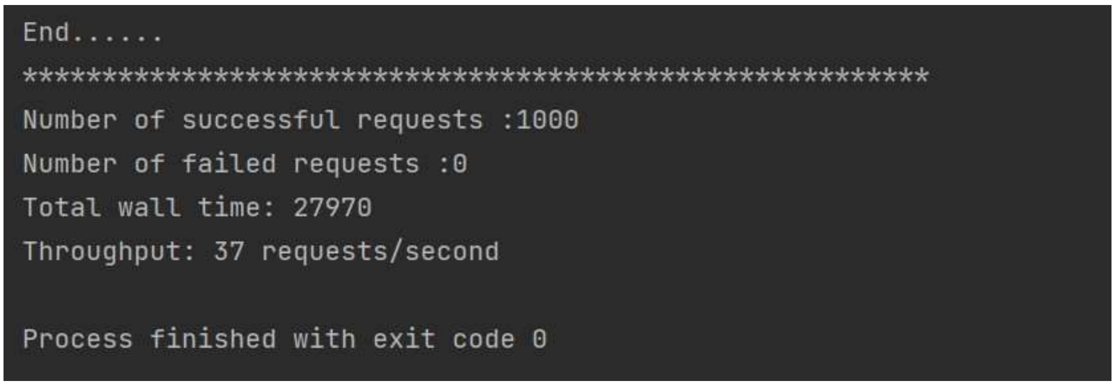
The url for my github repo:
https://github.com/BenjeminZhe/building-scalable-distributed-systems
Main working logics for client:
Client-part1:
I apply a general pattern of producer-consumer model in my design. The producer is designed as a single thread that produces events as posts and store these events into a global blockingqueue for use by consumers. The consumer constructs multiple threads to consume the events, which are popped consecutively from the queue, and make posts accordingly. To distinguish between the two phases of consumer threads, I design different logics for the threads to behave, and add a Boolean flag to set the phases apart. When all the posts are consumed, the threads will stop when they encounter the stopping signals from the queue. Besides, the queue must be thread-safe, so a blockingqueue will be a good choice.
Client-part2:
Based on my design from part1, I record the required data from every response and create a thread-safe global list to store these data. After getting all the response data, the CSVWriter is applied to output the data into a csv file and the StatiaticsProcessor is applied to get all the needed statistics and print these on the screen.
The output from the SingleThreadConsumer:



```
End......
********************************************************
Number of successful requests :1000
Number of failed requests :0
Total wall time: 27970
Throughput: 37 requests/second

Process finished with exit code 0
```

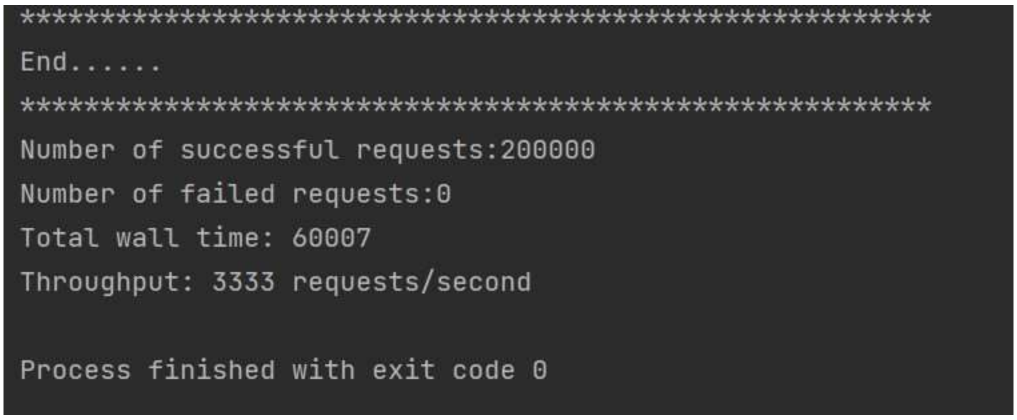
So the average response time from a single post is approximately 28ms. Since we have 100 threads in the second phase of my client, the predicted throughput from little's law is **100/27.97 * 1000 = 3575** requests/second.

Major classes Used in my client:
1. Model classes: including three classes-Events, OutputRecords, and SyncList. We use Events as a container for parameters of posts, OutputRecords as a container for performance metrics of each response we receive from posts, and SyncList that collect every OutputRecords from the threads synchronously.
2. Producer: A runnable thread that produces posts and stopping signals(as Events), and store these events into a global blockingqueue, for the consumer threads to use.
3. ConsumerRunnable: A single thread that get an event from the global queue at a time, make a post if the event is an authentic post, or stop running if the event is a stopping signal.
4. SingleThreadConsumer: A consumer of posts that use only one ConsumerRunnable

thread. Used to estimate the latency from geographical distance.

5. CSVWriter: A writer that writes all the records into csv.
6. StatisticsProcessor: Calculate the statistic factors to print on screen for client-part2
7. Consumer: A consumer with multiple threads. In phase 1, it has 32 threads; in phase 2, it has 100 threads in total.

Client-part1 result:

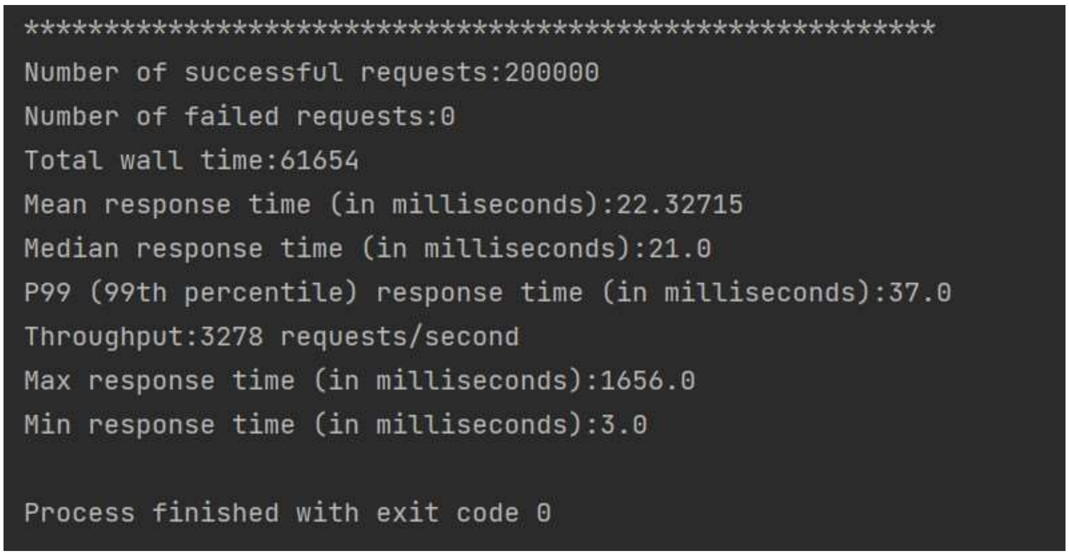


```
*********************************************************
End......
*********************************************************
Number of successful requests:200000
Number of failed requests:0
Total wall time: 60007
Throughput: 3333 requests/second

Process finished with exit code 0
```

The number of threads I use: Phase 1: 32, Phase 2: 100.

Client-part2 result:



```
*********************************************************
Number of successful requests:200000
Number of failed requests:0
Total wall time:61654
Mean response time (in milliseconds):22.32715
Median response time (in milliseconds):21.0
P99 (99th percentile) response time (in milliseconds):37.0
Throughput:3278 requests/second
Max response time (in milliseconds):1656.0
Min response time (in milliseconds):3.0

Process finished with exit code 0
```

The number of threads I use: Phase 1: 32, Phase 2: 100.