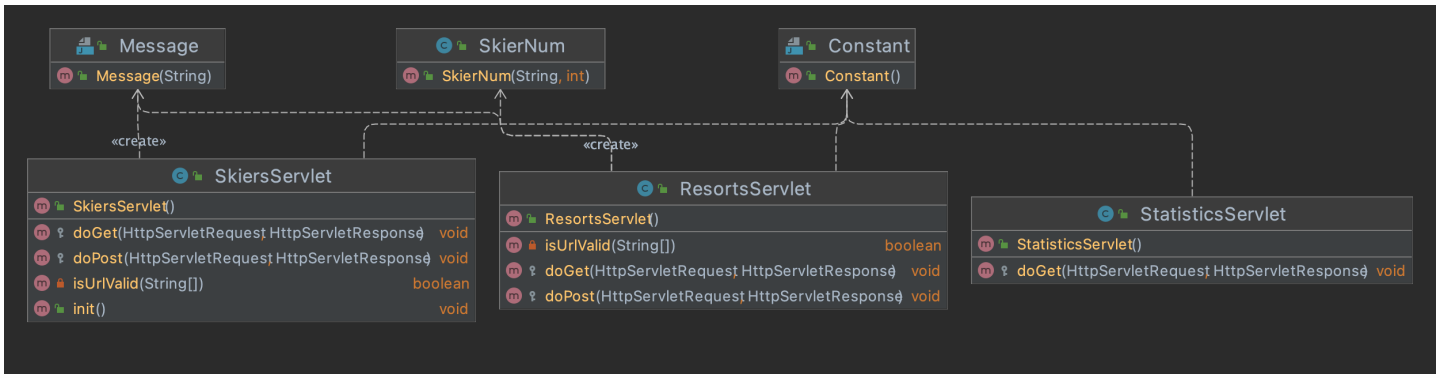
CS6650 Assignment2 Report

URL to Repo

Please visit Github Repository.

Design

Server



I adopts the server desgin from previous project, except adding a `BlockingQueue` to store the channels. Each time a channel is taken from the queue, a new channel will be added in to the queue.

Since we only test the `POST /skiers/{resortID}/seasons/{seasonID}/days/{dayID}/skiers/{skierID}` from https://app.swaggerhub.com/apis/cloud-perf/SkiDataAPI/1.16#/ , the main changes happen in `SkiersServlet` class.

In this class, I firstly use `ConnectionFactory` to build a connection with my rabbitmq which is located at `ec2-54-203-208-182.us-west-2.compute.amazonaws.com` . Since I have the private ipv4 address of this ec2 instance, I configured all needed information in `rabbitmy.conf` inside resource folder like this:

```
172.31.5.13
5672
admin
fuckjava
```

Once a skier post is received by Server, the configuration file will be automatically loaded and connect to Rabbitmq. Each post will be handled in `doPost` function. The primary function is listed below.

```
channel = channelPool.take();
channel.basicPublish("", Constant.QUEUE_NAME, null,
msg.toString().getBytes());
res.setStatus(HttpServletResponse.SC_CREATED);
System.out.println("Sent " + msg + " to rabbitmq");
res.getWriter().write("Sent " + msg + " to rabbitmq");
channelPool.add(channel);
```

