Performance evaluation report
- load balancer assignment -

Student info

Name:	**Cursaru Razvan-Mihai**
Group:	**341C5**
Completed parts:	**All excepting the bonus**
Expected grade:	**10/10**

System Limits Analysis

1. How many requests can be handled by a single machine?

I considered 15000ms the maximum response time for a good request. (Basically half of the classic 30s timeout) Anything above that threshold is just too much for a human to wait. I observed as an average of 500 concurrent requests on any of the 5 individual machines at a time will be just below this 15 seconds threshold.

2. What is the latency of each region?

By doing 10 requests and averaging the latency obtained from a simple curl command to the 3 regions I observed that the latencies are:

Emea -> 330ms, US -> 370ms, Asia -> 520ms

3. What is the computation time for a work request?

For all machines the computation time is around 20ms.

4. What is the response time of a worker when there is no load?

Basically the latency - work_time, in the same average manner:

Emea -> 310ms, US -> 350ms, Asia -> 500ms

5. What is the latency introduced by the forwarding unit?

I measured that by doing 10 requests to the local hosted forwarding unit, and 10 request to the heroku URL directly, computing the averages and subtracting them:

Emea -> 160ms, US -> 200ms, Asia -> 350ms

We can observe that although all services are deployed in Europe, forwarding unit is artificially adding latency to simulate the distance.

6. How many requests must be given in order for the forwarding unit to become the bottleneck of the system? How would you solve this issue?

This depends a lot on the machine where the forwarding unit will run and on its inside implementation. Based on the number of operations of the CPU, considering it is an asnyc implementation and that the network has an infinite bandwidth we can say that the forwarding unit will become the bottleneck when it will receive that many requests (+1) that the CPU cannot handle, so it will queue requests forwardings and it will produce higher latencies than the one artificially added discussed in the question 5.

I will solve this issue by using a distributed system, like a load balancer for proxies, so when many requests hit at the same time would not be handled by one unique FU.

7. What is your estimation regarding the latency introduced by Heroku?

I computed that by taking from any of the services instances logs the sum of connect time and service time when there was only 1 request to the service.

Did that 10 times in a row and averaged, resulting in an around 22ms latency added by Heroku, not bad at all.

8. What downsides do you see in the current architecture design?

The major downside is that all the traffic goes to an unique FU. (More details can be found in question 6). Another downside can be the usage of Heroku:

- cannot handle too many requests at the same time, in idea that it will stack up response time quite easily.
- adds a bit of latency described in question 7.
- goes idle in 15min if no requests are made.
- maximum response time is 30 seconds which is small if trying to do huge backend processing like ML services for example.

Implementation

The load balancer starts with waking up all heroku instances by sending 1 dummy request to each one. After that, based on the POLICY_ID argument given it splits up the number of requests in the desired manner. The implemented policies are:

1. Sending all the requests to the service with the lowest latency (Emea 0)
2. Splitting the requests equally to all 5 services.
3. Splitting the requests proportional to base latency of each server, described above in question 2, sending 25% to Asia, 35% to US, 40% to Emea
4. Sending all the requests in a random manner to all 5 services.
5. Splitting the requests equally to all 3 regions, but sending randomly to region servers.

The dummy requests are sent synchronized because they do not count.

All of the requests that have to be load balanced are sent asynchronized, without waiting in a blocking manner for the response. This way I am truly simulating the load.

Usage

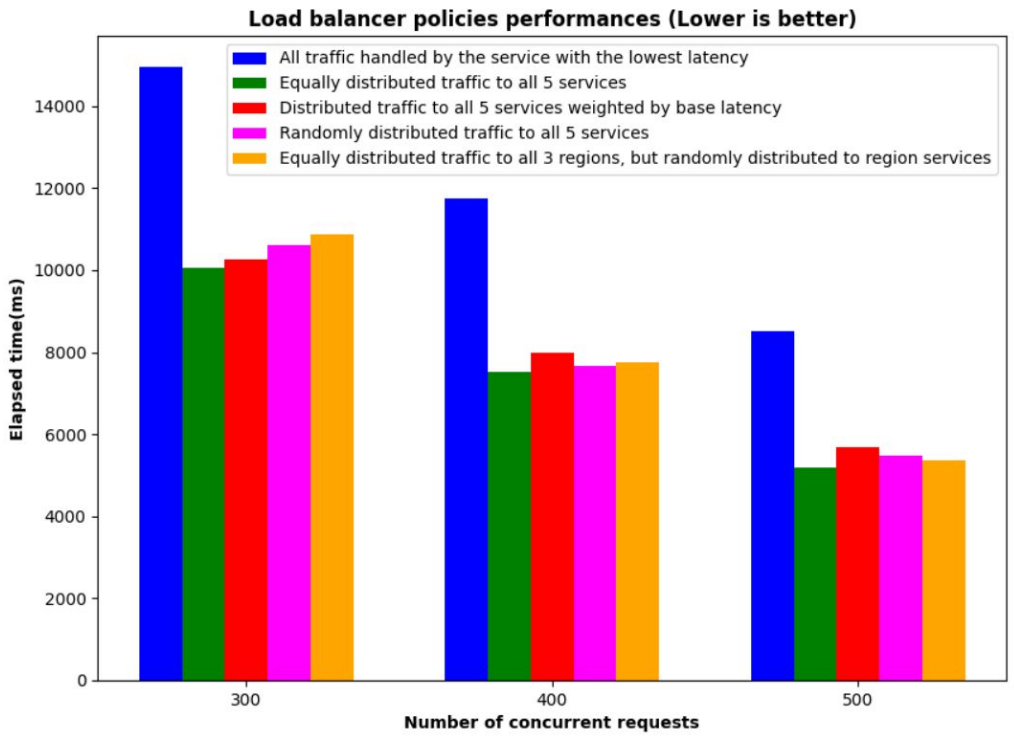
```
python load_balancer.py <NR_OF_REQUESTS> <FU_BASE_URL> <POLICY_ID>
```

Load Balancing Policies Comparison

As described above in question 1, I observed that a number of 500 concurrent requests will be enough to hit my 15s threshold, and because the first policy will be obviously the worst one, I decided that more than 500 requests at a time would not be relevant for my comparison.

I measured the total elapsed time for all of the policies 10 times when hit by 300, 400 and 500 concurrent requests and averaged the elapsed times.

Here is a self-explanatory graphic showing results:



It can be immediately observed that the first policy is the worst, because it is using only 1 service out of 5, so having a low latency server is not enough.

An interesting observation is that distributing based on the base region latency did not improve the elapsed time, because at a high number of requests the differences in response times are very small, and it is better to distribute them equally as shown.

Also, both policies distributing traffic randomly are quite the same on average measurements. When hit by a high number of requests, the one with only regional random distribution performed better because of the smaller change of unequal random distribution, while the other one performed better on smaller numbers.