Assignment 1 - Yang Chen

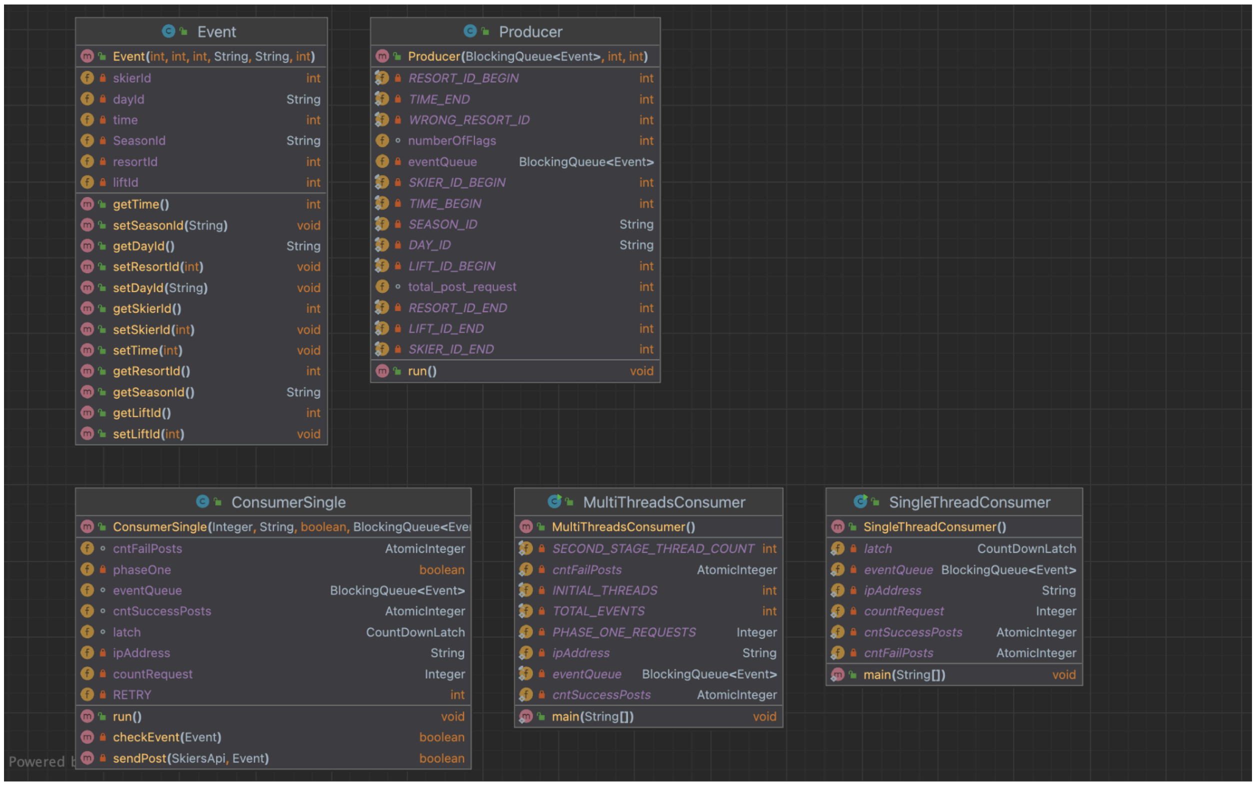
Git repo

https://github.com/Yangyanggogo/SkiResortDistributedSystem

Client design description

Client Part 1

Here is my client-part 1 UML diagram, which presents the classes included in client-part 1.



Event	
Event(int, int, int, String, String, int)	
skierId	int
dayId	String
time	int
SeasonId	String
resortId	int
liftId	int
getTime()	int
setSeasonId(String)	void
getDayId()	String
setResortId(int)	void
setDayId(String)	void
getSkierId()	int
setSkierId(int)	void
setTime(int)	void
getResortId()	int
getSeasonId()	String
getLiftId()	int
setLiftId(int)	void

Producer	
Producer(BlockingQueue<Event>, int, int)	
RESORT_ID_BEGIN	int
TIME_END	int
WRONG_RESORT_ID	int
numberOfFlags	int
eventQueue	BlockingQueue<Event>
SKIER_ID_BEGIN	int
TIME_BEGIN	int
SEASON_ID	String
DAY_ID	String
LIFT_ID_BEGIN	int
total_post_request	int
RESORT_ID_END	int
LIFT_ID_END	int
SKIER_ID_END	int
run()	void

ConsumerSingle	
ConsumerSingle(Integer, String, boolean, BlockingQueue<Eve	
cntFailPosts	AtomicInteger
phaseOne	boolean
eventQueue	BlockingQueue<Event>
cntSuccessPosts	AtomicInteger
latch	CountDownLatch
ipAddress	String
countRequest	Integer
RETRY	int
run()	void
checkEvent(Event)	boolean
sendPost(SkiersApi, Event)	boolean

