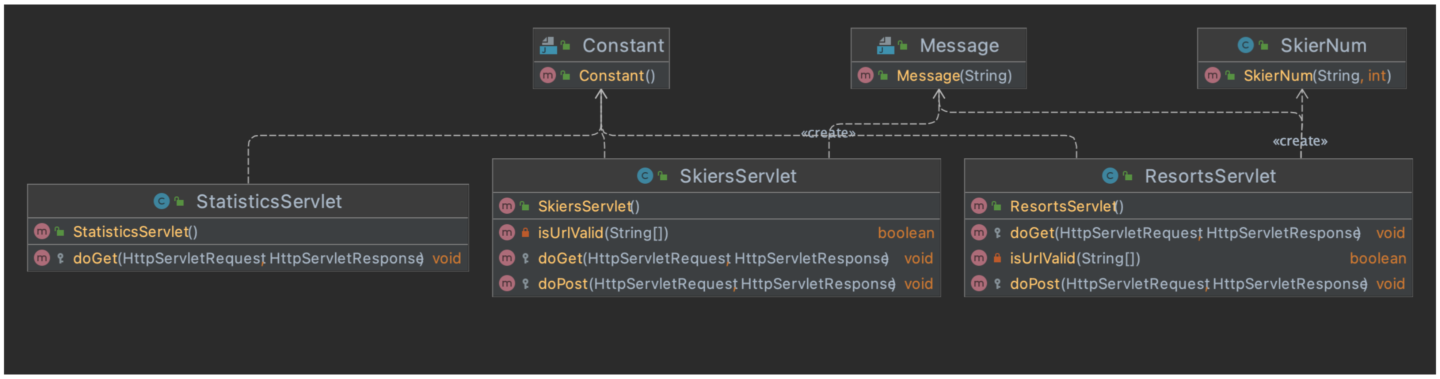
Report for Homework1 of CS6650

GitHub Repo

https://github.com/zjdx1998/CS6650/tree/Homework1/Assignment1

Server

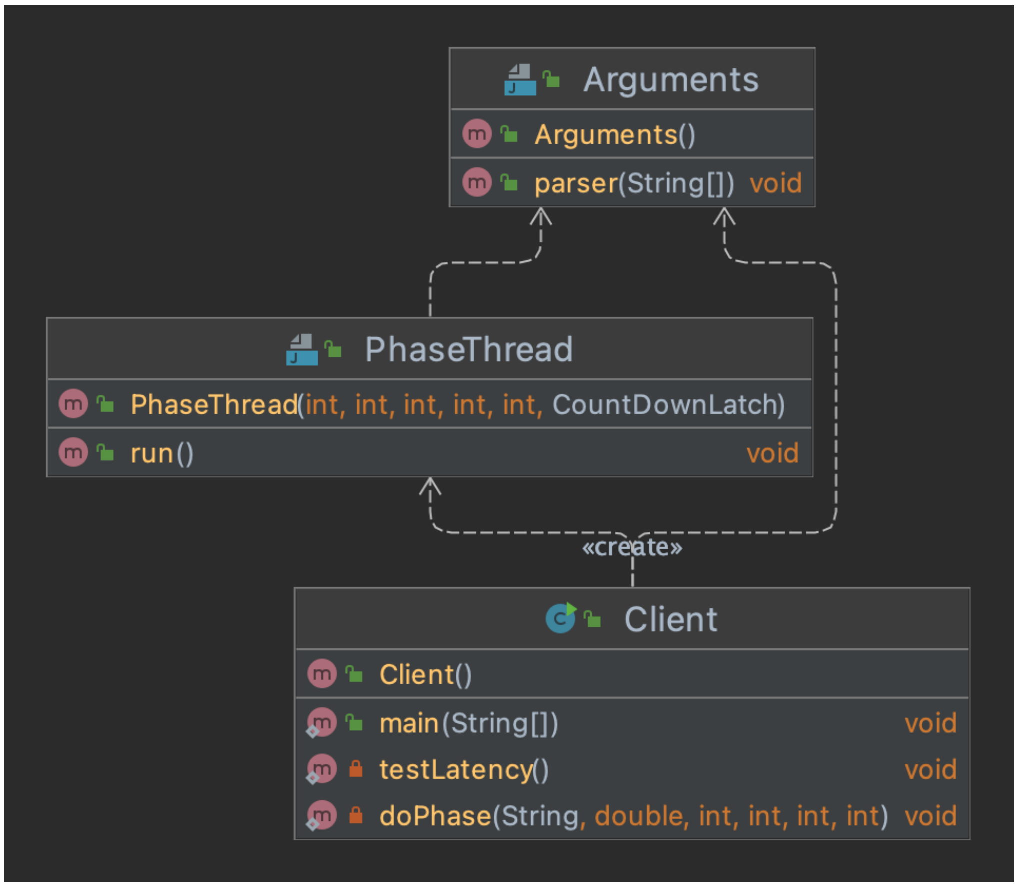
I implemented a full-completed server which can handle every request and return response as https://app.swaggerhub.com/apis/cloud-perf/SkiDataAPI/1.16#/skiers/getSkierResortTotals expected.



Constant
Constant()
Message
Message(String)
SkierNum
SkierNum(String, int)
«create»
«create»
StatisticsServlet
StatisticsServlet()
doGet(HttpServletRequest, HttpServletResponse) void
SkiersServlet
SkiersServlet()
isUrlValid(String[]) boolean
doGet(HttpServletRequest, HttpServletResponse) void
doPost(HttpServletRequest, HttpServletResponse) void
ResortsServlet
ResortsServlet()
doGet(HttpServletRequest, HttpServletResponse) void
isUrlValid(String[]) boolean
doPost(HttpServletRequest, HttpServletResponse) void