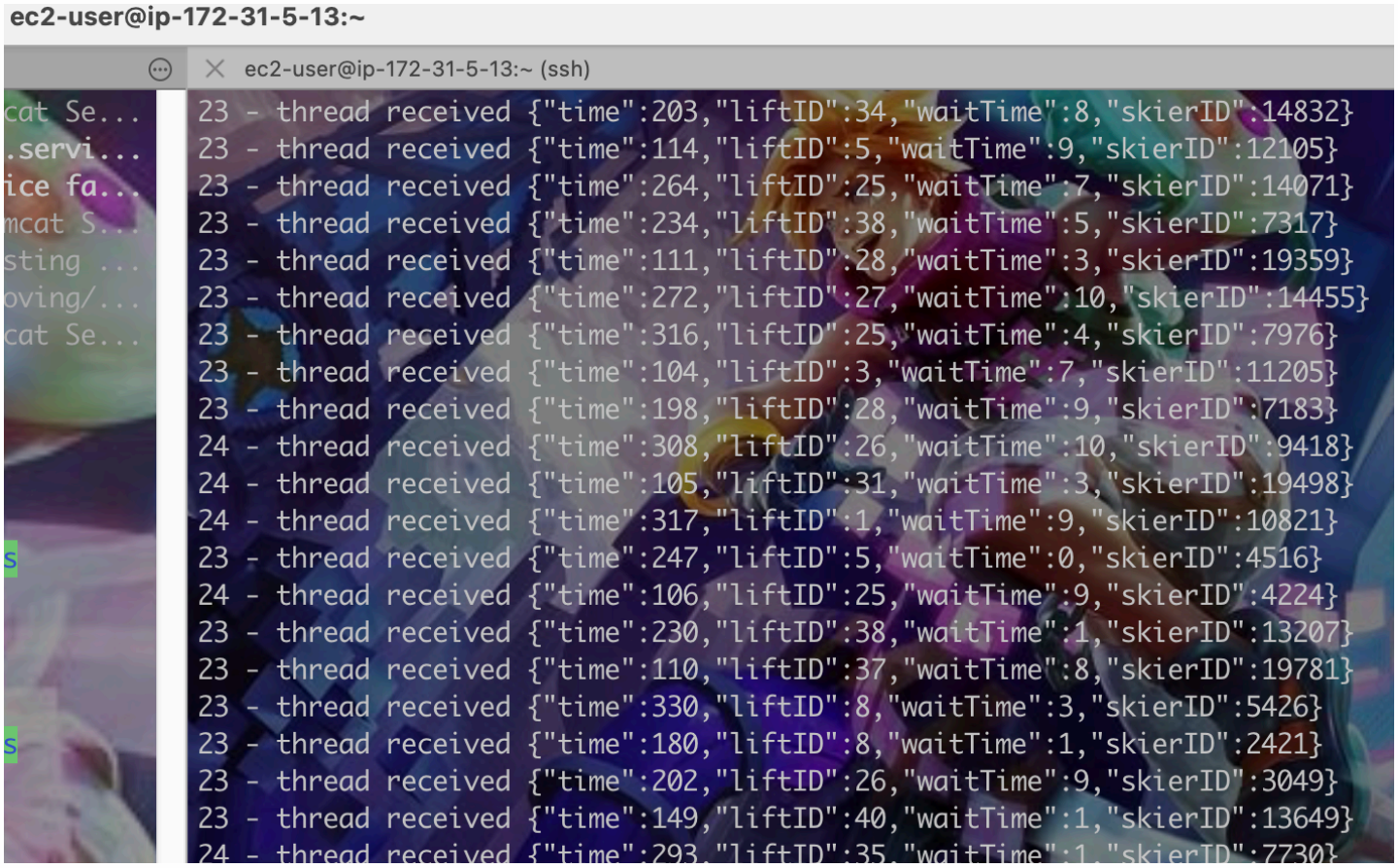
Others

For detailed design, please see the codes.

For your convinence, I build a docker image based on the **Dockerfile** . You can also `docker pull zjdx1998/consumer:mar11-amd64-latest` then `docker run -it --rm --name consumer zjdx1998/consumer:mar11-amd64-latest` to start the consumer service.

Or you can build manually to get a consumer run in the local environment.