MultiThreadsConsumer	
MultiThreadsConsumer()	
SECOND_STAGE_THREAD_COUNT	int
cntFailPosts	AtomicInteger
INITIAL_THREADS	int
TOTAL_EVENTS	int
PHASE_ONE_REQUESTS	Integer
ipAddress	String
eventQueue	BlockingQueue<Event>
cntSuccessPosts	AtomicInteger
main(String[])	void

SingleThreadConsumer	
SingleThreadConsumer()	
latch	CountDownLatch
eventQueue	BlockingQueue<Event>
ipAddress	String
countRequest	Integer
cntSuccessPosts	AtomicInteger
cntFailPosts	AtomicInteger
main(String[])	void

I implemented the client-part 1 using a general design pattern of producer-consumer. The **producer** is a single thread, responsible for generating the events and storing them in a BlockingQueue (thread-safe), which consumers can access.

The **ConsumerSingle** class is responsible for consuming the event (defined in **Event class**) in the BlockingQueue and sending a post request to my server, with an event in each request's body.

According to the assignment's requirements, phase one has 32 threads and each thread sends 1000 post requests to the server. As soon as one of these 32 threads finishes its task, I can allocate x number of threads and start phase two (x is a variable to test my multithreads client-part 1).

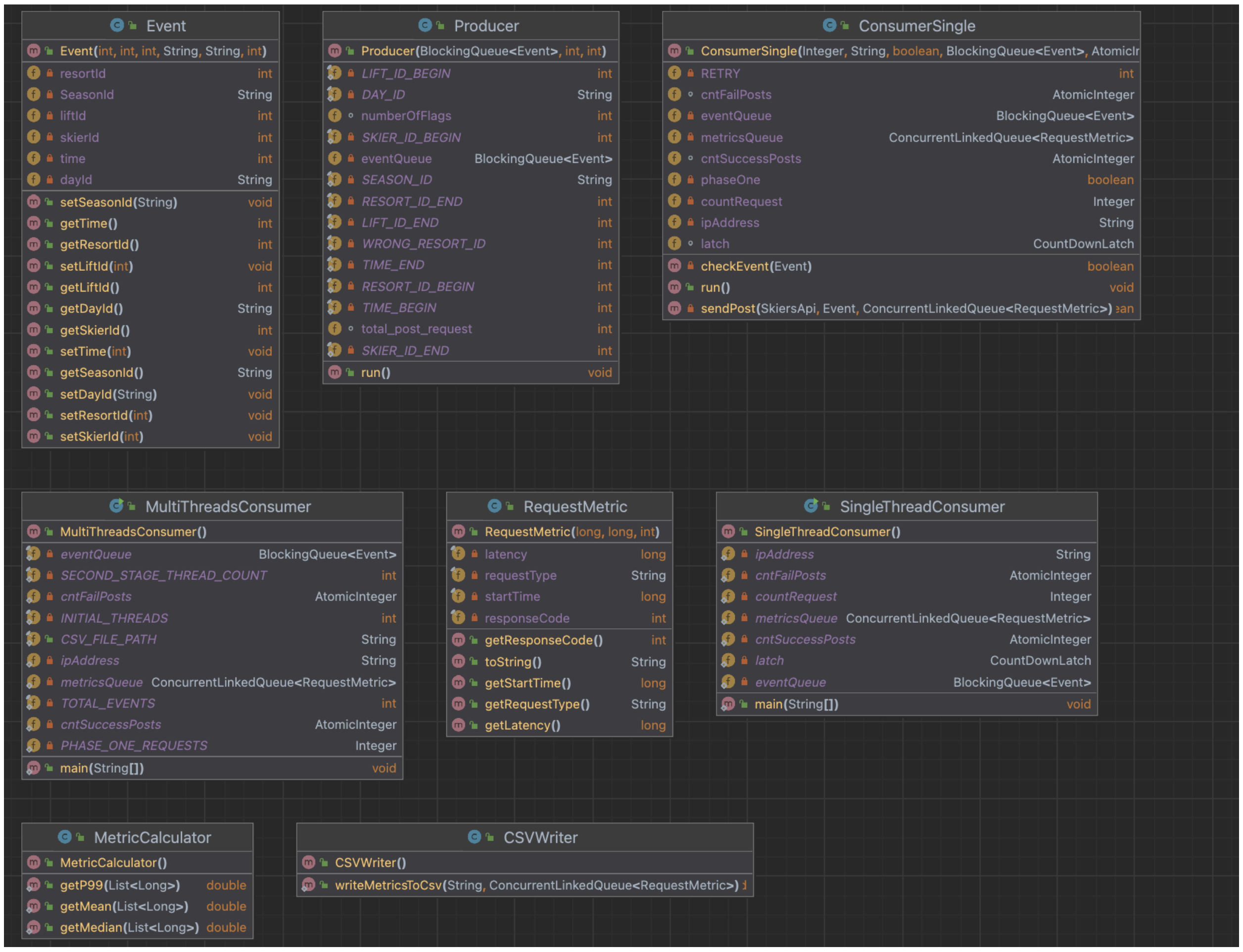
My ConsumerSingle class can handle both requirements by passing the boolean "phaseOne" to this class.

