

# Assignment 1

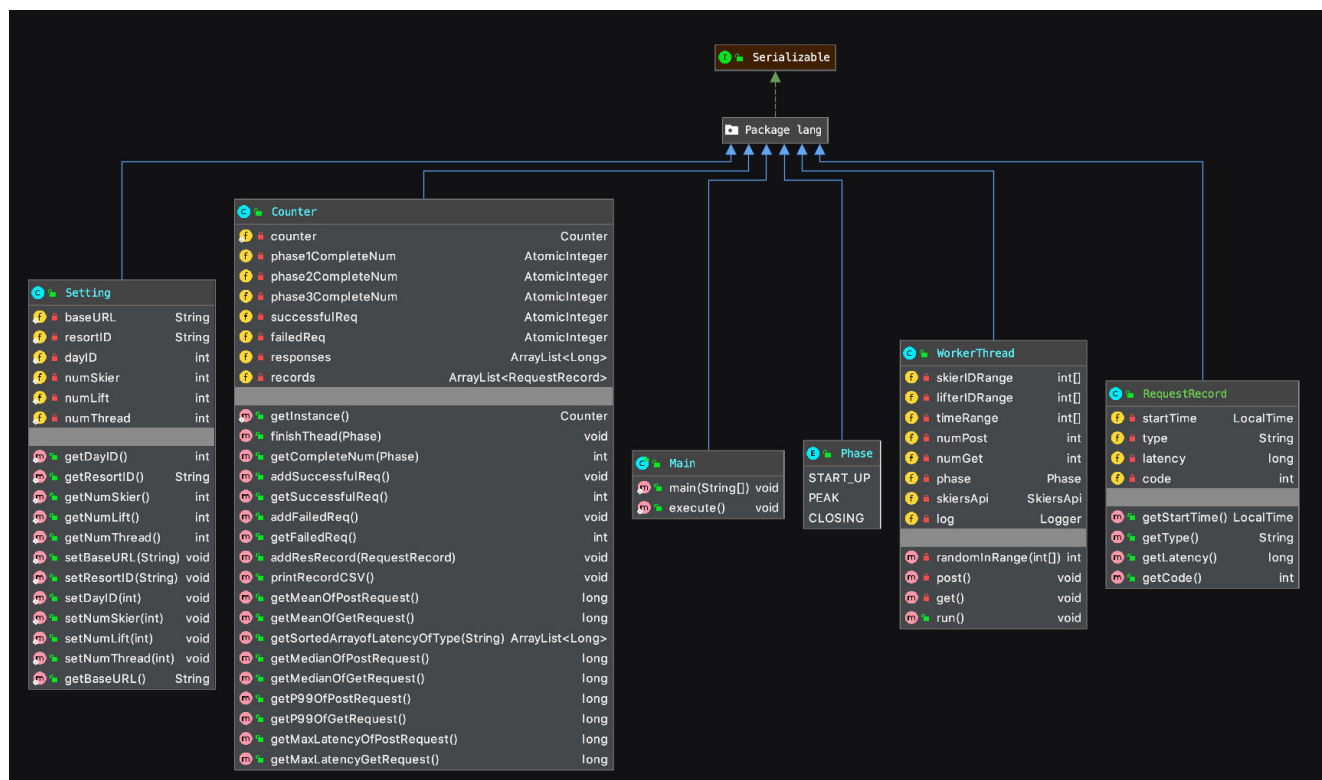
## Github Repo

[https://github.com/duskcloudxu/bsds2020fall\\_Assignment1](https://github.com/duskcloudxu/bsds2020fall_Assignment1)

## Server IP Address

`http://54.227.95.209:8080/remoteServer_war/`

## Project Structure



## Overview

- **Setting**: a **static class** stores all necessary parameters, like serverURL, resortID and dayID etc. Those parameters come with default value according to the assignment description, and would be referenced by other component. In this way, we reduce the structure complexity by avoiding passing same value from component to component.
- **Phase**: **Enum** to represent 3 different phase for tagging threads of different phases.
- **Main**: **Entrance** of program, parse parameters and update those parameters into **Setting**,

create and start threads of three phases. With support of *apache cli* library, it supports standard style of commendline input. (e.g. `-T 256 -S 20000 -L 40`).

