Assignment 1 Write Up

CS 6650, Spring 2022
Sean Stevens

Github Repository URL

https://github.com/seanmstevens/CS6650-Assignment1

Client Design

My design for the client portion of this assignment heavily relies on Java's synchronization aids to ensure race conditions and deadlock do not occur. The Java CountDownLatch, ExecutorService, and AtomicInteger classes are used to keep thread contention to a minimum so that request throughput is maximized.

Major Classes

- ***Args:*** A class to parse command line arguments and convert values to the types expected by the main program. Uses the JCommander command line parser library to parse the arguments and has implementations for custom validators and converters.
- ***Client:*** The "entry point" class for the program that contains the main functionality of the program. Creates instances of other classes to parse arguments, generates synchronization aids, executes phases with their specific parameters, and prints out the final report with run statistics.
- ***DataProcessor:*** Holds data (in the form of a list of LatencyRecord items) and allows synchronized access for threads to add their request data to the list. Has methods to get statistics related to the run performance, including: mean response time, median response time, max and min response times, and the 99th percentile response time.
- ***LatencyRecord:*** A POJO class that serves as a container for request data. Has fields for the start time of the request, the latency, the request type, and the response code.
- ***LatencyTest:*** A utility class that generates some sample statistics for single-threaded performance of API calls. Used to establish a baseline for response latency and expected throughput.
- ***PhaseOptions:*** Another POJO class that holds the options/parameters relevant to a particular phase of the program. Allows a decoupled way to execute a phase without needing the main method to pass many parameters around to its helper methods.
- ***WorkerRunnable:*** The Runnable task that is instantiated with different parameters when added to the fixed size thread pool for execution. The runnable performs the request with the specified bounds, retries failed requests up to 5 times, and updates the synchronization aids while adding the generated request data to the DataProcessor instance.

Little's Law Predictions

Below are some Little's Law predictions for 32, 64, 128, and 256 thread runs with 20,000 skiers and 40 lifts. The predictions are used as a comparison metric to measure how efficient the client is at sending requests (given a theoretical server with infinite request processing capacity). **Note:** The mean response time is derived from the client's own data output.

Little's Law:

$$L = A \, x \, W$$

Where L is the number of concurrent requests (threads), A is the number of requests entering and leaving the system, and W is the average time each request spends in the system (response time).

Threads	**Mean Response Time** (ms)	**Throughput** (reqs/sec)
32	35	914.28
64	35	1828.57
128	36	3657.14
256	37	7314.28

Packages

As mentioned above, this client implementation takes advantage of a few packages for ease of development. For command line argument parsing, the JCommander library is used as it provides good options, customization, and flexibility that can easily support this project's needs. Additionally, the Apache Commons CSV library provides a framework to print CSV files to disk while keeping client code relatively uncluttered. Finally, the Swagger auto-generated client library is included to allow for thread-safe API calls and relatively simple API set up code.

Client Part 1 Results

Through testing, I found that I had significant discrepancies between the results generated when all three phases ran in sequence compared to when the peak phase (phase 2) ran by itself. The phase 2 results are closer to what I expect from the Little's law calculations, so I have provided both sets of reports below.

32 Threads (All Phases):

```
Total successes: 160003
Total failures: 0
Time elapsed (sec): 316.027
Total throughput (reqs/sec): 506.2953481822756

Process finished with exit code 0
```

32 Threads (Phase 2 Only):

```
Total successes: 120000
Total failures: 0
Time elapsed (sec): 142.814
Total throughput (reqs/sec): 840.2537566345037

Process finished with exit code 0
```

64 Threads (All Phases):

```
Total successes: 159814
Total failures: 0
Time elapsed (sec): 158.356
Total throughput (reqs/sec): 1009.2071029831519

Process finished with exit code 0
```

64 Threads (Phase 2 Only):

```
Total successes: 119808
Total failures: 0
Time elapsed (sec): 75.693
Total throughput (reqs/sec): 1582.8147913281282

Process finished with exit code 0
```