32 threads in phase one can be terminated when finishing sending 1000 requests. To terminate my threads in phase two, I added extra "invalid events" in the BlockingQueue, when any thread gets an "invalid event" in the BlockingQueue, it will be terminated.

In the **MultiThreadConsumer class**, I can test the amount of throughput by changing the value of x, the number of threads added in phase two. This adding threads operation is trigged by using CountDownLatch, latch(1), following the requirement of assignment 1.

Client part 2

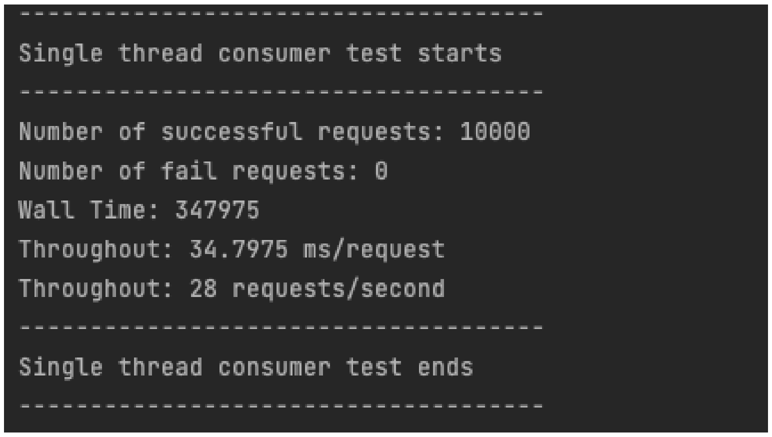
Following my client part 1 design, a **CSVWriter** class is added. The required record per request is generated, encapsulated in a new **RequestMetricClass**, and stored in a ConcurrentLinkedQueue. After all posts are sent and all request records are stored in ConcurrentLinkedQueue, the CSVWriter class is called to generate all records in a CSV file. The **MetricCalculator** class handles the calculation of different properties of latency.



Output

Client part 1

SingleThreadConsumer



```
------------------------------------
Single thread consumer test starts
------------------------------------
Number of successful requests: 10000
Number of fail requests: 0
Wall Time: 347975
Throughout: 34.7975 ms/request
Throughout: 28 requests/second
------------------------------------
Single thread consumer test ends
------------------------------------
```