- `WorkerThread`: **Subclass** of `Thread`, execute requests to the server and monitor status of each requests. All the failed requests would be logged and reported to `Counter`.
- `Counter`: **a thread-safe class** under *single instance* design. It manage the state of threads, requests and request records (e.g. number of failed requests, number of start-up phase threads finished) . Also it would calculate the metrics in the end of the program and return corresponding data to `Main`.
- `RequestRecord`: **a model class** to store necessary information for a request record(start time, type(post or get), latency and returned status code), for convenience of printing csv.

## Client Part 1 running Screenshot and plot

numThread\metric	Successful Request	Failed Request	Wall Time(Sec)	Throughput(per Sec)
32	5080	0	21	241
64	10160	0	34	298
128	20320	0	33	615
256	40541	99	45	903

32 threads

```
Successful Request:
5080
Failed Request:
0
Wall Time:
21
Throughput:
241

Process finished with exit code 0
```

64 threads

```
Successful Request:
10160
Failed Request:
0
Wall Time:
34
Throughput:
298

Process finished with exit code 0
```

128 threads

```
Successful Request:
20320
Failed Request:
0
Wall Time:
33
Throughput:
615

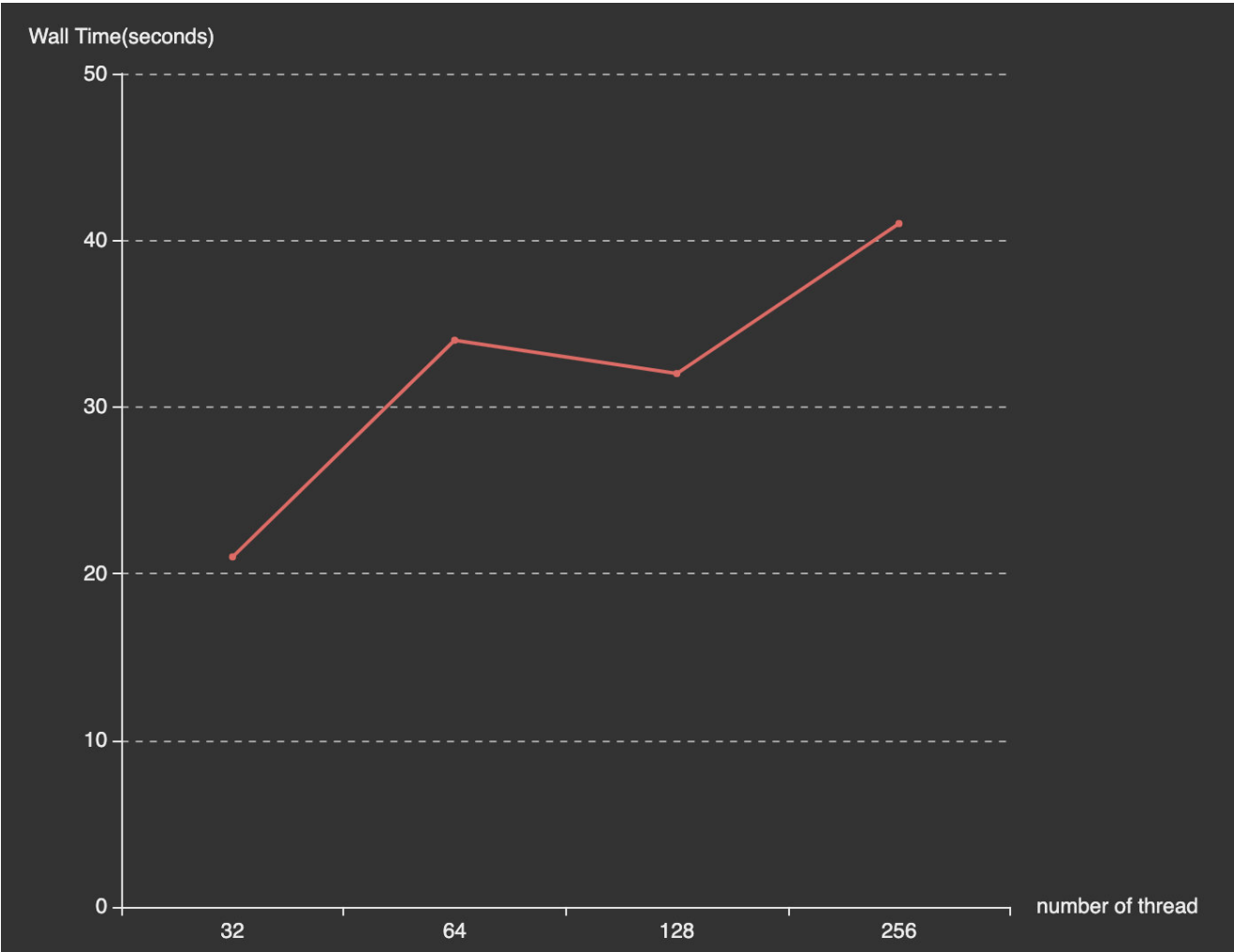
Process finished with exit code 0
```