128 Threads (All Phases):

```
Total successes: 159820
Total failures: 0
Time elapsed (sec): 104.286
Total throughput (reqs/sec): 1532.516349270276

Process finished with exit code 0
```

128 Threads (Phase 2 Only):

```
Total successes: 119808
Total failures: 0
Time elapsed (sec): 42.974
Total throughput (reqs/sec): 2787.918276166985

Process finished with exit code 0
```

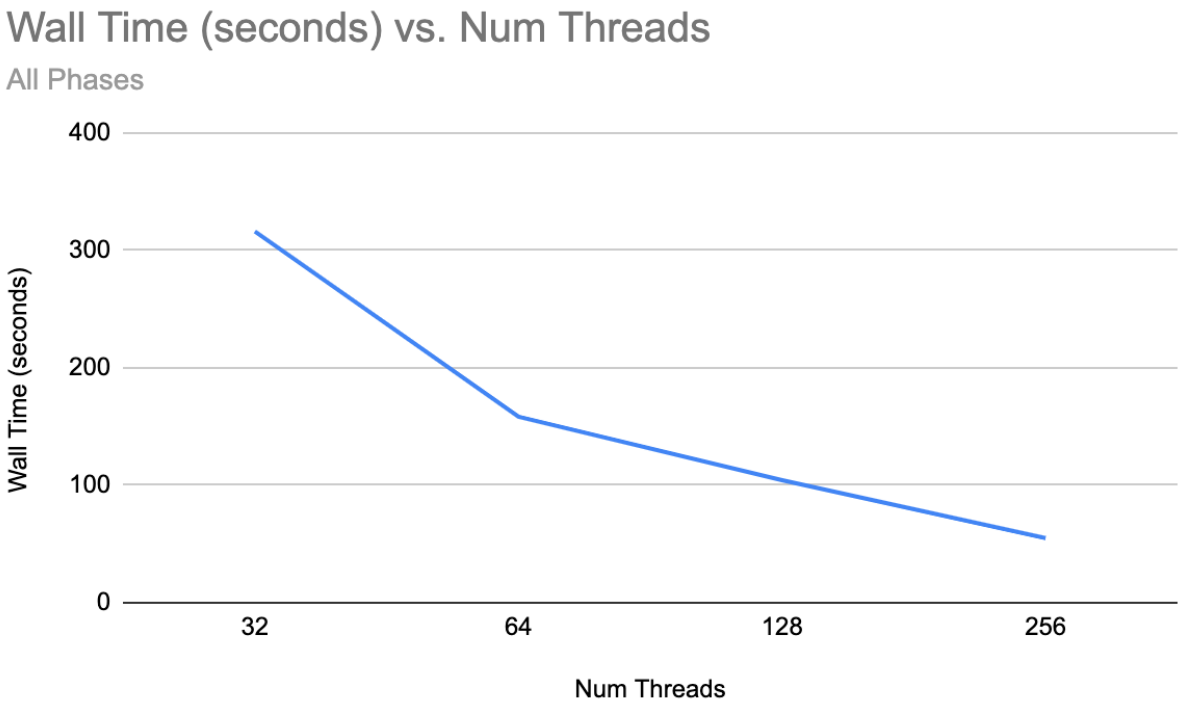
256 Threads (All Phases):