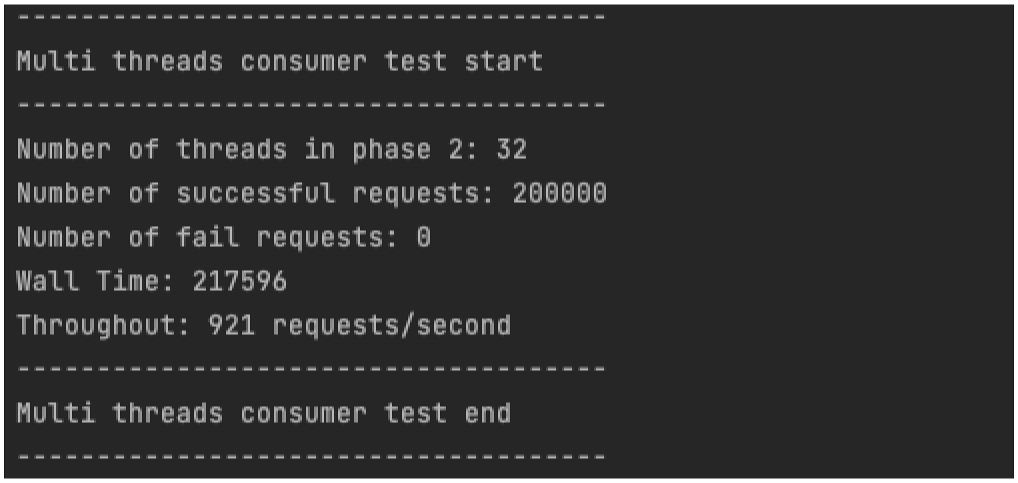
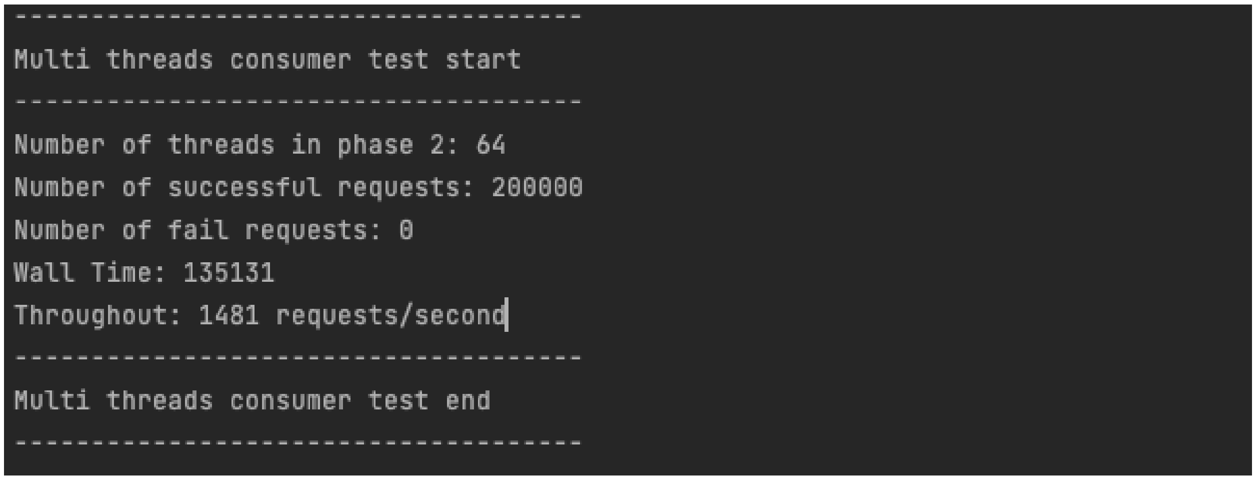
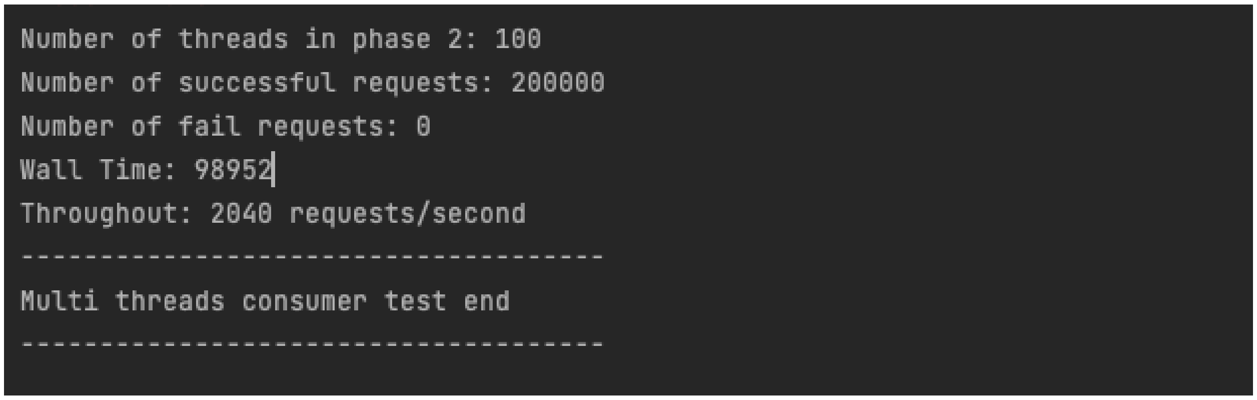
The average response time from a single-thread client is 35 ms/request.

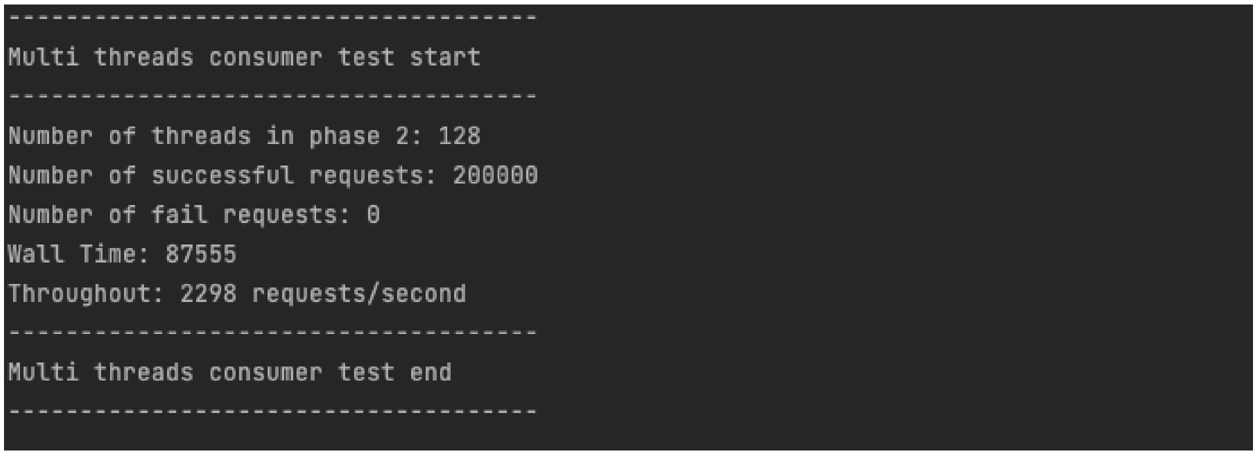
Multi threads client - part 1

Several options for the number of threads in phase two are tested. The best throughput outcome is 3846 requests/second, with 800 threads in phase two.
According to Little's Law, the predicted throughput is 200/35*1000 = 5714 requests/second, which is larger than the test.