Client Overview

Design

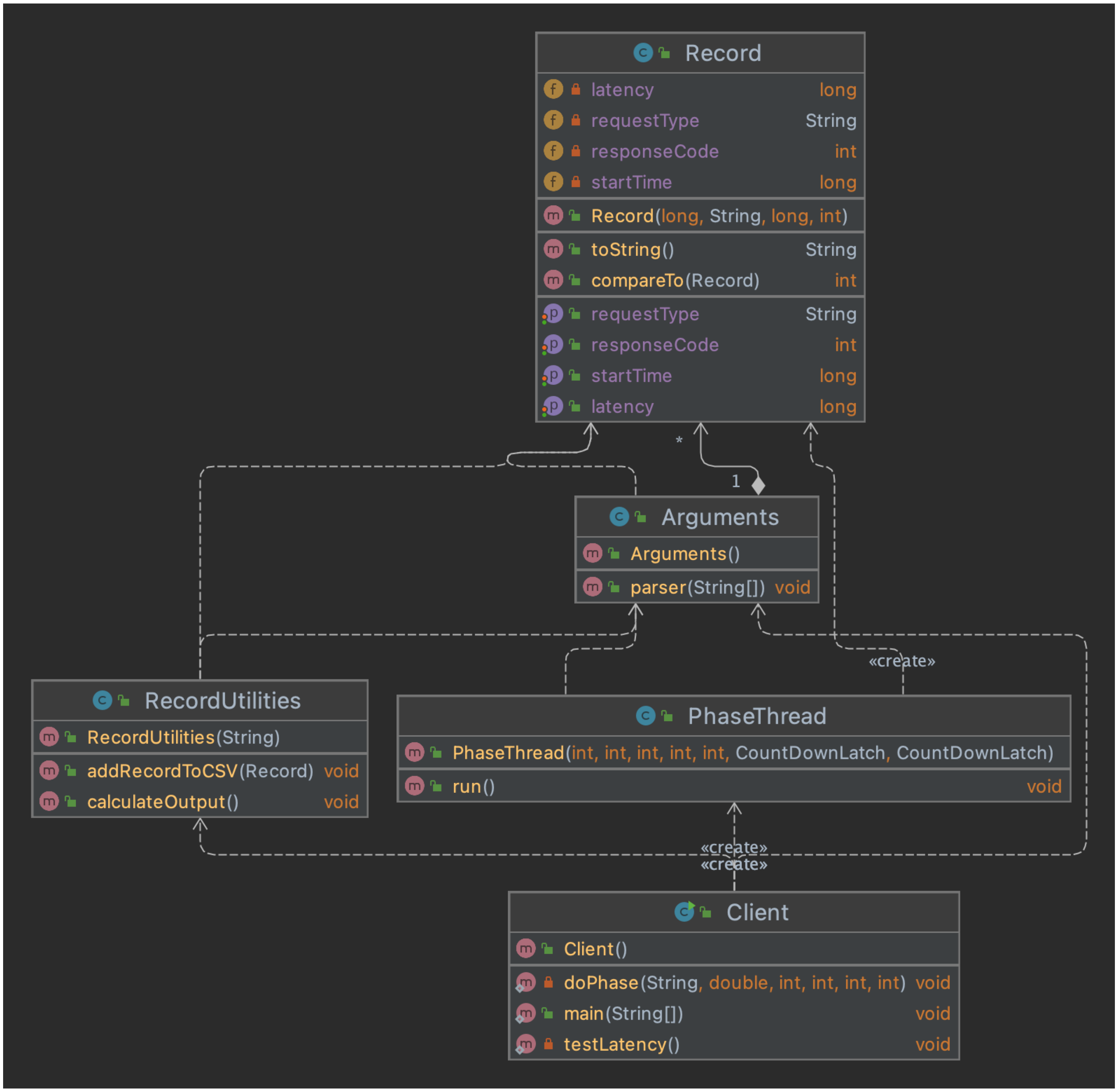


Client 1 is consist of three parts:

The first part is global arguments processing, where I utilized `args4j` to parse the arguments. I also stored the global variables like `successCount`, `failureCount` which represents the total number of successful and failed requests.

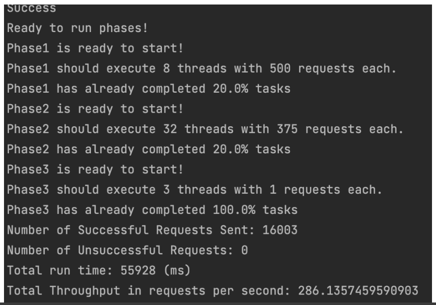
The second part is a runnable class `PhaseThread`, which accepts `int startID, int endID, int startTime, int endTime, int numOfReqs, CountDownLatch latch` as parameter, this is the per client request model.

The third part is main class `client` which has a `static void main` function. This class firstly load the arguments from `Argument` then create a thread pool with total numbers of three phases. All the staticstics data are calculated here.

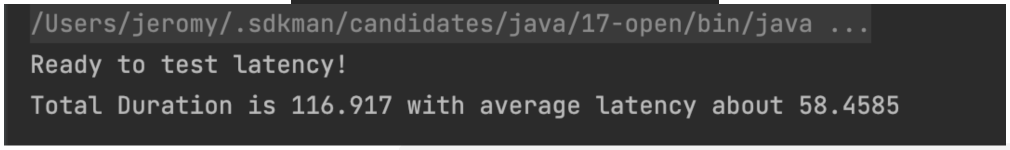


In addition to Client 1, Client 2 also contains a `csvUtiliizer` class to utilize the data.