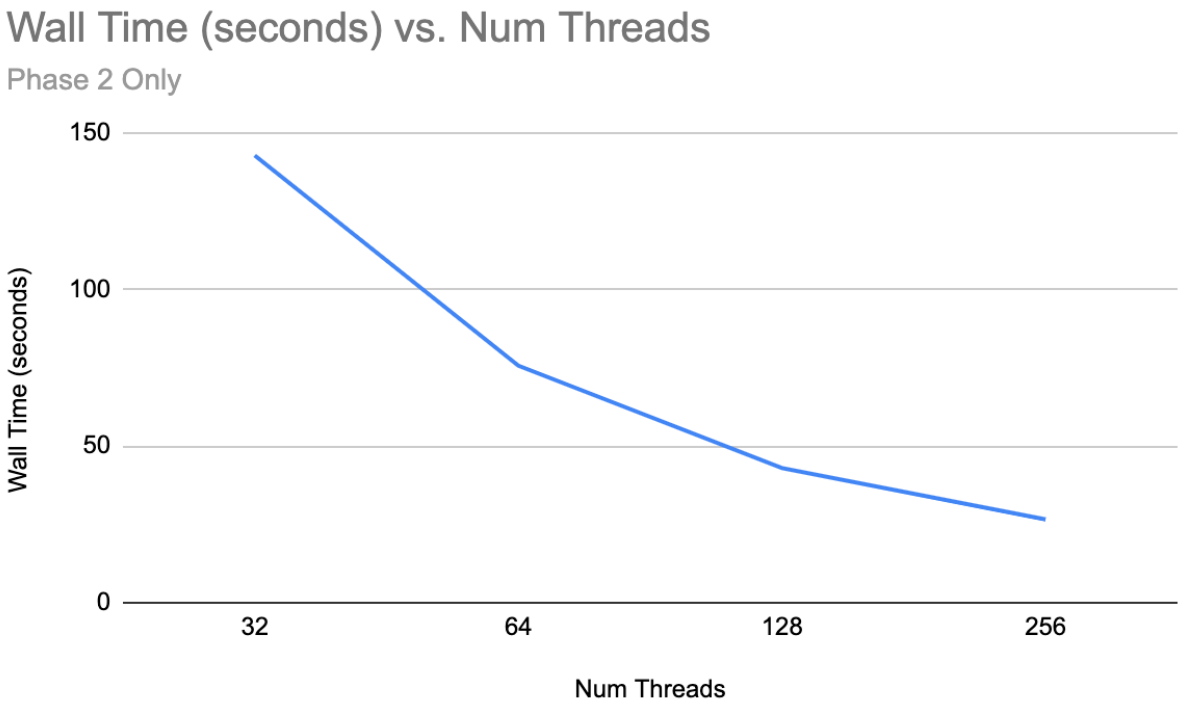
```
Total successes: 159769
Total failures: 0
Time elapsed (sec): 55.022
Total throughput (reqs/sec): 2903.7294173239798

Process finished with exit code 0
```

256 Threads (Phase 2 Only):

```
Total successes: 119808
Total failures: 0
Time elapsed (sec): 26.612
Total throughput (reqs/sec): 4502.029159777544

Process finished with exit code 0
```

Plots of Thread Count and Wall Time in All Phase and Phase 2 Only Runs:





Client Part 2 Results

As above, the differences in throughput between “all phase” and “phase 2 only” runs were significant, so both sets of data are provided in this section.

32 Threads (3 Phases):

```
Total successes: 160003
Total failures: 0
Time elapsed (sec): 314.9
Total throughput (reqs/sec): 508.10733

Max response time: 7951
Min response time: 19
Mean response time: 35.846626418679065
Median response time: 35
99th percentile response time: 63

Process finished with exit code 0
```

32 Threads (Phase 2 Only):

```
Total successes: 120000
Total failures: 0
Time elapsed (sec): 148.459
Total throughput (reqs/sec): 808.30396

Max response time: 2362
Min response time: 23
Mean response time: 39.300808333333336
Median response time: 37
99th percentile response time: 70

Process finished with exit code 0
```

64 Threads (3 Phases):

```
Total successes: 159814
Total failures: 0
Time elapsed (sec): 194.728
Total throughput (reqs/sec): 820.70374

Max response time: 10113
Min response time: 21
Mean response time: 46.97395123049783
Median response time: 43
99th percentile response time: 85

Process finished with exit code 0
```