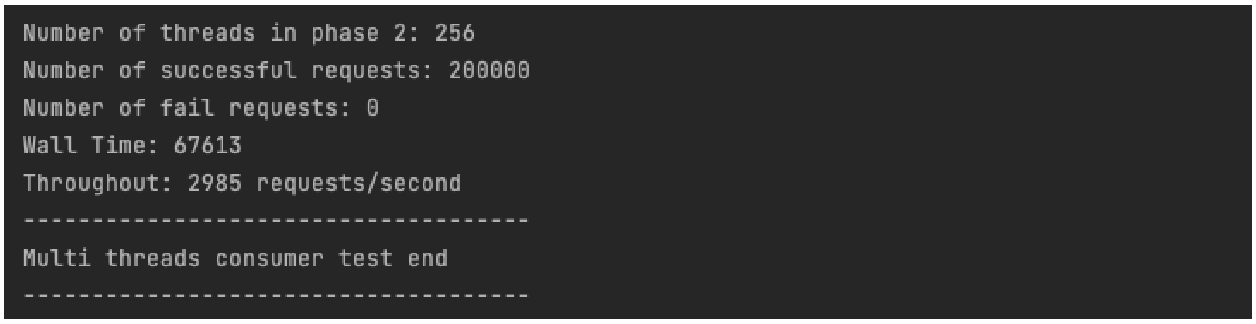
Tomcat serve has max 200 threads, but when increasing the client side threads number from 128, 256, 512, to 800, the throughput output of the multi-threads client keeps increasing. Given the Little Law, the minimal value between the number of threads on the server side and the client side, which is 200, should be the bound of the throughput of my program. However, the test of 256, 512, and 800 threads on the client side generated different throughput.

One possible explanation could be more number of client-side threads ensures that the server has a constant queue of requests to process, maximizing its utilization. When the number of client-side threads is too close to the server's capacity, any temporary dip in server-side processing (due to GC pauses, CPU scheduling, etc.) can lead to underutilization. More client threads mean the server is more likely to be kept busy, up to the point where the overhead of managing these additional threads or other resource constraints outweighs the benefits.

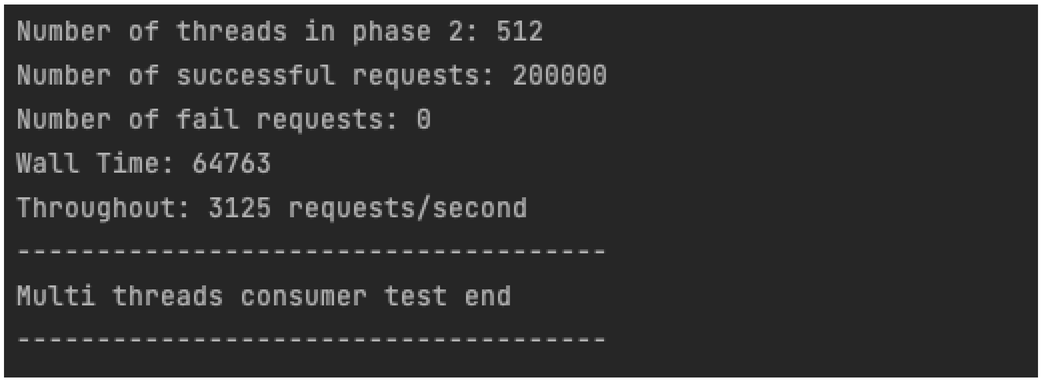

```
--------------------------------------
Multi threads consumer test start
--------------------------------------
Number of threads in phase 2: 32
Number of successful requests: 200000
Number of fail requests: 0
Wall Time: 217596
Throughout: 921 requests/second
--------------------------------------
Multi threads consumer test end
--------------------------------------
```

```
-------------------------------------
Multi threads consumer test start
-------------------------------------
Number of threads in phase 2: 64
Number of successful requests: 200000
Number of fail requests: 0
Wall Time: 135131
Throughout: 1481 requests/second
-------------------------------------
Multi threads consumer test end
-------------------------------------
```

```
Number of threads in phase 2: 100
Number of successful requests: 200000
Number of fail requests: 0
Wall Time: 98952
Throughout: 2040 requests/second
--------------------------------------
Multi threads consumer test end
--------------------------------------
```



