a day to the **totalSuccess** and **totalFailure** respectively. As only one thread is able to access the synchronize method, the client is designed to call these two synchronized methods once per store (maxStore times in total) rather than once per purchase request(maxStore * 540 times in total).

- For the Part2, there are two other classes other than **StoreClientP2**.

  OutputReporter is designed to calculate the statistic, show the result in the console, and write the output csv file.

  RequestRecord represents the record for a single purchase request.

  In **StoreClientP2** class, the ***sendPurchaseOrder()*** method will construct a new **RequestRecord** object to store the start time, type, latency, and response status code for the request. And then send the record object to the class static field **BlockingQueue<RequestRecord> allRecord.** The BlockingQueue is used to ensure the thread-safe when receiving and containing all purchase request records for future statistical analysis.

Output Screen Shots:

- **Part1**

32 threads:

```
[ec2-user@ip-172-31-93-254 cs6650]$ java -jar ClientPart1.jar
---- Open stores in East ----
---- Waiting 3 hours for Central ----
---- Open stores in Central ----
---- Waiting 2 hours for West ----
---- Open stores in West ----
---- Waiting for all stores closed ----
---- All stores closed ----
The max thread: 32
The total number of successful requests sent: 17280
The total number of unsuccessful requests sent: 0
The wall time (in second): 7.423
The throughput (requests per second): 2327.899770982083
```

64 threads:

```
[ec2-user@ip-172-31-93-254 cs6650]$ java -jar ClientPart1.jar
---- Open stores in East ----
---- Waiting 3 hours for Central ----
---- Open stores in Central ----
---- Waiting 2 hours for West ----
---- Open stores in West ----
---- Waiting for all stores closed ----
---- All stores closed ----
The max thread: 64
The total number of successful requests sent: 34560
The total number of unsuccessful requests sent: 0
The wall time (in second): 11.985
The throughput (requests per second): 2883.6045056320404
```

128 threads:

```
[ec2-user@ip-172-31-93-254 cs6650]$ java -jar ClientPart1.jar
---- Open stores in East ----
---- Waiting 3 hours for Central ----
---- Open stores in Central ----
---- Waiting 2 hours for West ----
---- Open stores in West ----
---- Waiting for all stores closed ----
---- All stores closed ----
The max thread: 128
The total number of successful requests sent: 69120
The total number of unsuccessful requests sent: 0
The wall time (in second): 20.836
The throughput (requests per second): 3317.3353810712233
```

256 threads:

```
[ec2-user@ip-172-31-93-254 cs6650]$ java -jar ClientPart1.jar
---- Open stores in East ----
---- Waiting 3 hours for Central ----
---- Open stores in Central ----
---- Waiting 2 hours for West ----
---- Open stores in West ----
---- Waiting for all stores closed ----
---- All stores closed ----
The max thread: 256
The total number of successful requests sent: 138240
The total number of unsuccessful requests sent: 0
The wall time (in second): 38.748
The throughput (requests per second): 3567.6680086714155
```

- **Part2**

32 threads:

```
[ec2-user@ip-172-31-93-254 cs6650]$ java -jar ClientPart2.jar
---- Open stores in East ----
---- Waiting 3 hours for Central ----
---- Open stores in Central ----
---- Waiting 2 hours for West ----
---- Open stores in West ----
---- Waiting for all stores closed ----
---- All stores closed ----
The max thread: 32
The total number of successful requests sent: 17280
The total number of unsuccessful requests sent: 0
The wall time (in second): 7.695
The throughput (requests per second): 2245.614035087719
The mean response time: 9.345601851851852
The median response time: 8
The p99(99th percentile) response time: 25
The max response time: 518
17280 records written to csv file.
```