64 Threads (Phase 2 Only):

```
Total successes: 119808
Total failures: 0
Time elapsed (sec): 75.709
Total throughput (reqs/sec): 1582.4803

Max response time: 300
Min response time: 19
Mean response time: 39.84193876869658
Median response time: 38
99th percentile response time: 72

Process finished with exit code 0
```

128 Threads (3 Phases):

```
Total successes: 159820
Total failures: 0
Time elapsed (sec): 118.643
Total throughput (reqs/sec): 1347.0664

Max response time: 10075
Min response time: 22
Mean response time: 53.303631188815565
Median response time: 49
99th percentile response time: 100

Process finished with exit code 0
```

128 Threads (Phase 2 Only):

```
Total successes: 119808
Total failures: 0
Time elapsed (sec): 46.44
Total throughput (reqs/sec): 2579.845

Max response time: 7985
Min response time: 22
Mean response time: 42.551358344113844
Median response time: 40
99th percentile response time: 75

Process finished with exit code 0
```

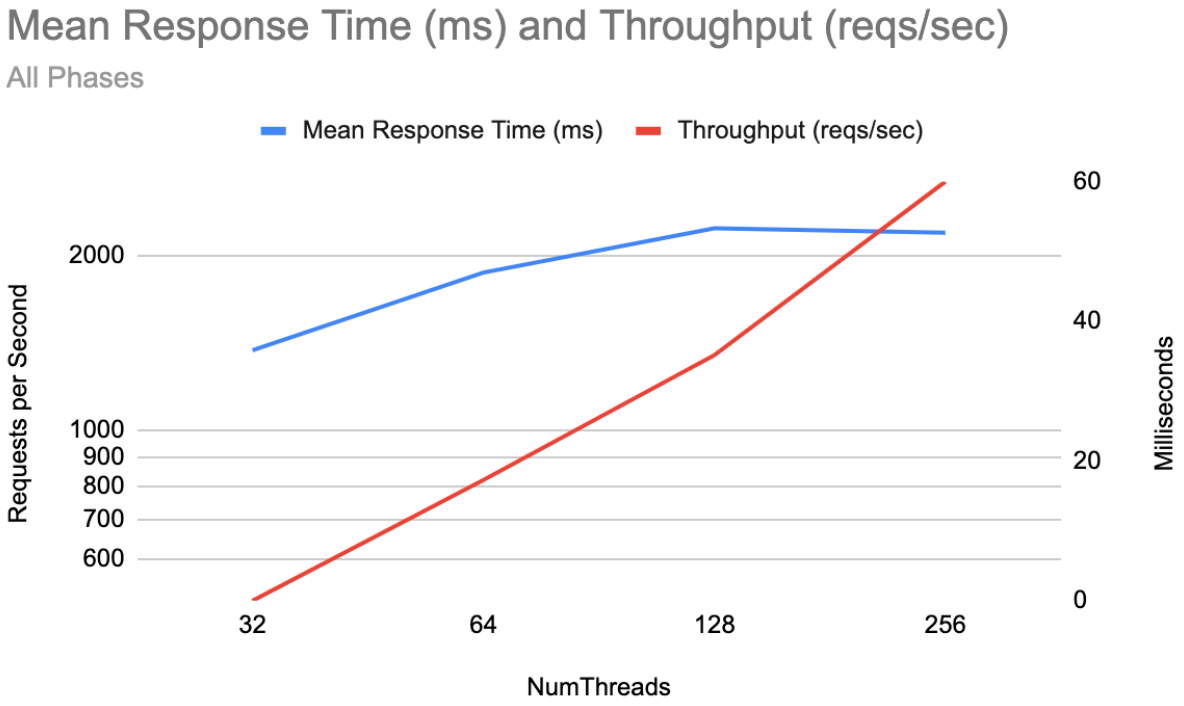
256 Threads (3 Phases):