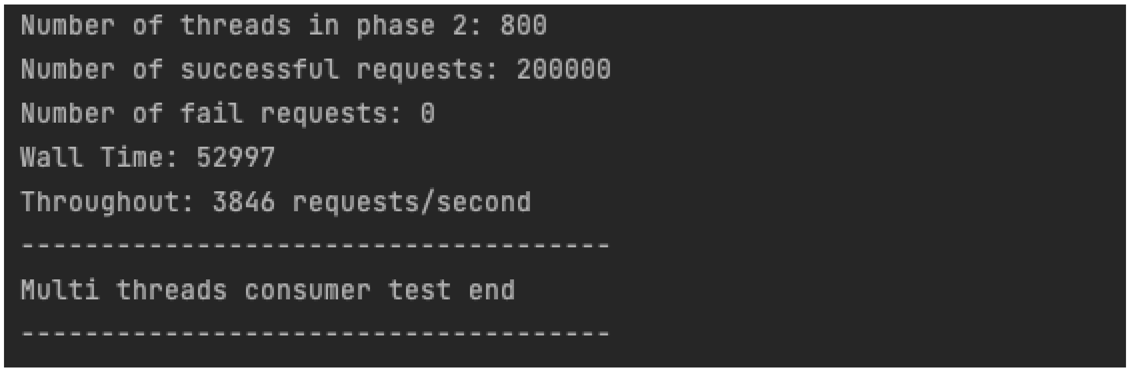
```
-------------------------------------
Multi threads consumer test start
-------------------------------------
Number of threads in phase 2: 128
Number of successful requests: 200000
Number of fail requests: 0
Wall Time: 87555
Throughout: 2298 requests/second
-------------------------------------
Multi threads consumer test end
-------------------------------------
```



```
Number of threads in phase 2: 256
Number of successful requests: 200000
Number of fail requests: 0
Wall Time: 67613
Throughout: 2985 requests/second
-------------------------------------
Multi threads consumer test end
-------------------------------------
```

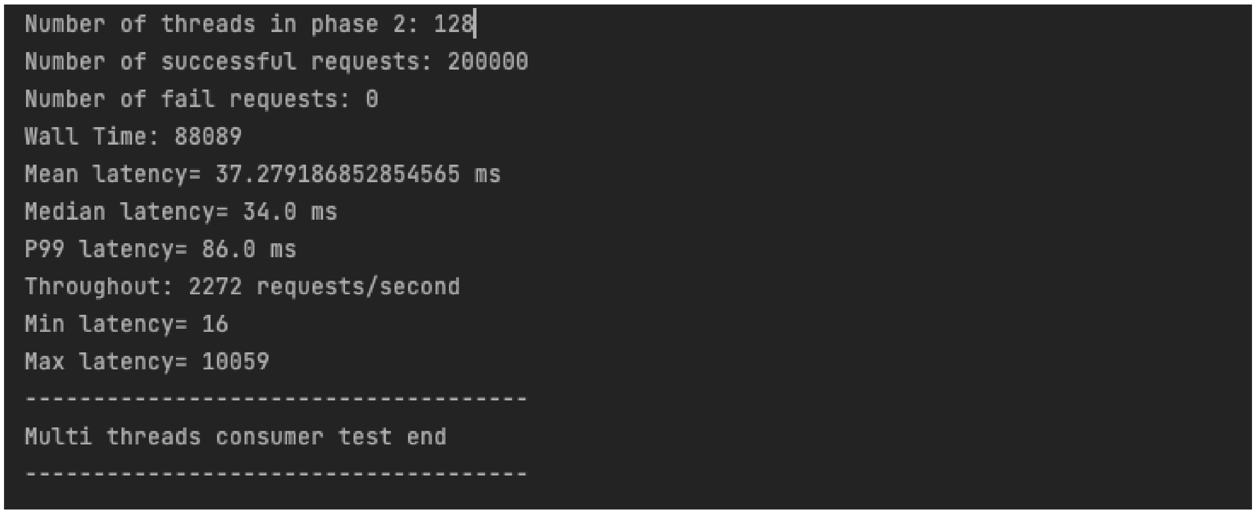


```
Number of threads in phase 2: 512
Number of successful requests: 200000
Number of fail requests: 0
Wall Time: 64763
Throughout: 3125 requests/second
-------------------------------------
Multi threads consumer test end
-------------------------------------
```

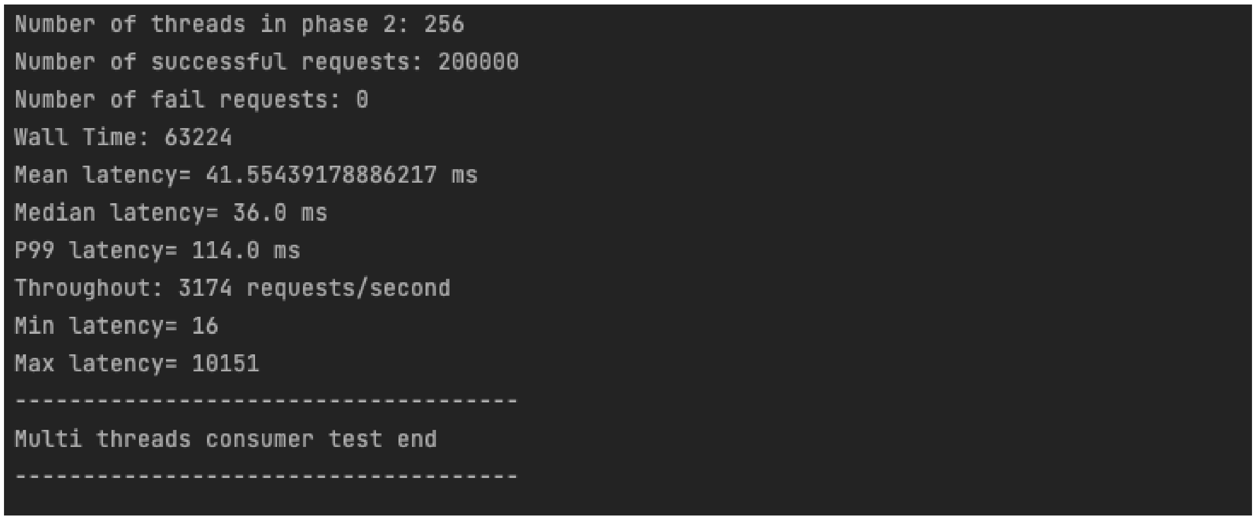


```
Number of threads in phase 2: 800
Number of successful requests: 200000
Number of fail requests: 0
Wall Time: 52997
Throughout: 3846 requests/second
-------------------------------------
Multi threads consumer test end
-------------------------------------
```