256 threads

```
Successful Request:
40541
Failed Request:
99
Wall Time:
45
Throughput:
903

Process finished with exit code 0
```

# Plot

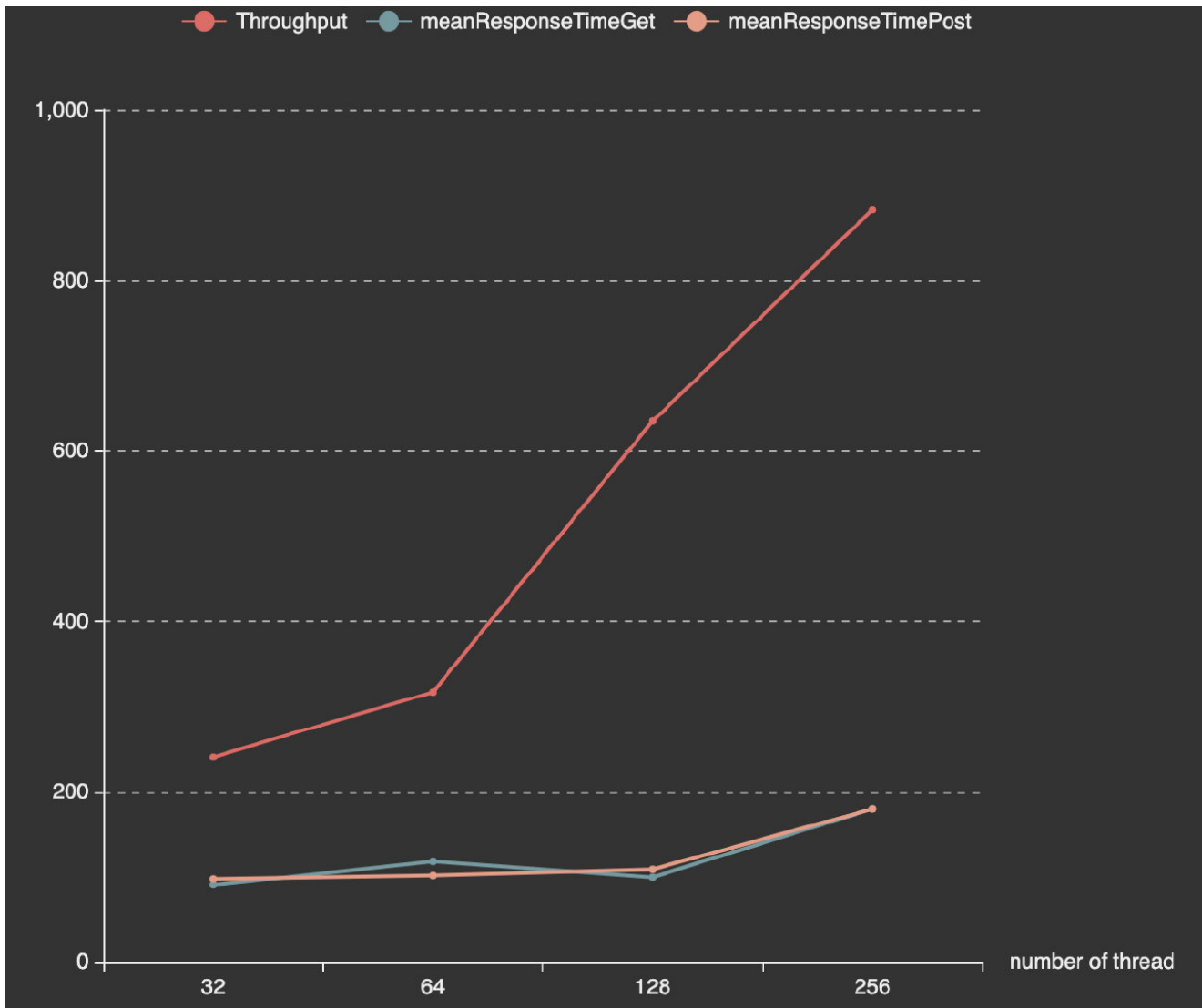


## Client Part 2

numThread\metric	Mean Post Latency	Mean Get Latency	Median Post Latency	Median Get Latency	Wall Time(Sec)	Throughput(per Sec)	P99 of Post	P99 of Get	Max Response of Post	Max Response of Get
32	98	91	92	91	21	241	205	197	657	204
64	102	118	93	91	32	317	527	994	1021	1059
128	109	100	93	92	32	635	640	326	4382	1503
256	180	148	99	96	46	883	2131	1083	11573	10783

32 threads	64 threads	128 threads	256 threads
<pre> mean latency for POST 98ms mean latency for GET 91ms median latency for POST 92ms median latency for GET 91ms Wall Time: 21 Throughput: 241 P99 for POST 205ms P99 for GET 197ms Max Latency for POST 657ms Max Latency for GET 204ms  Process finished with exit code 0 </pre>	<pre> mean latency for POST 102ms mean latency for GET 118ms median latency for POST 93ms median latency for GET 91ms Wall Time: 32 Throughput: 317 P99 for POST 527ms P99 for GET 994ms Max Latency for POST 1021ms Max Latency for GET 1059ms  Process finished with exit code 0 </pre>	<pre> mean latency for POST 109ms mean latency for GET 100ms median latency for POST 93ms median latency for GET 92ms Wall Time: 32 Throughput: 635 P99 for POST 640ms P99 for GET 326ms Max Latency for POST 4382ms Max Latency for GET 1503ms  Process finished with exit code 0 </pre>	<pre> mean latency for POST 180ms mean latency for GET 148ms median latency for POST 99ms median latency for GET 96ms Wall Time: 46 Throughput: 883 P99 for POST 2131ms P99 for GET 1083ms Max Latency for POST 11573ms Max Latency for GET 10783ms </pre>