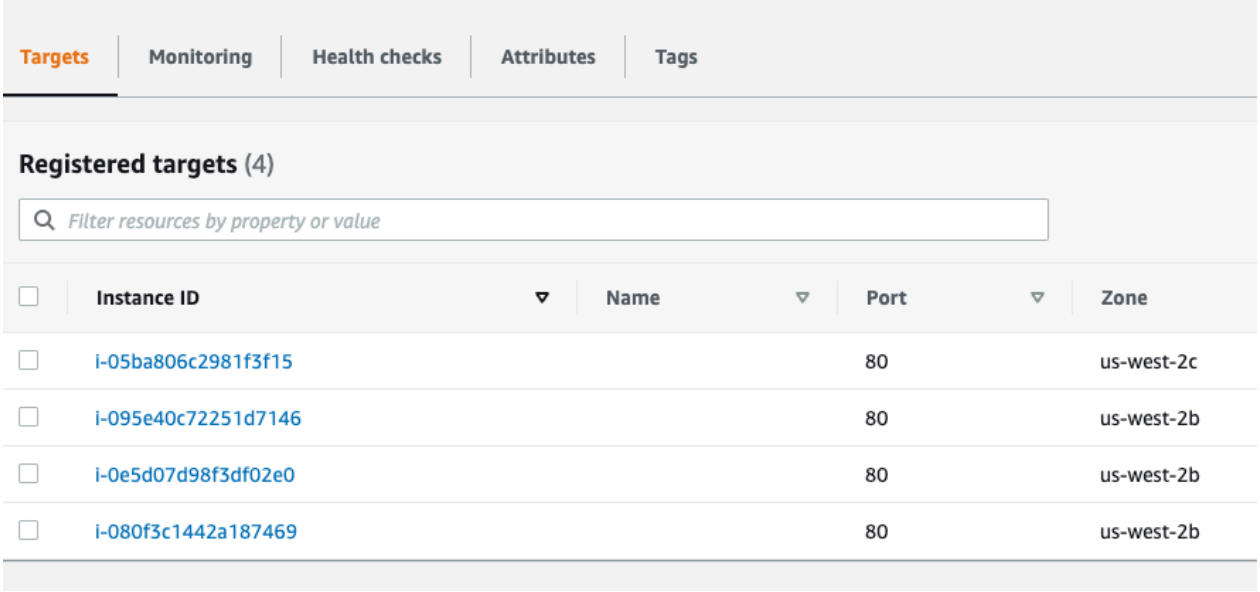
Here is a output window for remote Consumer.