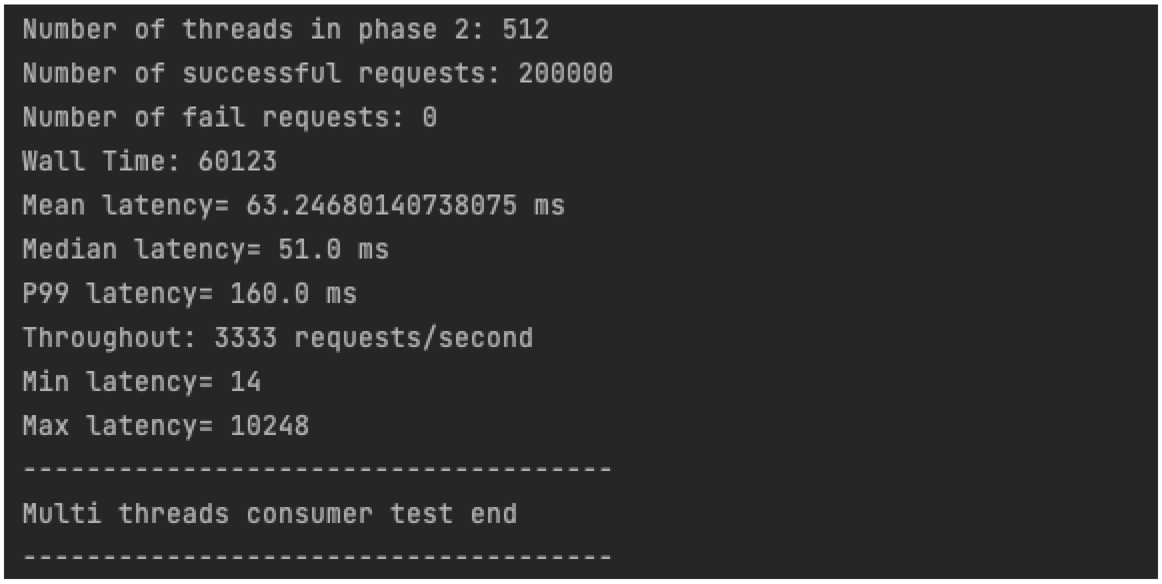
Multi threads client - part 2



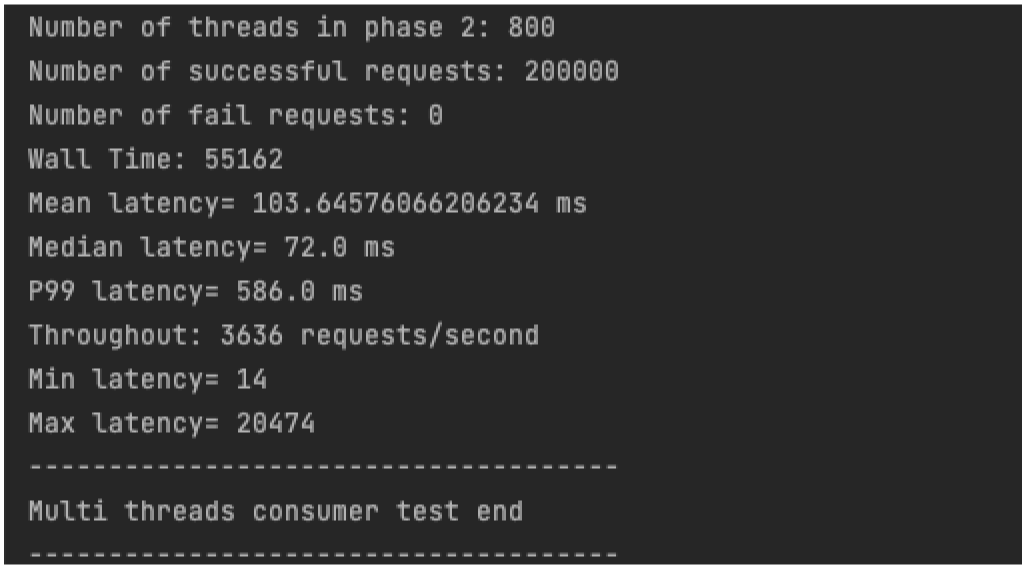
```
Number of threads in phase 2: 128
Number of successful requests: 200000
Number of fail requests: 0
Wall Time: 88089
Mean latency= 37.279186852854565 ms
Median latency= 34.0 ms
P99 latency= 86.0 ms
Throughout: 2272 requests/second
Min latency= 16
Max latency= 10059
-------------------------------------
Multi threads consumer test end
-------------------------------------
```



```
Number of threads in phase 2: 256
Number of successful requests: 200000
Number of fail requests: 0
Wall Time: 63224
Mean latency= 41.55439178886217 ms
Median latency= 36.0 ms
P99 latency= 114.0 ms
Throughout: 3174 requests/second
Min latency= 16
Max latency= 10151
-------------------------------------
Multi threads consumer test end
-------------------------------------
```



```
Number of threads in phase 2: 512
Number of successful requests: 200000
Number of fail requests: 0
Wall Time: 60123
Mean latency= 63.24680140738075 ms
Median latency= 51.0 ms
P99 latency= 160.0 ms
Throughout: 3333 requests/second
Min latency= 14
Max latency= 10248
-------------------------------------
Multi threads consumer test end
-------------------------------------
```