Little's Law Verification



```
Success
Ready to run phases!
Phase1 is ready to start!
Phase1 should execute 8 threads with 500 requests each.
Phase1 has already completed 20.0% tasks
Phase2 is ready to start!
Phase2 should execute 32 threads with 375 requests each.
Phase2 has already completed 20.0% tasks
Phase3 is ready to start!
Phase3 should execute 3 threads with 1 requests each.
Phase3 has already completed 100.0% tasks
Number of Successful Requests Sent: 16003
Number of Unsuccessful Requests: 0
Total run time: 55928 (ms)
Total Throughput in requests per second: 286.1357459590903
```



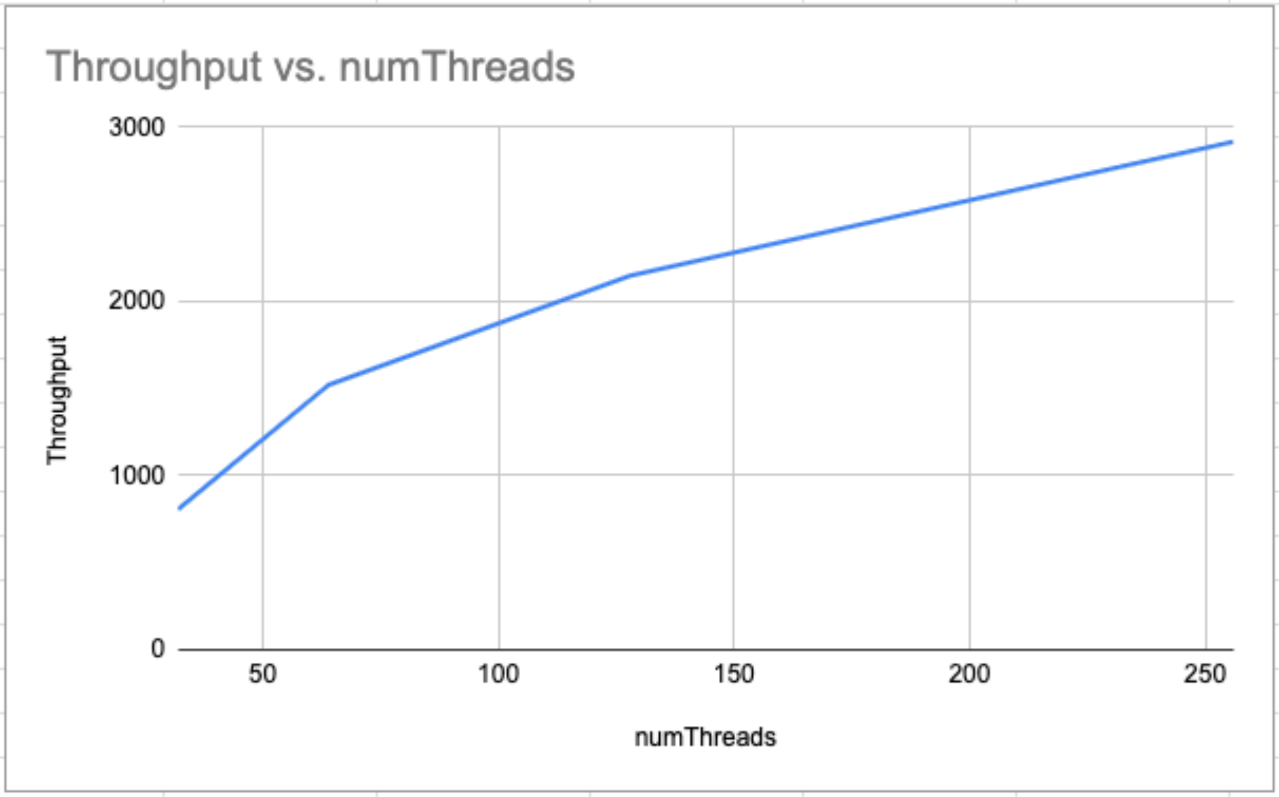
```
/Users/jeromy/.sdkman/candidates/java/17-open/bin/java ...
Ready to test latency!
Total Duration is 116.917 with average latency about 58.4585
```

The above results are generated by `-nt 32 -ns 2000 -nl 40 -nr 10 -test-only -server ec2-54-149-212-65.us-west-2.compute.amazonaws.com` arguments with `-test-only` enabled or disabled. The test only data is calculated by test same number of threads

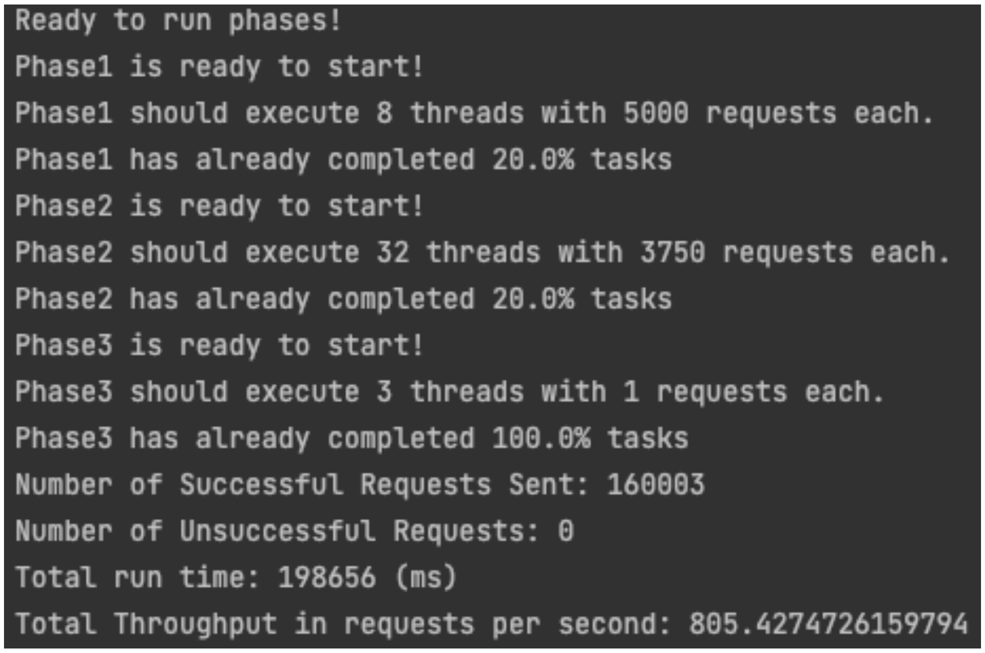
We can see the predicted throughput for test-only is $\frac{32}{58.4585} \approx \frac{16003}{55928} = 0.286$, especially consider three phases of clients using different number of threads.