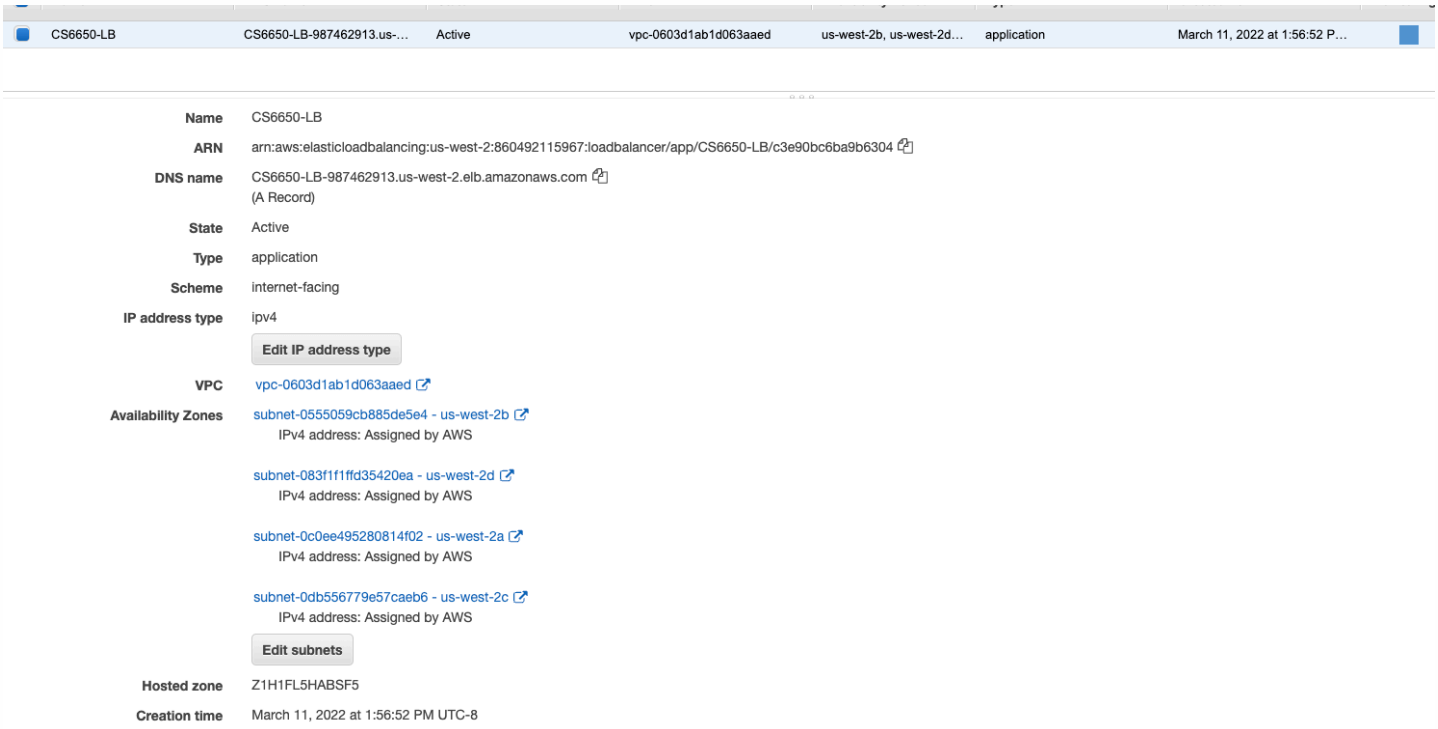


For Clients, there is no change since last project. But the parameter should be changed from single instance public url to load balancing url.

```
java Client -nt 128 -ns 20000 -nl 40 -nr 10 -server CS6650-LB-987462913.us-west-2.elb.amazonaws.com ec2-54-149-212-65.us-west-2.compute.amazonaws.com localhost:8080/Server_war
```

Load Balancing:

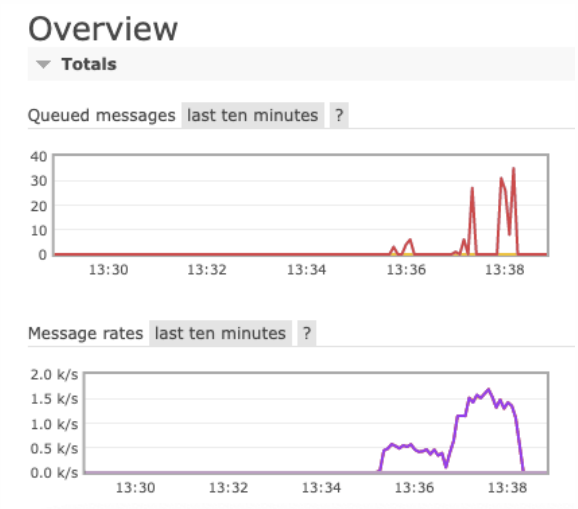




Results Analysis

Do we really need load balancing?

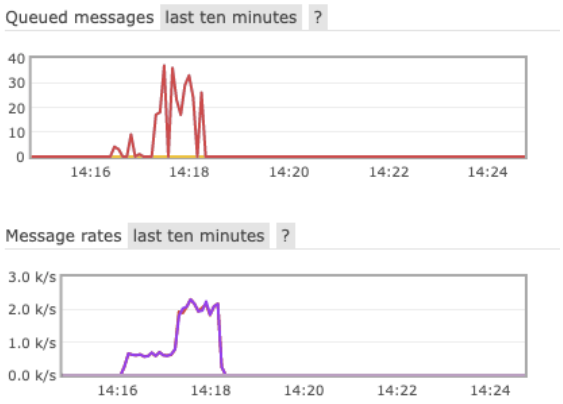
64-20000 Without Load Balancing




```
/Users/jeromy/.sdkman/candidates/java/11.0.14-librca/bin/java ...
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: 157374
Number of Unsuccessful Requests: 0
Total run time: 178026 (ms)
Total Throughput in requests per second: 883.9944727174683
Mean response time: 44.71920675352064
Median response time: 34.0
Throughput: 22.36175622505362
99th response time: 170.0
min and max response time: min: 12.0 , max: 5049.0

Process finished with exit code 0
```