64 threads:

```
[ec2-user@ip-172-31-93-254 cs6650]$ java -jar ClientPart2.jar
---- Open stores in East ----
---- Waiting 3 hours for Central ----
---- Open stores in Central ----
---- Waiting 2 hours for West ----
---- Open stores in West ----
---- Waiting for all stores closed ----
---- All stores closed ----
The max thread: 64
The total number of successful requests sent: 34560
The total number of unsuccessful requests sent: 0
The wall time (in second): 12.128
The throughput (requests per second): 2849.6042216358837
The mean response time: 14.73810763888889
The median response time: 13
The p99(99th percentile) response time: 56
The max response time: 550
34560 records written to csv file.
```

128 threads:

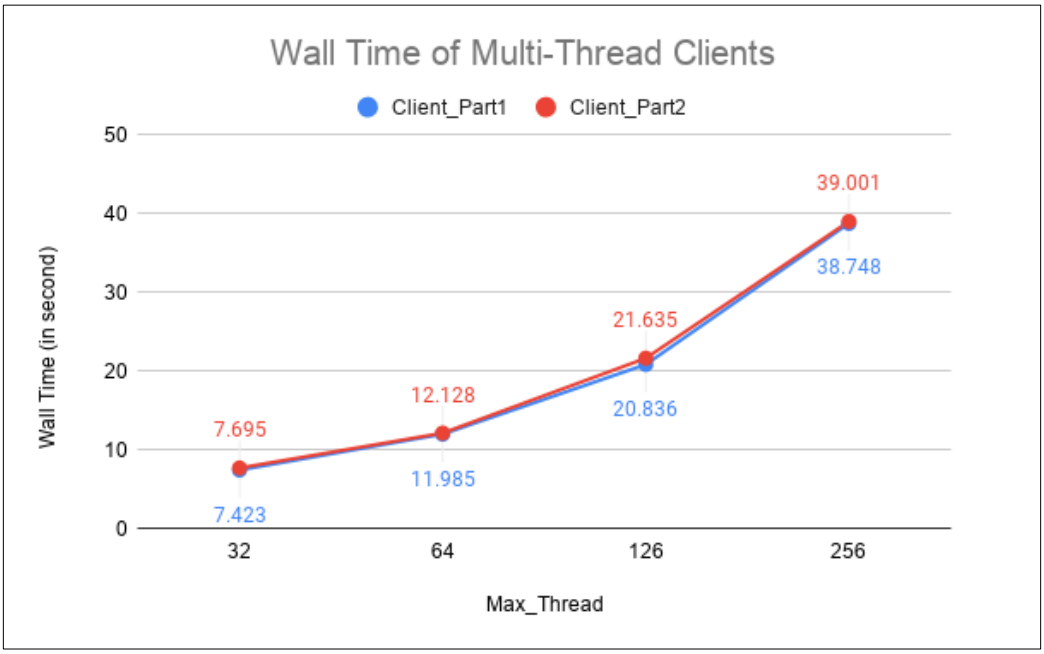
```
[ec2-user@ip-172-31-93-254 cs6650]$ java -jar ClientPart2.jar
---- Open stores in East ----
---- Waiting 3 hours for Central ----
---- Open stores in Central ----
---- Waiting 2 hours for West ----
---- Open stores in West ----
---- Waiting for all stores closed ----
---- All stores closed ----
The max thread: 128
The total number of successful requests sent: 69120
The total number of unsuccessful requests sent: 0
The wall time (in second): 21.635
The throughput (requests per second): 3194.823203143055
The mean response time: 25.31365740740741
The median response time: 20
The p99(99th percentile) response time: 106
The max response time: 646
69120 records written to csv file.
```

256 threads:

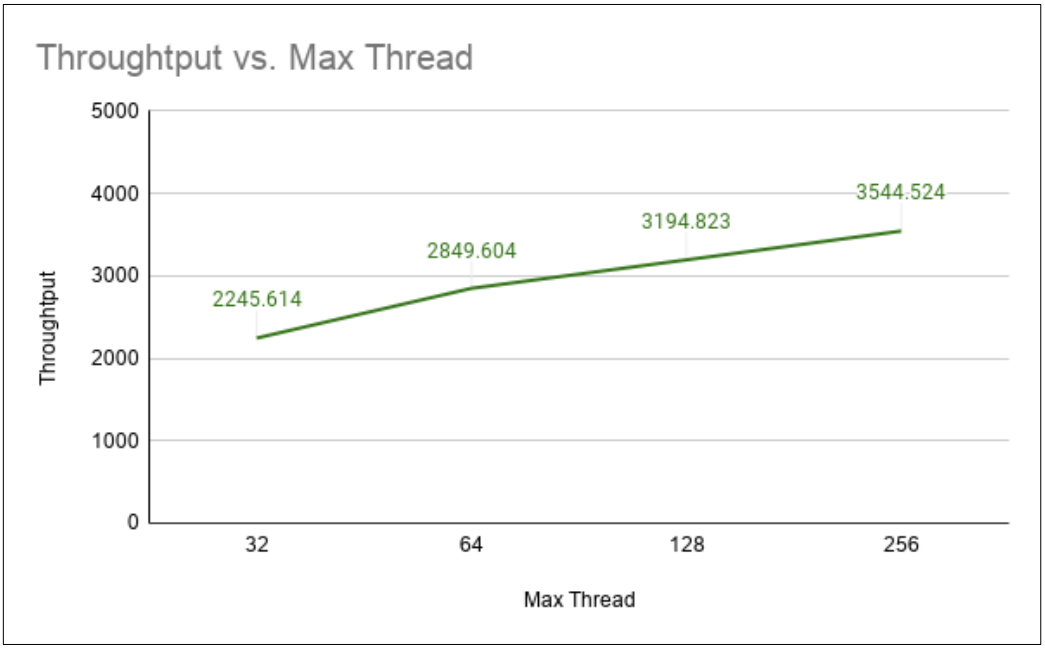
```
[ec2-user@ip-172-31-93-254 cs6650]$ java -jar ClientPart2.jar
---- Open stores in East ----
---- Waiting 3 hours for Central ----
---- Open stores in Central ----
---- Waiting 2 hours for West ----
---- Open stores in West ----
---- Waiting for all stores closed ----
---- All stores closed ----
The max thread: 256
The total number of successful requests sent: 138240
The total number of unsuccessful requests sent: 0
The wall time (in second): 39.001
The throughput (requests per second): 3544.524499371811
The mean response time: 44.702235243055554
The median response time: 22
The p99(99th percentile) response time: 351
The max response time: 913
138240 records written to csv file.
```

Output Charting:

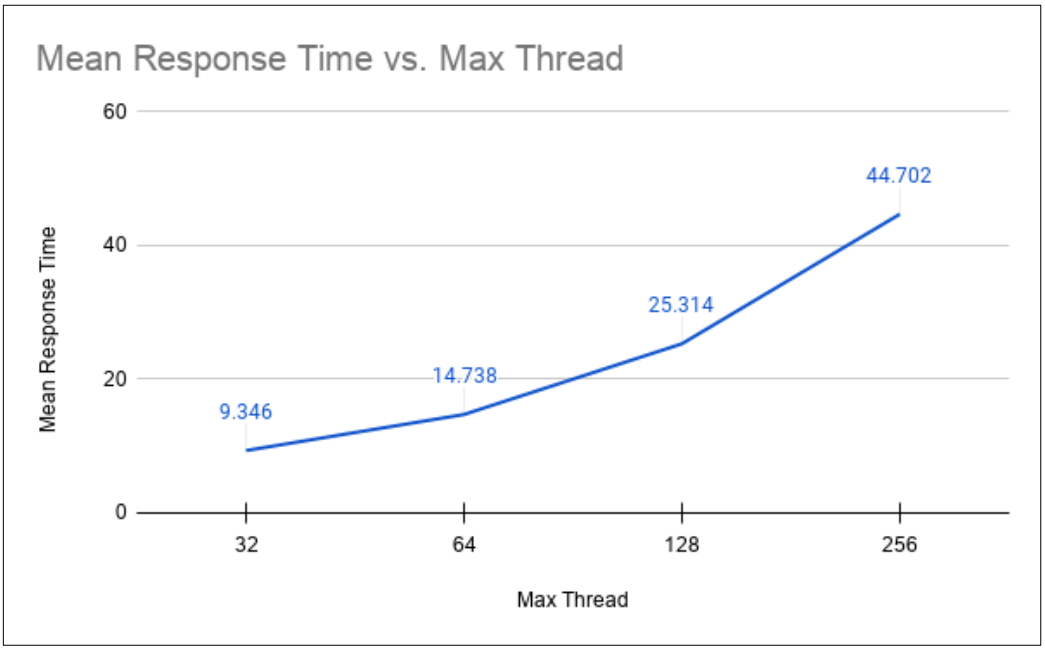
- Wall Time Against Number of Threads(Part1 and Part2):