Plot of throughput, meanResponseTimeGet and mean ResponseTimePost against number of threads



- A CSV file named "record\_output.csv" would be generated at the root directory, which contains the information of every request, namely start time, request type, latency and reponse model

```

startTime,requestType,latency,responseCode
16:30:56.501,POST,443,200
16:30:56.502,POST,442,200
16:30:56.498,POST,466,200
16:30:56.503,POST,480,200
16:30:56.498,POST,485,200
16:30:56.501,POST,482,200
16:30:56.500,POST,483,200
16:30:56.502,POST,484,200
16:30:56.498,POST,489,200
16:30:56.503,POST,487,200
16:30:56.499,POST,491,200
○ 16:30:56.501,POST,506,200
16:30:56.498,POST,511,200
16:30:56.499,POST,511,200
16:30:56.498,POST,515,200
16:30:56.502,POST,512,200
16:30:56.503,POST,512,200
16:30:56.504,POST,513,200
16:30:56.501,POST,516,200
16:30:56.504,POST,515,200
16:30:56.499,POST,521,200
16:30:56.502,POST,520,200
16:30:56.502,POST,520,200
16:30:56.502,POST,522,200

```

## Break Things

- result interfere by network traffic (*run client locally and server remotely*)

when numThread comes to 256 or above, it would be heavily interfered by network traffic.

Number of failed Request varies depending on when you run it. For example, result of midnight is usually better than result of afternoon. It still holds even if you changed the maxthread in your tomcat server

- Break server with 1000+ threads

tried to run the client with `maxThread=1024`, and caused server crashed. All the queries to the server return time out error, and cannot use ssh to connect and restart the server. Finally figured out this problem by completely shutting down and restart the EC2 instance.