Here is the result of `numThreads=64 numSkiers=20000` with two instances Load Balanced.




```
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: 156590
Number of Unsuccessful Requests: 0
Total run time: 122976 (ms)
Total Throughput in requests per second: 1273.3378870673953
Mean response time: 30.33827809835061
Median response time: 24.0
Throughput: 32.96165974740559
99th response time: 110.0
min and max response time: min: 12.0 , max: 3931.0
```

From the chart we know all throughput, total run time, message rates have a significant improvement.

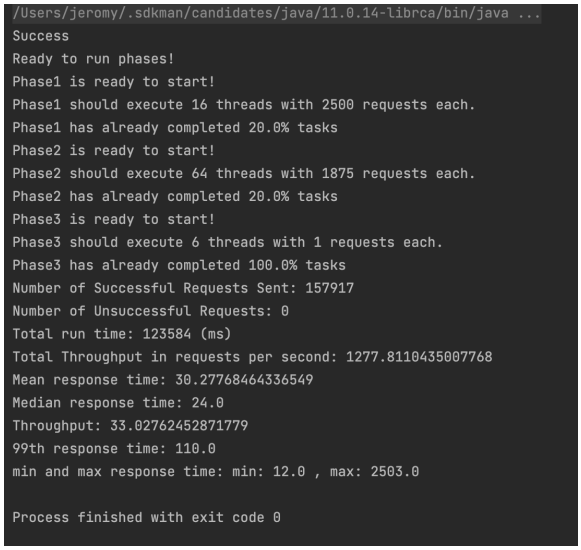
The improvement percentage for each parameter is

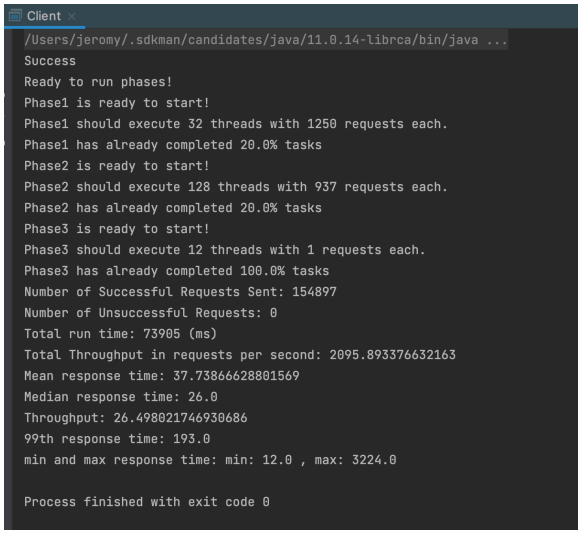
$$
\begin{aligned}
P_{throughput} &= \frac{1277.33}{883.99} = 1.445 \\
P_{totalRunTime} &= \frac{122976}{178026} = 0.6907 \\
P_{messageRates} &= \frac{2.4k/s}{1.6k/s} = 1.5
\end{aligned} \tag{1}
$$

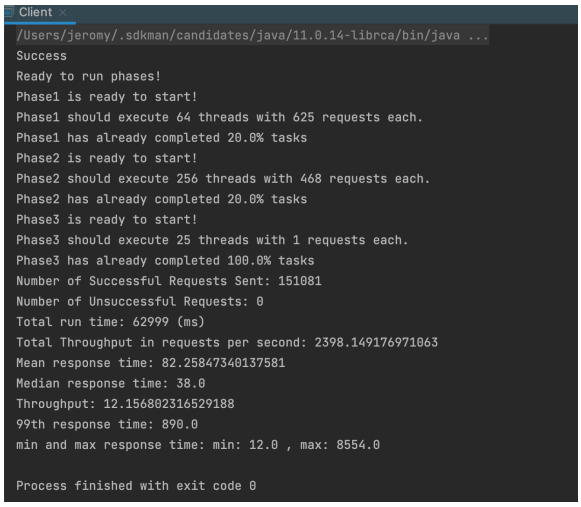
So we can get the conclusion that we need load balancing!