Client 1 Statistics

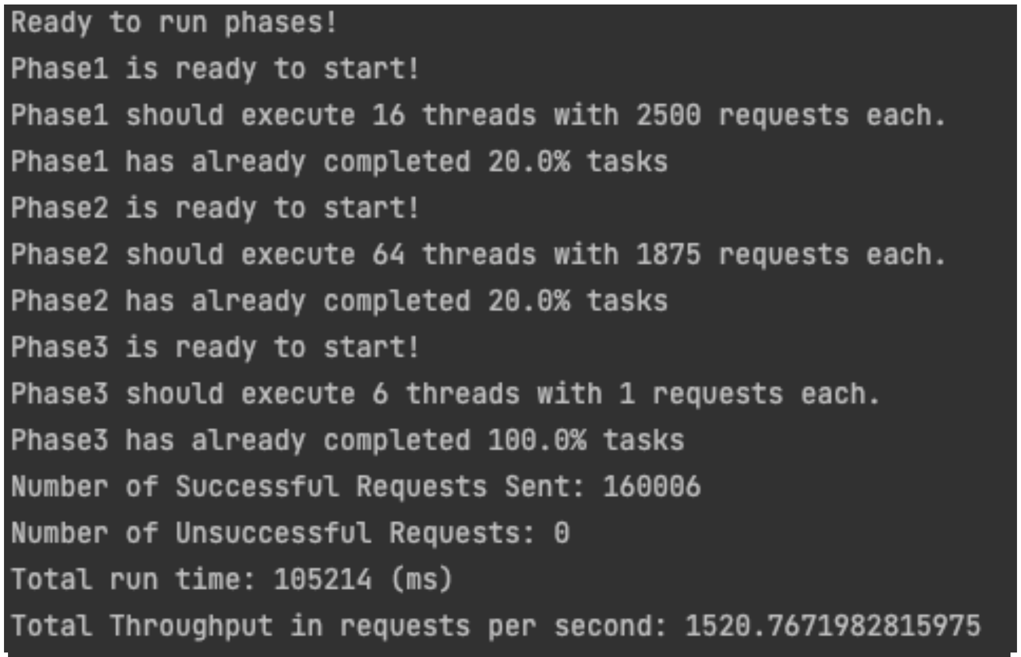
numThreads	32	64	128	256
Throughput	805.4274	1520.7671	2147.7313	2916.9799



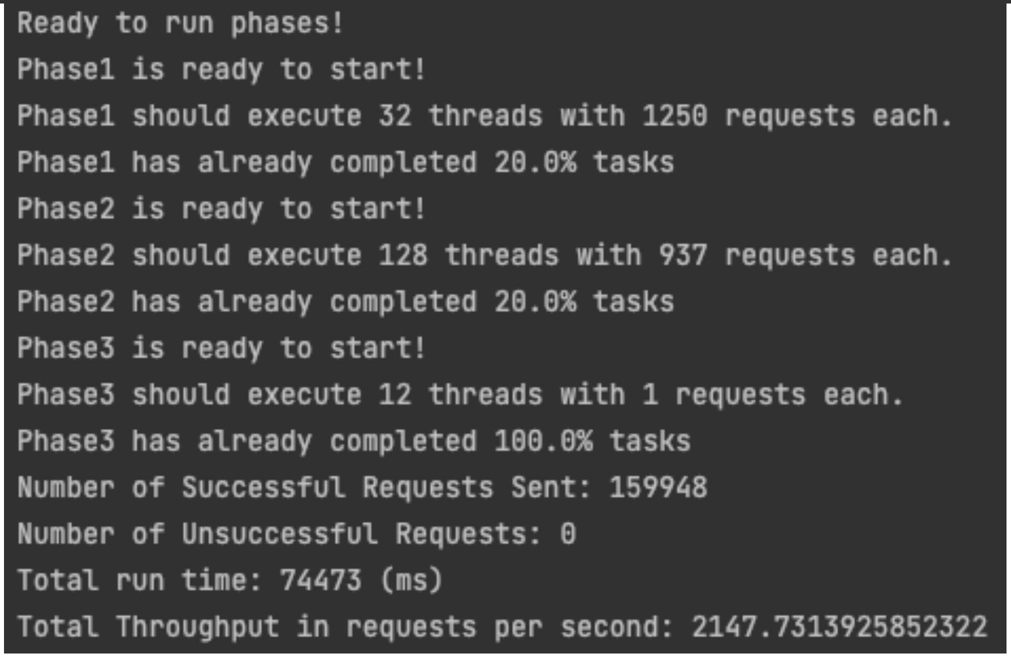
We can see the throughput is linear related to the num of Threads.