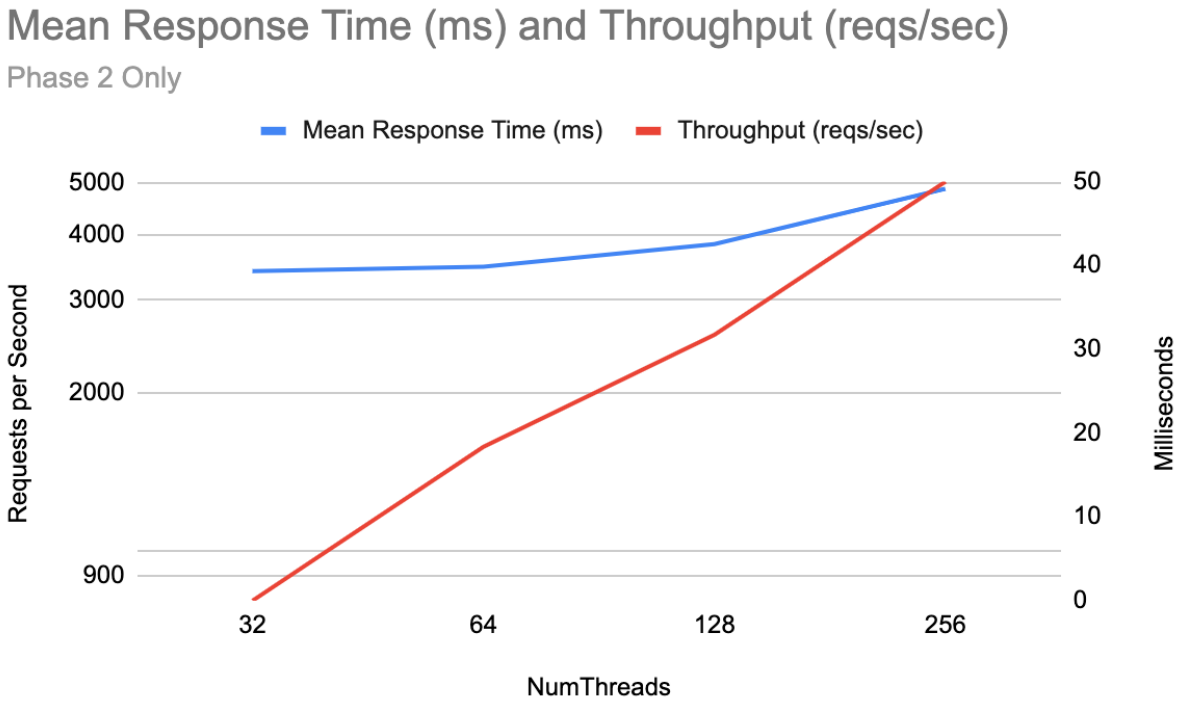
```
Total successes: 159769
Total failures: 0
Time elapsed (sec): 59.522
Total throughput (reqs/sec): 2684.201

Max response time: 10002
Min response time: 21
Mean response time: 52.68165260496207
Median response time: 50
99th percentile response time: 104

Process finished with exit code 0
```

256 Threads (Phase 2 Only):

```
Total successes: 119808
Total failures: 0
Time elapsed (sec): 23.782
Total throughput (reqs/sec): 5037.76

Max response time: 386
Min response time: 21
Mean response time: 49.14888822115385
Median response time: 47
99th percentile response time: 86
```

Plots of Thread Count, Mean Response Time, and Throughput for All Phase and Phase 2 Only Runs:





Bonus: Chart of Average Latencies Over Time Elapsed (256 Threads, All Phases)

Below is an aggregate chart that shows a 256-thread run with all phases active. The latencies for each request were recorded and the average latencies per second of the run are produced below. As we can see, the latencies during the early part of the run (Phase 1) are relatively low, and they increase and become more volatile as the peak phase (Phase 2) ramps up. The latencies re-normalize as Phase 2 ends and Phase 3 initiates and completes.