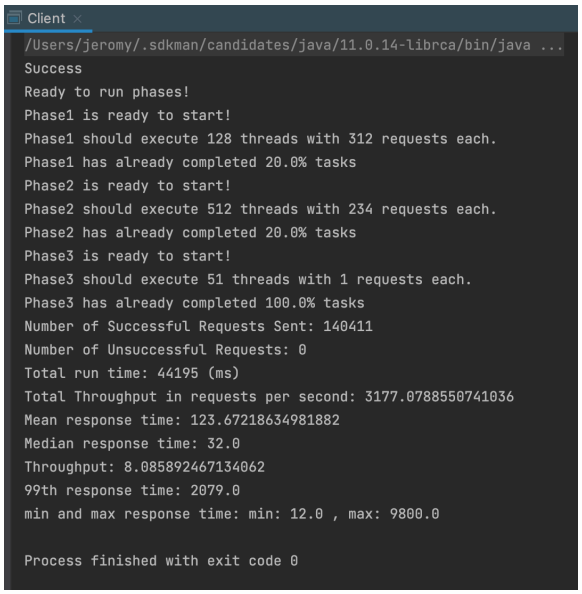
Here is a result for `numThreads=64, 128, 256 and 512` with 4 instances load balanced.

And the parameter for consumer is `numChannel = 10, basicQos = 0`.



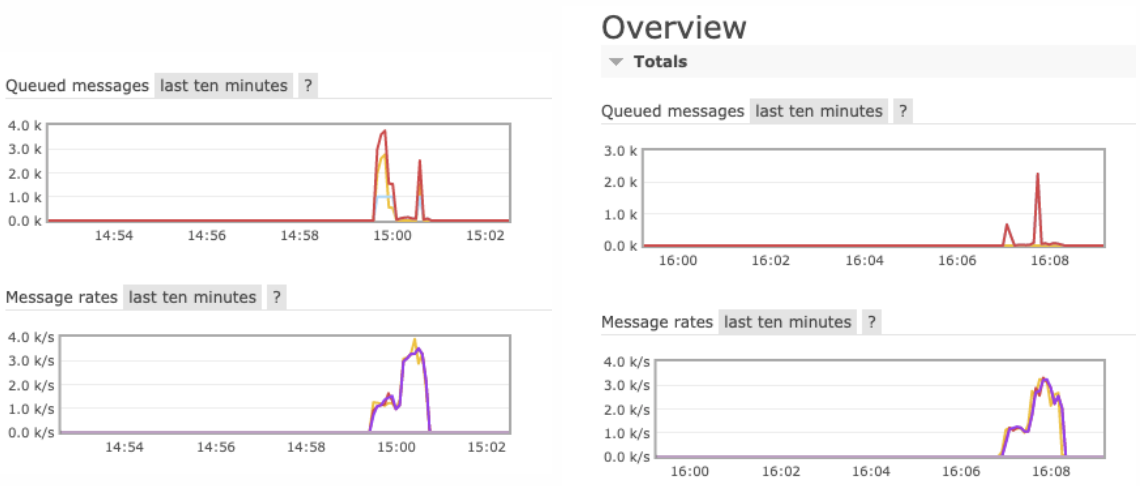






Some analysis on number of threads of consumer

Below are two images with `numThreads=128 & numSkiers=20000 & numChannel = 100 & basicQos = 10` and `numThreads=128 & numSkiers=20000 & numChannel = 100 & basicQos = 100`.



From which we can see the queued messages for the one with larger `basicQos` is noticeably smaller than the former.

But if we enlarge the `basicQos` until it become 0, the number of queued message is getting smaller then larger. That's an interesting thing.