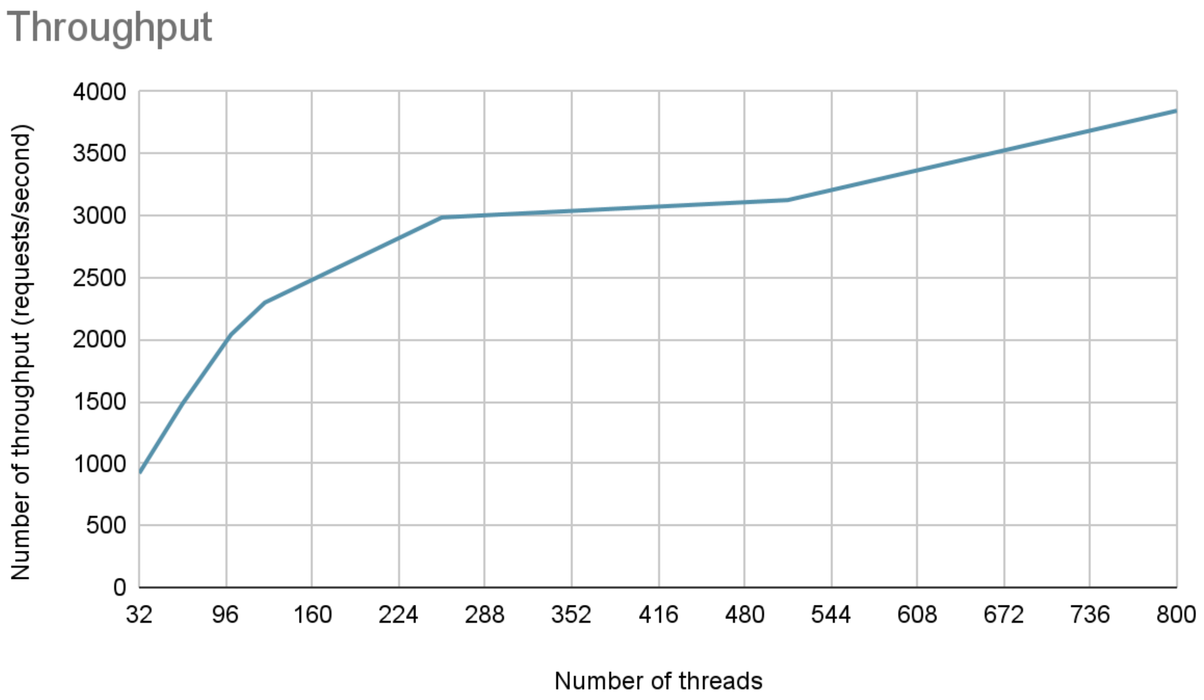
```
Number of threads in phase 2: 800
Number of successful requests: 200000
Number of fail requests: 0
Wall Time: 55162
Mean latency= 103.64576066206234 ms
Median latency= 72.0 ms
P99 latency= 586.0 ms
Throughout: 3636 requests/second
Min latency= 14
Max latency= 20474
-------------------------------------
Multi threads consumer test end
-------------------------------------
```

Number of threads	Throughput - client 1	Throughput - client 2
125	2298	2272
256	2985	3174
512	3125	3333
800	3846	3636

The difference between the number of threads in client 1 and client 2 is within 5%.

The plot of throughput over the number of threads in phase 2 (client 1)

Number of threads	Throughput - client 1
32	921
64	1481
100	2040
125	2298
256	2985
512	3125
800	3846


Throughput
Number of throughput (requests/second)
4000
3500
3000
2500
2000
1500
1000
500
0
32
96
160
224
288
352
416
480
544
608
672
736
800
Number of threads