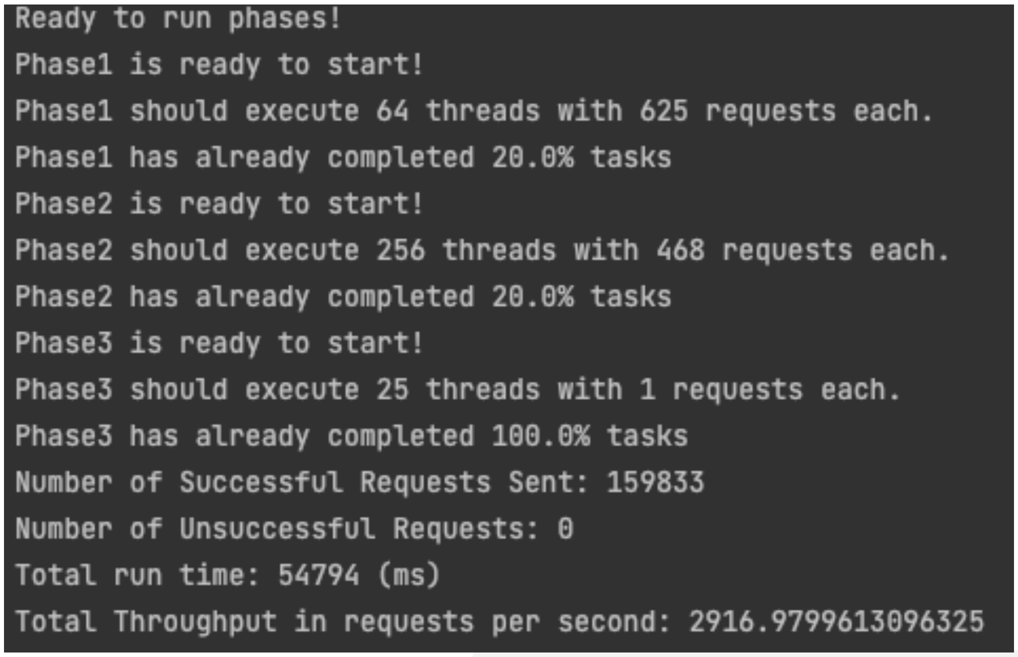
```
Ready to run phases!
Phase1 is ready to start!
Phase1 should execute 8 threads with 5000 requests each.
Phase1 has already completed 20.0% tasks
Phase2 is ready to start!
Phase2 should execute 32 threads with 3750 requests each.
Phase2 has already completed 20.0% tasks
Phase3 is ready to start!
Phase3 should execute 3 threads with 1 requests each.
Phase3 has already completed 100.0% tasks
Number of Successful Requests Sent: 160003
Number of Unsuccessful Requests: 0
Total run time: 198656 (ms)
Total Throughput in requests per second: 805.4274726159794
```



```
Ready to run phases!
Phase1 is ready to start!
Phase1 should execute 16 threads with 2500 requests each.
Phase1 has already completed 20.0% tasks
Phase2 is ready to start!
Phase2 should execute 64 threads with 1875 requests each.
Phase2 has already completed 20.0% tasks
Phase3 is ready to start!
Phase3 should execute 6 threads with 1 requests each.
Phase3 has already completed 100.0% tasks
Number of Successful Requests Sent: 160006
Number of Unsuccessful Requests: 0
Total run time: 105214 (ms)
Total Throughput in requests per second: 1520.7671982815975
```



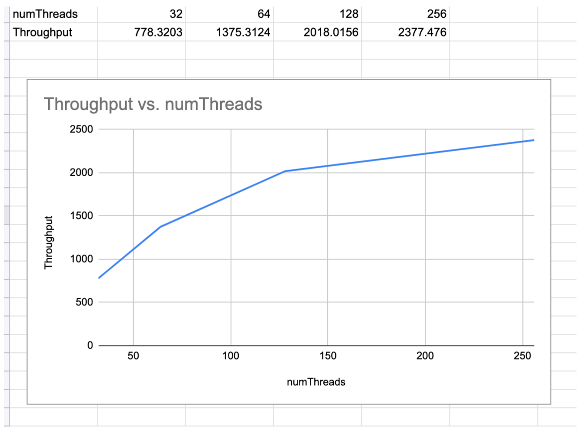
```
Ready to run phases!
Phase1 is ready to start!
Phase1 should execute 32 threads with 1250 requests each.
Phase1 has already completed 20.0% tasks
Phase2 is ready to start!
Phase2 should execute 128 threads with 937 requests each.
Phase2 has already completed 20.0% tasks
Phase3 is ready to start!
Phase3 should execute 12 threads with 1 requests each.
Phase3 has already completed 100.0% tasks
Number of Successful Requests Sent: 159948
Number of Unsuccessful Requests: 0
Total run time: 74473 (ms)
Total Throughput in requests per second: 2147.7313925852322
```



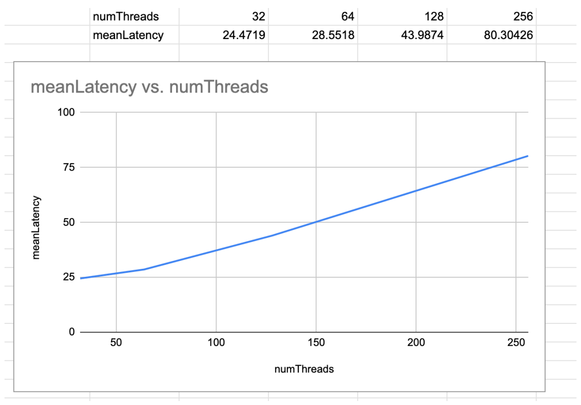
```
Ready to run phases!
Phase1 is ready to start!
Phase1 should execute 64 threads with 625 requests each.
Phase1 has already completed 20.0% tasks
Phase2 is ready to start!
Phase2 should execute 256 threads with 468 requests each.
Phase2 has already completed 20.0% tasks
Phase3 is ready to start!
Phase3 should execute 25 threads with 1 requests each.
Phase3 has already completed 100.0% tasks
Number of Successful Requests Sent: 159833
Number of Unsuccessful Requests: 0
Total run time: 54794 (ms)
Total Throughput in requests per second: 2916.9799613096325
```

The above charts and results are did by `-nt 32 -ns 2000 -nl 40 -nr 10 -server ec2-54-149-212-65.us-west-2.compute.amazonaws.com` with `nt` changes to `32/64/128/256` respectively.

Clients 2 Statistics

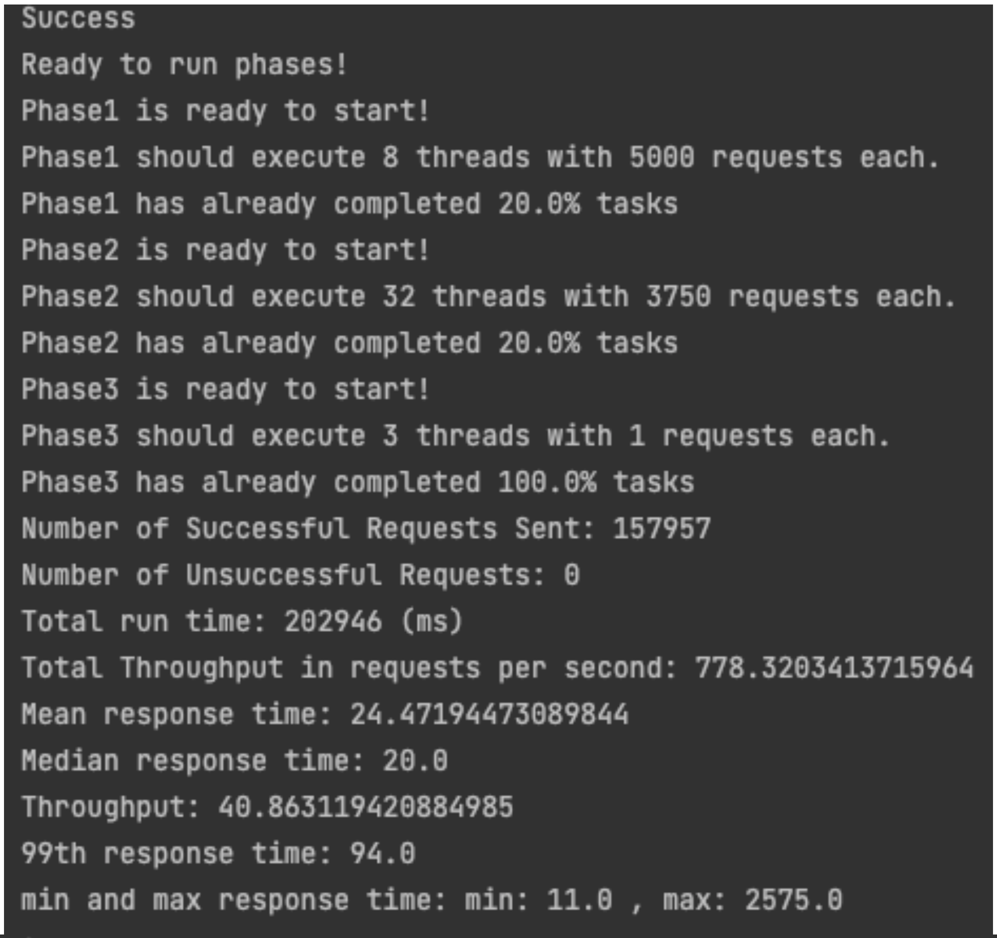


numThreads	32	64	128	256
Throughput	778.3203	1375.3124	2018.0156	2377.476

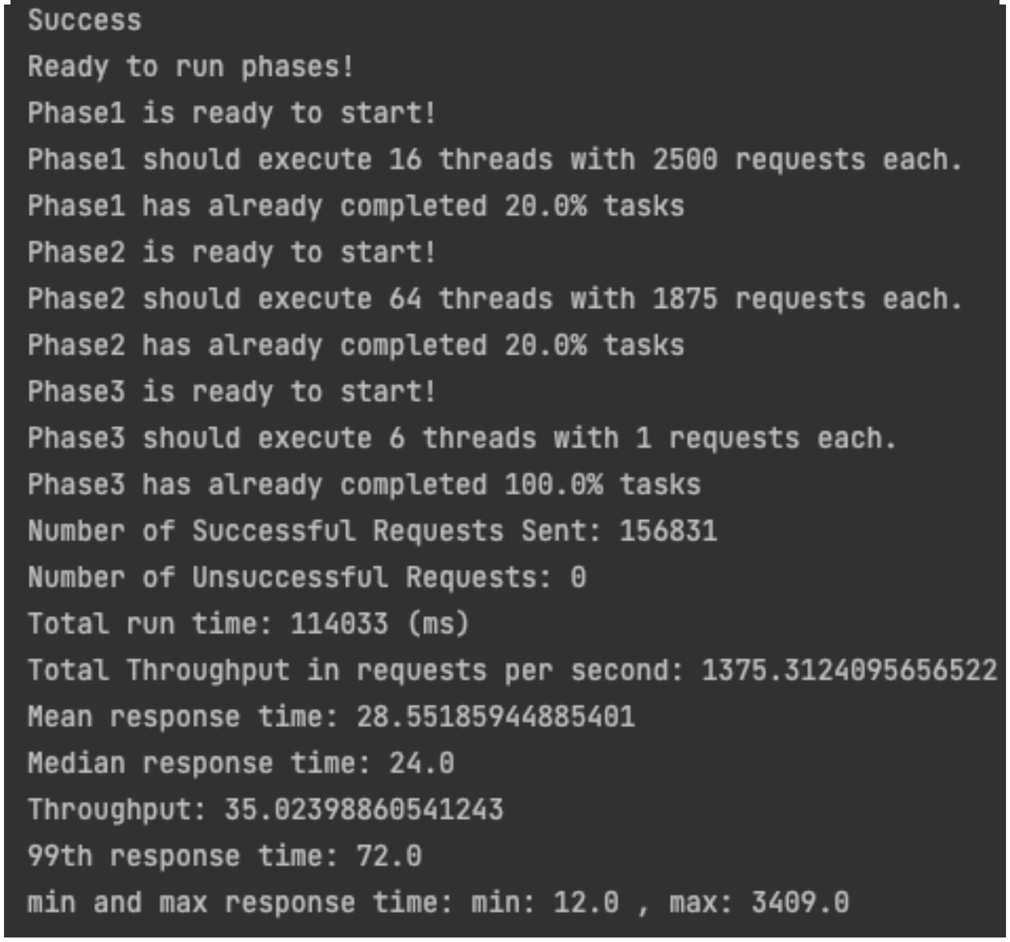


numThreads	32	64	128	256
meanLatency	24.4719	28.5518	43.9874	80.30426

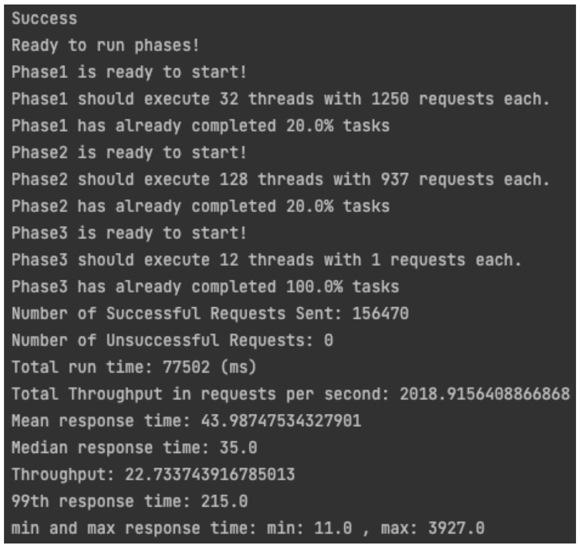
We can see both Throughput and mean Latency are linearly related to the number of threads. Consider the default maximum restrictions of Tomcat in AWS EC2 is 200, the first chart contains a inflection point which represents this case.



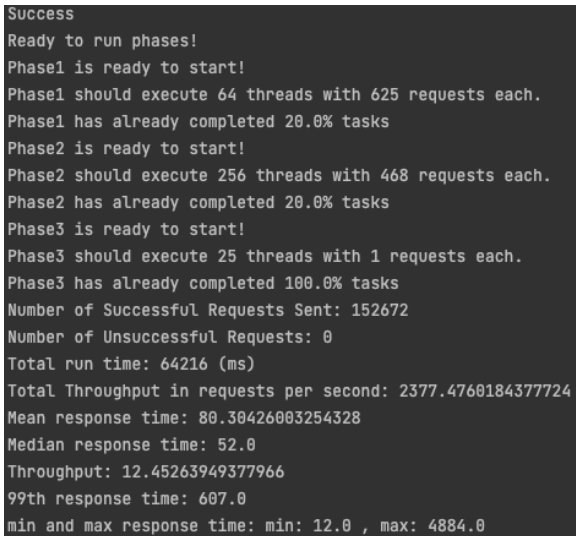
```
Success
Ready to run phases!
Phase1 is ready to start!
Phase1 should execute 8 threads with 5000 requests each.
Phase1 has already completed 20.0% tasks
Phase2 is ready to start!
Phase2 should execute 32 threads with 3750 requests each.
Phase2 has already completed 20.0% tasks
Phase3 is ready to start!
Phase3 should execute 3 threads with 1 requests each.
Phase3 has already completed 100.0% tasks
Number of Successful Requests Sent: 157957
Number of Unsuccessful Requests: 0
Total run time: 202946 (ms)
Total Throughput in requests per second: 778.3203413715964
Mean response time: 24.47194473089844
Median response time: 20.0
Throughput: 40.863119420884985
99th response time: 94.0
min and max response time: min: 11.0 , max: 2575.0
```



```
Success
Ready to run phases!
Phase1 is ready to start!
Phase1 should execute 16 threads with 2500 requests each.
Phase1 has already completed 20.0% tasks
Phase2 is ready to start!
Phase2 should execute 64 threads with 1875 requests each.
Phase2 has already completed 20.0% tasks
Phase3 is ready to start!
Phase3 should execute 6 threads with 1 requests each.
Phase3 has already completed 100.0% tasks
Number of Successful Requests Sent: 156831
Number of Unsuccessful Requests: 0
Total run time: 114033 (ms)
Total Throughput in requests per second: 1375.3124095656522
Mean response time: 28.55185944885401
Median response time: 24.0
Throughput: 35.02398860541243
99th response time: 72.0
min and max response time: min: 12.0 , max: 3409.0
```

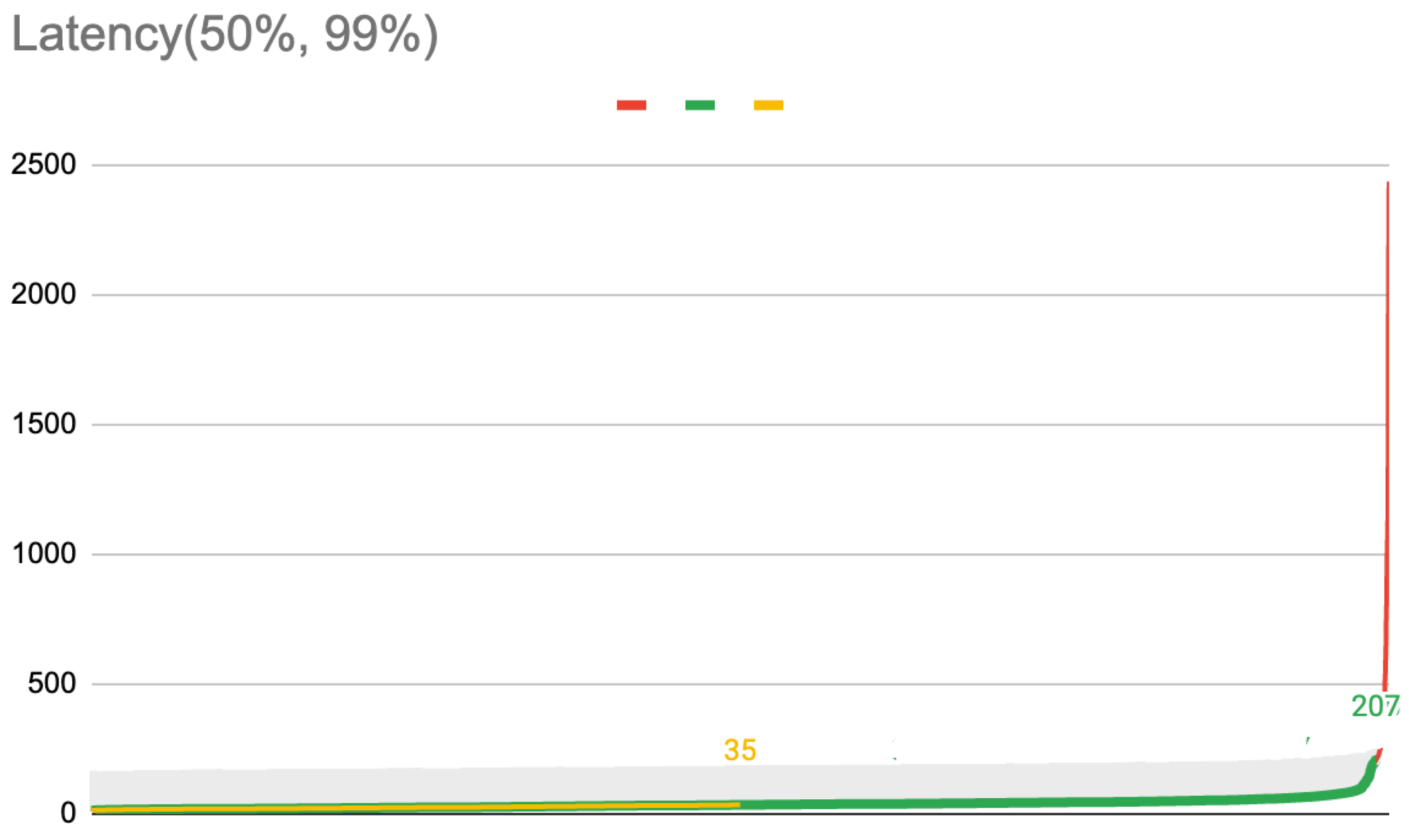


```
Success
Ready to run phases!
Phase1 is ready to start!
Phase1 should execute 32 threads with 1250 requests each.
Phase1 has already completed 20.0% tasks
Phase2 is ready to start!
Phase2 should execute 128 threads with 937 requests each.
Phase2 has already completed 20.0% tasks
Phase3 is ready to start!
Phase3 should execute 12 threads with 1 requests each.
Phase3 has already completed 100.0% tasks
Number of Successful Requests Sent: 156470
Number of Unsuccessful Requests: 0
Total run time: 77502 (ms)
Total Throughput in requests per second: 2018.9156408866868
Mean response time: 43.98747534327901
Median response time: 35.0
Throughput: 22.733743916785013
99th response time: 215.0
min and max response time: min: 11.0 , max: 3927.0
```

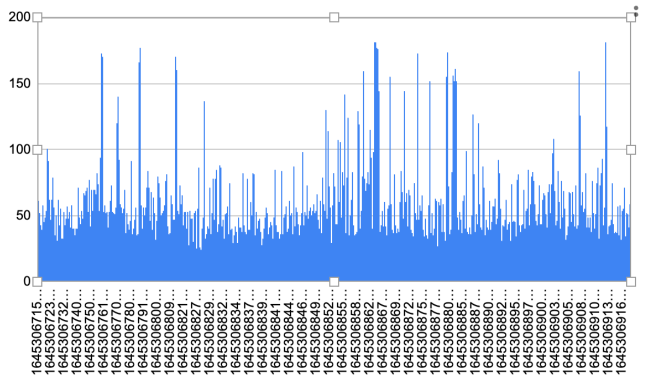


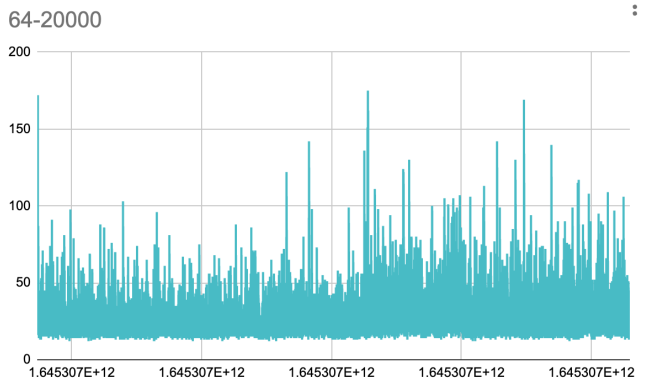
```
Success
Ready to run phases!
Phase1 is ready to start!
Phase1 should execute 64 threads with 625 requests each.
Phase1 has already completed 20.0% tasks
Phase2 is ready to start!
Phase2 should execute 256 threads with 468 requests each.
Phase2 has already completed 20.0% tasks
Phase3 is ready to start!
Phase3 should execute 25 threads with 1 requests each.
Phase3 has already completed 100.0% tasks
Number of Successful Requests Sent: 152672
Number of Unsuccessful Requests: 0
Total run time: 64216 (ms)
Total Throughput in requests per second: 2377.4760184377724
Mean response time: 80.30426003254328
Median response time: 52.0
Throughput: 12.45263949377966
99th response time: 607.0
min and max response time: min: 12.0 , max: 4884.0
```

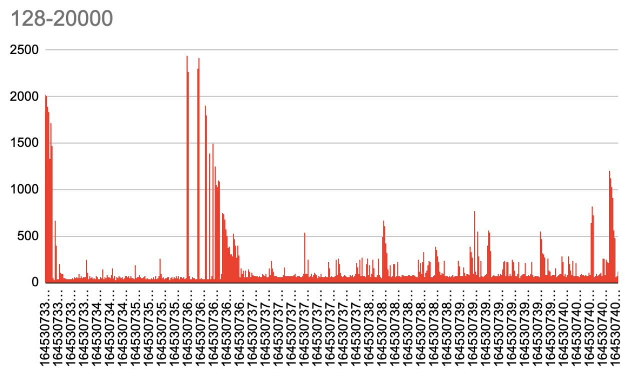
Bonus

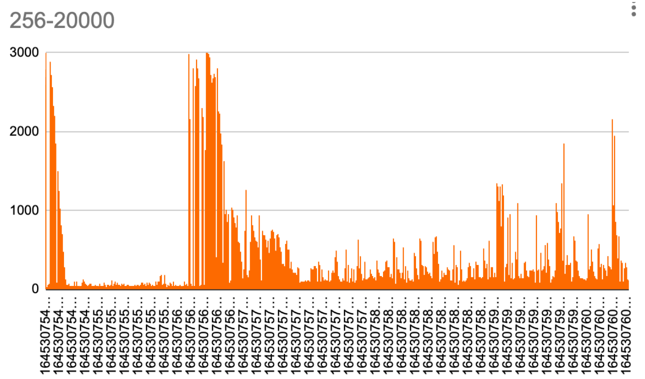


The above chart is ploted for `numThreads=128, numSkiers=20000`, we can clearly see the p50 and p99, but it's not easy to figure out how it's distributed.









Now we know the latency distribution is very interesting which matches well with the actual request layout. This is exactly ploted by average response time for certain time interval bucket.

All the raw data can be found at **Clients/res/records**.