- Throughput Against Number of Threads(Part2):



- Mean Response Time Against Number of Threads(Part2):



Break Things:

- **Break caused by insufficient memory ---- Threads: 4069 for ClientPart2**

```
[ec2-user@ip-172-31-93-254 cs6650]$ java -jar ClientPart2.jar
---- Open stores in East ----
---- Waiting 3 hours for Central ----
---- Open stores in Central ----
[171.906s][warning][os,thread] Failed to start thread - pthread_create fail
ed (EAGAIN) for attributes: stacksize: 1024k, guardsize: 0k, detached.
Exception in thread "main" java.lang.OutOfMemoryError: unable to create nat
ive thread: possibly out of memory or process/resource limits reached
        at java.base/java.lang.Thread.start0(Native Method)
        at java.base/java.lang.Thread.start(Thread.java:803)
        at StoreClientP2.main(StoreClientP2.java:166)
OpenJDK 64-Bit Server VM warning: INFO: os::commit_memory(0x00007f07ec27f00
0, 16384, 0) failed; error='Not enough space' (errno=12)
#
# There is insufficient memory for the Java Runtime Environment to continue
.
# Native memory allocation (mmap) failed to map 16384 bytes for committing
reserved memory.
# An error report file with more information is saved as:
# /home/ec2-user/cs6650/hs_err_pid12147.log
```

The ClientPart2 run successfully for 3000 threads on EC2, but failed for 4096 threads. The client broke in the phase 2 – creating the threads for the central zone stores, and the error message showed that there was insufficient memory to continue. I thought the potential solution could be lowering memory usage somewhere, thus I reviewed my code and found that I initialized the BlockingQueue<RequestRecord> allRecord to receive and contain all purchase request records in a thread-safe manner, I set the compacity to be the double size of all RequestRecord objects. It was not a big deal as the max threads number is 256 in this assignment , however, when the threads number increases to 4096, it is a different story. Thus, I revised my code, set a more reasonable capacity for the blockingqueue, and ran the new ClientPart2_2 on the EC2 for 4096 threads. Then I got another break as follow:

- **Break caused by server failure ---- Threads: 4069 for ClientPart2_2**

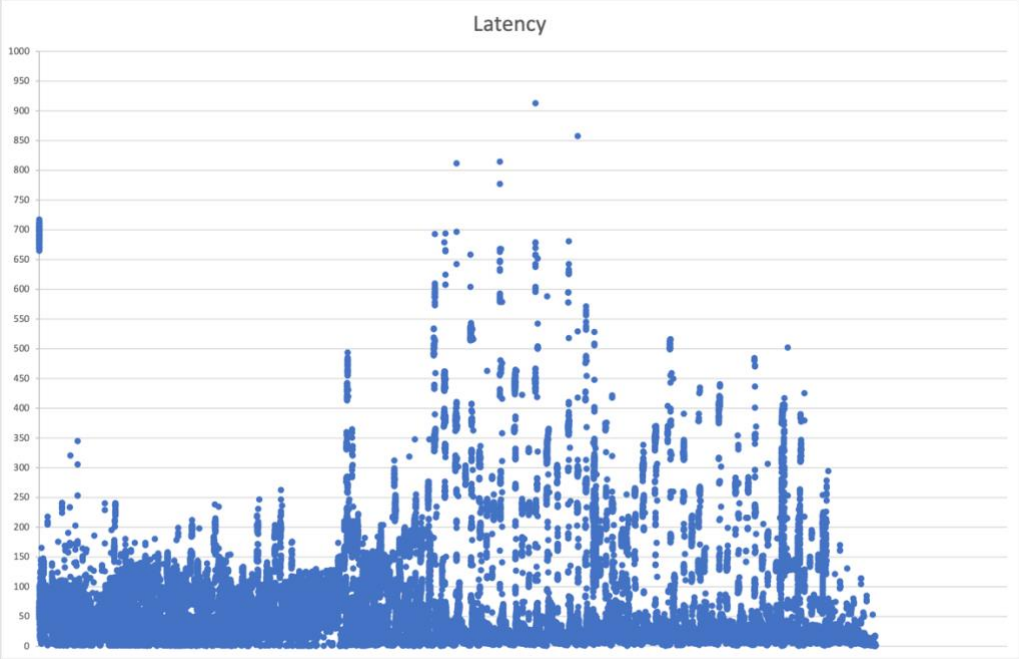
```
[ec2-user@ip-172-31-93-254 cs6650]$ java -jar ClientPart2_2.jar
---- Open stores in East ----
---- Waiting 3 hours for Central ----
---- Open stores in Central ----
---- Waiting 2 hours for West ----
---- Open stores in West ----
---- Waiting for all stores closed ----
Exception when calling PurchaseApi#newPurchase in sendPurchaseOrder()
io.swagger.client.ApiException: java.io.IOException: unexpected end of stream on com.squareup.okhttp.Address@6f9a458
f
        at io.swagger.client.ApiClient.execute(ApiClient.java:842)
        at io.swagger.client.ApiClient.execute(ApiClient.java:822)
        at io.swagger.client.api.PurchaseApi.newPurchaseWithHttpInfo(PurchaseApi.java:165)
        at StoreClientP2.sendPurchaseOrder(StoreClientP2.java:107)
        at StoreClientP2.run(StoreClientP2.java:62)
        at java.base/java.lang.Thread.run(Thread.java:834)
Caused by: java.io.IOException: unexpected end of stream on com.squareup.okhttp.Address@6f9a458f
        at com.squareup.okhttp.internal.http.Http1xStream.readResponse(Http1xStream.java:201)
        at com.squareup.okhttp.internal.http.Http1xStream.readResponseHeaders(Http1xStream.java:127)
        at com.squareup.okhttp.internal.http.HttpEngine.readNetworkResponse(HttpEngine.java:737)
        at com.squareup.okhttp.internal.http.HttpEngine.access$200(HttpEngine.java:87)
        at com.squareup.okhttp.internal.http.HttpEngine$NetworkInterceptorChain.proceed(HttpEngine.java:722)
        at com.squareup.okhttp.internal.http.HttpEngine.readResponse(HttpEngine.java:576)
        at com.squareup.okhttp.Call.getResponse(Call.java:287)
        at com.squareup.okhttp.Call$ApplicationInterceptorChain.proceed(Call.java:243)
        at com.squareup.okhttp.Call.getResponseWithInterceptorChain(Call.java:205)
        at com.squareup.okhttp.Call.execute(Call.java:80)
        at io.swagger.client.ApiClient.execute(ApiClient.java:838)
        ... 5 more
Caused by: java.io.EOFException: \n not found: size=0 content=...
        at okio.RealBufferedSource.readUtf8LineStrict(RealBufferedSource.java:201)
        at com.squareup.okhttp.internal.http.Http1xStream.readResponse(Http1xStream.java:186)
        ... 15 more
```

For this time, I saw the client had already created threads for all the stores, but failed to send all the purchase requests. I thought this break was probably caused by a server failure.

Charting for Latencies:

Generated based on the output csv for ClientPart2 run on EC2 – Max Thread 256

The Scatter Chart:



The Box Chart: