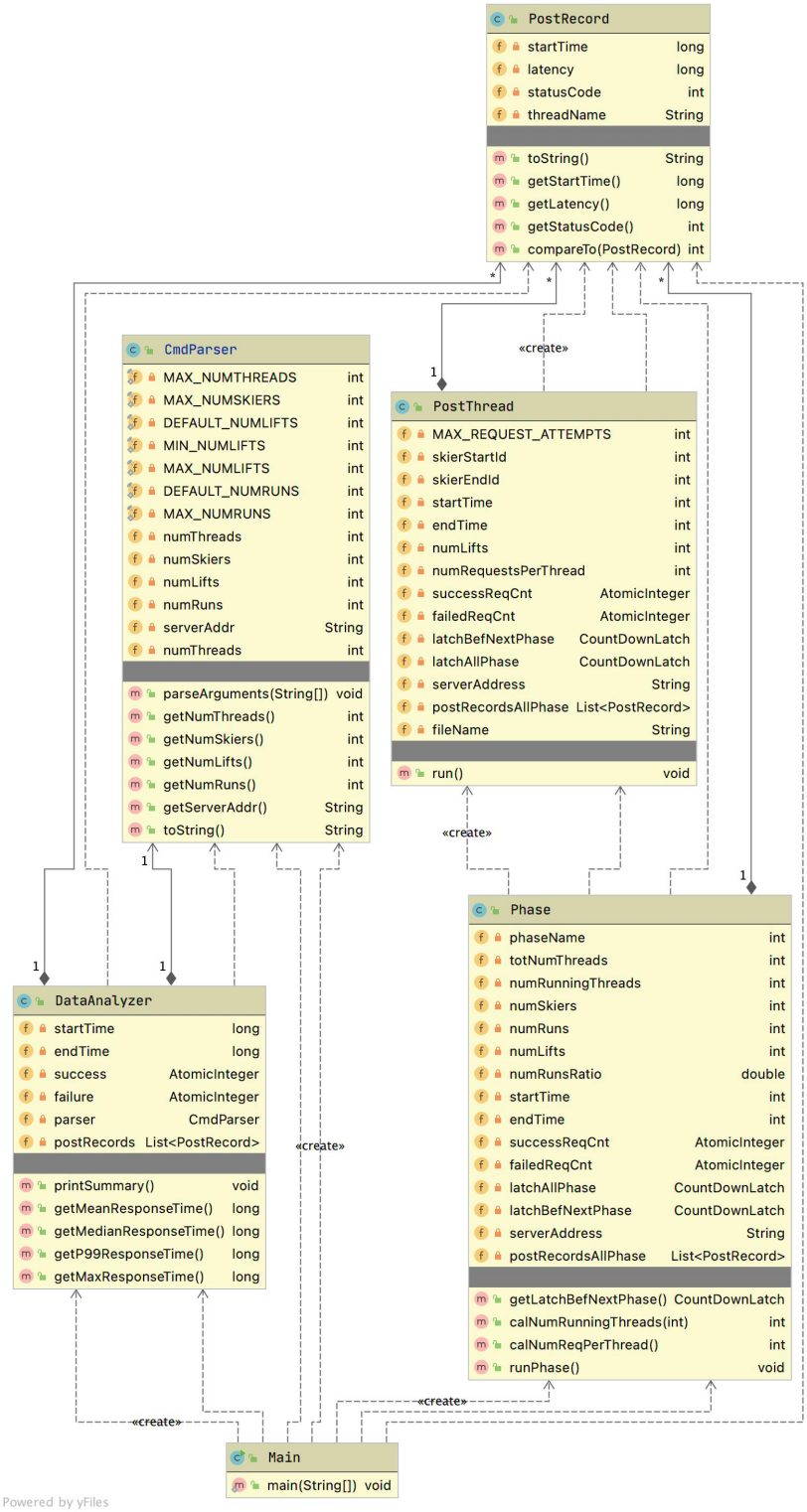
CS6650 Fall 2021 Assignment1

Yan Wei

1. Github Repository https://github.com/ywei20/Ski_Resort
2. Description of client design

Below is the UML diagram of Client2, which covers all the functions of Client1.



The Main class is where the program starts. First a CmdParser object is instantiated to parse command line arguments. A valid set of program arguments is as “numThreads 256 numSkiers 20000 numLifts 40 serverAddress http://54.218.16.242:8080”. All these arguments can be easily accessed by their respective getter methods. In the main method, a CountDownLatch for all phases is created and passed as a parameter to each phase so that the DataAnalyzer object won’t be created until all three phases complete. Additionally, a synchronizedList is also created and passed as a parameter to each phase to store each post request information, such as start timestamp, request type, latency, status code and threadName sending this post request.

Then three Phase instances are instantiated and the runPhase() method of three phases are called in order. In each Phase instance, a CountDownLatch called latchBefNextPhase is created. By calling latchBefNextPhase.await() in the main method between each phase, the start of the next phase will be delayed according to our required time. In the runPhase() method, multiple threads are started to execute their tasks.

The PostThread class contains details of what a working thread should do. Each thread uses the Swagger API to send multiple number of post requests. For each post request, a PostRecord instance is created to store information associated with the request, and then the postRecord instance is added to the SynchronizedList called postRecordsAllPhase. In addition, every postRecord instance is written to csv file. When each thread completes their job, we call countDown() on latchAllPhase and latchBefNextPhase. Note we have only two not-null latchBefNextPhase, one is between phase1 and phase2, the other is between phase2 and phase3.

After three phases complete their jobs, the main thread creates a DataAnalyzer object, which is responsible for stats summary, including # of successful/failed requests, wall time, throughput, mean/median/99 percent/max response time.

3. Client1 results

3.1 Little’s law throughput prediction using 10000 requests

```
******* Test Results Sending 10000 Requests From A Single Thread *******
# of successful requests: 10000
# of failed requests: 0
Total run time(ms)(wall time): 274547
Total throughput(# of requests/sec): 36
```

3.2 32 Threads

```
CmdParser{numThreads=32, numSkiers=20000, numLifts=40, numRuns=10, serverAddr='http://54.218.16.242:8080'}
Phase 1 start...
Phase 2 start...
Phase 3 start...
All three phases end...
============Print Out Summary===========
# of successful requests: 160008
# of failed requests: 0
Total run time(ms)(wall time): 256910
Total throughput(# of requests/sec): 623
```

3.3 64 Threads

```
CmdParser{numThreads=64, numSkiers=20000, numLifts=40, numRuns=10, serverAddr='http://54.218.16.242:8080'}
Phase 1 start...
Phase 2 start...
Phase 3 start...
All three phases end...
============Print Out Summary===========
# of successful requests: 160016
# of failed requests: 0
Total run time(ms)(wall time): 150049
Total throughput(# of requests/sec): 1066
```

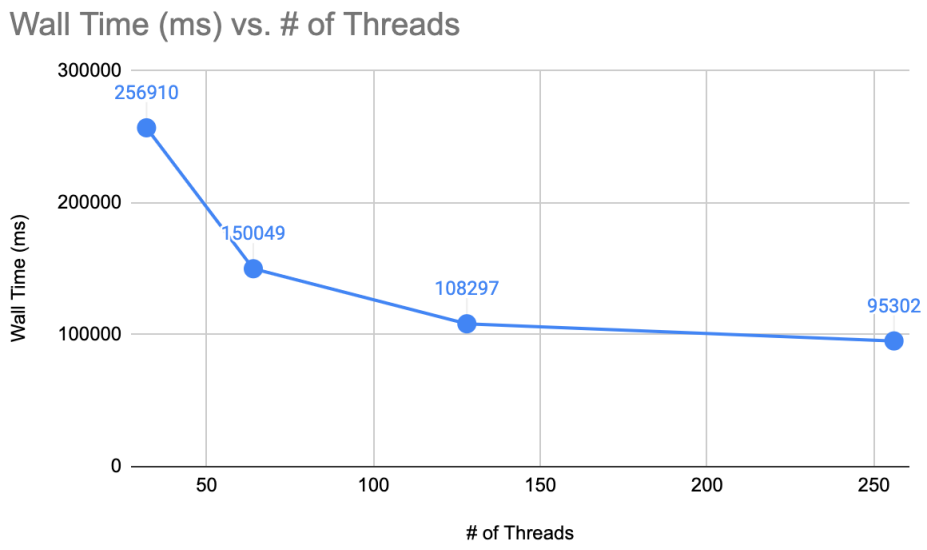
3.4 128 Threads

```
CmdParser{numThreads=128, numSkiers=20000, numLifts=40, numRuns=10, serverAddr='http://54.218.16.242:8080'}
Phase 1 start...
Phase 2 start...
Phase 3 start...
All three phases end...
============Print Out Summary===========
# of successful requests: 160096
# of failed requests: 0
Total run time(ms)(wall time): 108297
Total throughput(# of requests/sec): 1478
```

3.5 256 Threads

```
CmdParser{numThreads=256, numSkiers=20000, numLifts=40, numRuns=10, serverAddr='http://54.218.16.242:8080'}
Phase 1 start...
Phase 2 start...
Phase 3 start...
All three phases end...
============Print Out Summary===========
# of successful requests: 160128
# of failed requests: 0
Total run time(ms)(wall time): 95302
Total throughput(# of requests/sec): 1680
```

3.6 Wall time VS. # of threads



4. Client2 results

4.1 32 Threads

```
CmdParser{numThreads=32, numSkiers=20000, numLifts=40, numRuns=10, serverAddr='http://54.218.16.242:8080'}
Phase 1 start...
Phase 2 start...
Phase 3 start...
All three phases end...
============Print Out Summary===========
# of successful requests: 160008
# of failed requests: 0
Total run time(wall time)(millisec): 258787
Total throughput(# of requests/sec): 618
Mean Response Time(millisec): 30
Median Response Time(millisec): 28
99 Percentile Response Time(millisec): 70
Max Response Time(millisec): 690
```

4.2 64 Threads

```
CmdParser{numThreads=64, numSkiers=20000, numLifts=40, numRuns=10, serverAddr='http://54.218.16.242:8080'}
Phase 1 start...
Phase 2 start...
Phase 3 start...
All three phases end...
============Print Out Summary===========
# of successful requests: 160016
# of failed requests: 0
Total run time(wall time)(millisec): 155140
Total throughput(# of requests/sec): 1031
Mean Response Time(millisec): 38
Median Response Time(millisec): 34
99 Percentile Response Time(millisec): 208
Max Response Time(millisec): 834
```

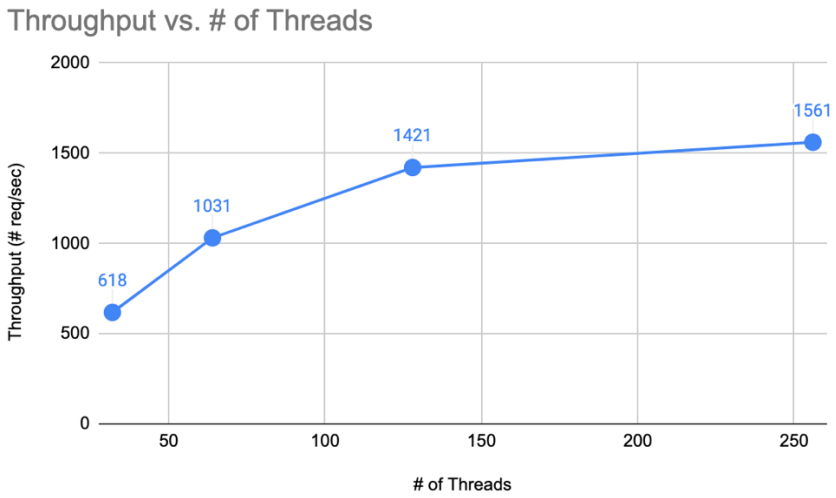
4.3 128 Threads

```
CmdParser{numThreads=128, numSkiers=20000, numLifts=40, numRuns=10, serverAddr='http://54.218.16.242:8080'}
Phase 1 start...
Phase 2 start...
Phase 3 start...
All three phases end...
============Print Out Summary===========
# of successful requests: 160096
# of failed requests: 0
Total run time(wall time)(millisec): 112697
Total throughput(# of requests/sec): 1421
Mean Response Time(millisec): 66
Median Response Time(millisec): 73
99 Percentile Response Time(millisec): 221
Max Response Time(millisec): 795
```

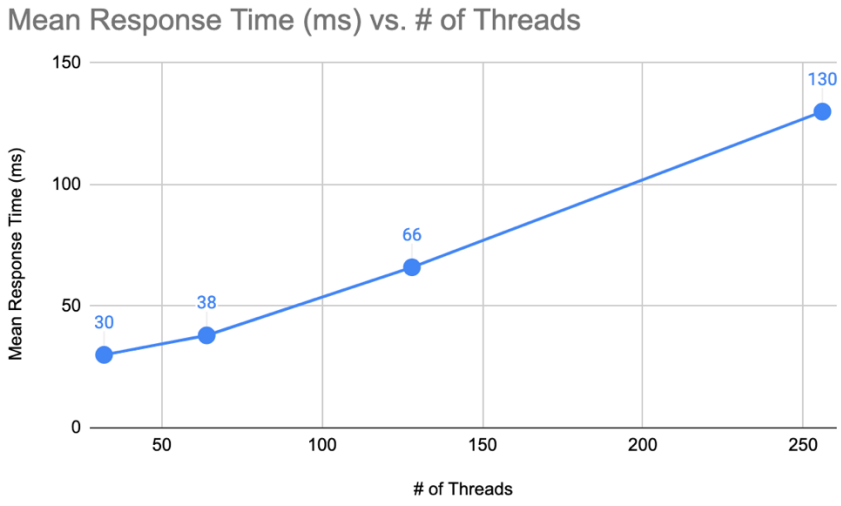
4.4 256 Threads

```
CmdParser{numThreads=256, numSkiers=20000, numLifts=40, numRuns=10, serverAddr='http://54.218.16.242:8080'}
Phase 1 start...
Phase 2 start...
Phase 3 start...
All three phases end...
============Print Out Summary===========
# of successful requests: 160128
# of failed requests: 0
Total run time(wall time)(millisec): 102593
Total throughput(# of requests/sec): 1561
Mean Response Time(millisec): 130
Median Response Time(millisec): 143
99 Percentile Response Time(millisec): 457
Max Response Time(millisec): 1496
```

4.5 Throughput VS. # of Threads



4.6 Mean Response Time VS. # of Threads



Bonus:
Plot latency by time

The Bonus class reads in csv file of 256 threads, sort all post records by their associated start timestamp, then calculate average response time within 1 second, and write the result to a new csv file. In this way, I convert around 160000 records to 160 records then plot the average latency (ms) VS time (sec).