Performance Evaluation Report

Matei Barbu, 341C2

19 decembrie 2022

Abstract

This report contains my solutions for the Performance Evaluation course assignament. I completed all of it (except bonuses) and consider this report should be evaluated with 100 points.

Contents

I Prerequisites

Contrary to official recommendation, I took the liberty of running this homework on a native Linux machine (Debian 11 with kernel `5.10.0-20-amd64`) with a Intel i7-8550U processor and Python `3.9.2`.[1]

I have used a virtual machine only for bandwidth testing and for generating the `requirements.txt` file.

I.A Mininet machine

```
$ sudo apt install mininet
$ sudo /usr/share/openvswitch/scripts/ovs-ctl
↪  start
```

I.B Run the topology

```
$ sudo python3 topology.py matei.barbu1905 -t
matei.barbu1905
...
Running base test with only one server
Done
*** Starting CLI:
stopping h1
```

[1] Because I don't know how much the Mininet actually uses the network card, for all intents and purposes it is a Intel Wireless-AC 9260 rev. 29.

Host	Response procentage
`h1`	99
`h2`	23
`h3`	31
`h4`	26
`h5`	2
`h6`	2

Table 1: Ilustration of the procentage of responded requests in a houndred sized batch (using the original `client.py`) with a 2 second timeout.

II Evaluation - System Limits Analysis

When answering any performance questions in this analysis, I measured the latency of a route by dividing in half the average round trip time for a hundred control packages (ICMP echo). Note, I chose not to filter any outliers, because other (unknown to the user) control packages and internal host parameters are expected to cause delays. Loss percentage was measured in the same conditions. Keep in mind that latency is influenced by package loss (i.e. latency is directly proportional to it).

Bandwidth, on the other hand, was provided by `iperf` (as a mininet built-in), specifically it's first return value (i.e. the servers reported bandwidth). Bare in mind that any tools which uses IP addresses cannot measure bandwidth between a host and a switch. For reasons unknown to me, measurements could only be performed on the virtual machine. For this reason I tested this part manually, all of the rest being automated in `test.py`

> How many requests can be handled by a single machine?

The ambiguity of this question, alongside with the technical particularities of a network configuration make it impossible to answer or simulate. The biggest problem arises from not knowing the network cards on each host. If it supports LSO [1] then simple bandwidth information will not suffice. When using Mininet we face scheduling challenges, how threads are given CPU time, and which of these are chosen with respect to the hosts/clients interaction. In the end this debate will be settled by the throughput of each servers connection to the command unit. See table 1 for values.

> What is the latency of each region?

I consider the latency for a region to be equal to the

Region	Latency (ms)
ASIA	36.881
EMEA	43.34425
US	20.09025

Table 2: Latency for each region.

average latency any route from the command unit to any host in that region (because there weren't any user in the given topology). See table 2 for values, precision was observed amongst runs.

> What is the server path with the smallest response time? But the slowest?

US's top dollar acquired the most responsive servers, host 5, as opposed to EMEA's old-timer, host 4.

> What is the path that has the greatest loss percentage?

The most (unique) packets lost are those intended for host 6 along the path starting from the command unit. While other hosts can rank the same as the former, this result is the most consistent along multiple runs of the same test. [2]

> What is the latency introduced by the first router in our path?

Without router 0, I imagined that routers 1, 2 and 3 would be connected by a direct link to switch 0. So the latency introduced by the first router is equal to the average of all links from router 0 to 1, 2 and 3, which I measured to be in the range of 8.8014 to 11.911, around 9 most times.

> Is there any bottleneck in the topology? How would you solve this issue?

A bottleneck should be any low bandwidth connection. [3] See table 3 for values. [4]

A (kind of obvious) solution would be to change these links. Another, (unreasonable and forced) idea is to add servers, as many as it takes, to compensate for the limited requests/second throughput.

> What is your estimation regarding the latency introduced?

This one's trickier to answer. A naive approach was to calculate the latency from switches 1, 2 and 3 to their connected hosts, and subtract those values from the latencies of the command unit accessing the same hosts. Such a simple analysis will not suffice, because it doesn't take into account scheduling latencies (inside the load balancer) and how many collision domains are configured on the switch.

Given how I defined latency for each region to be, a rough approximation of "the latency introduced" could be the average of tables 2 values, 33.438 ms.

Client	Server	Bandwidth (Kbits/sec)
`c0`	`r0`	19.6
`r1`	`h1`	15.6
`r0`	`r2`	12.7
`r0`	`r1`	10.5
`r0`	`r3`	3.74
`r1`	`h2`	1.72
`r2`	`h3`	2.83
`r2`	`h4`	2.85
`r3`	`h5`	0.97
`r3`	`h6`	1.83

Table 3: Bandwidth between two station, one acting like a `iperf` client , and another like a server. Separated by a double horizontal line, are on top the high bandwidth links and low ones below.

> What downsides do you see in the current architecture design?

There is, without a doubt! The fact that every request passes through the command unit. Let's say 1000 "newly logged in" users in the US want to access a server. And there are 2000 users in ASIA in the same situation. So 3000 requests will assault the control unit.

Another downside is the infrastructure cost. Given that the block `10.0.0.0/8` is set aside for use in private networks [2], this means the servers in any region are inside the same private network.

A notable alternative is one based on anycasting (a bigger bang for your buck). However there is no load balancing (or has inherit firewall problems). [3]

III Implementation

In order to call the exposed endpoints of the topology depending on the number of request, we are forced to simulate user connections on the command unit, for there are no user nodes in the topology. To accurately do so, with some precision loss, it is inevitable to run requests in parallel, or at least concurrently.[5]

Please note that it is not specified whether or not the load balancer uses a parallel scheduler to deal with many newly incoming connections (see [4] to understand why parallelism or asynchronous operations are needed).

Because of the ambiguity previously mentioned I shifted my focus from a master-centric view of the scheduling to a time dependent functional model, captured in a scheduling function which simply answers what destination does each request have. This approach

[2] Decompiling the topology setup script, yields a different answer. However, after thorough testing I have arrived at the conclusion that there is a decoupling between the ping implementation (for example a moving window average) and the analytical statistic approach, the result of which I was expecting.

[3] Because a physical bottleneck is any point in which flow is obstructed.

[4] Decompiling the topology setup script, yields a higher value for all results. Given that all measurements had about the same accuracy, the difference is tolerable.

[5] Users are not in contention, i.e. there are no synchronizing elements between any of them. Sure there is the inherit sequencing of requests/responses on a physical link, but there is no ordering.

has it's drawbacks because response times will depend on arrival rates.

I chose three functions which fit our criteria:

- round robin
- a function which returns a uniformly distributed random host ip
- a function which returns a random variable whose distribution is weighted according to the responsiveness of all hosts measured at runtime, similar to table 1); this make sense in a production environment where client connections are expected to have timeouts configured, however it has the downside of stressing cross-regional servers, which may not make sense from a business perspective

Because I find it difficult to model my solution in Kendall's notation terms, let's jump into the technicalities. (Just to mention my intuition is that arrivals are according to a Poisson process, there are 6 service nodes, but latency won't cut it for specifying service times.)

III.A Solution

On the command unit I ran `test_distributions.py` which sends a 300 (running time justified value) requests using a `ThreadPoolExecutor` with implicit `max_workers` ("it will default to the number of processors on the machine, multiplied by 5"). All threads call a scheduling function which only synchronizes them when it models a round robin choice. All request are uniform accessing a directory listing. However all content but the response time is discarded to remove writing speed limitation of our flooding.

I penalized connection errors with 3 seconds because it is the tipping point at which all servers show improvements in terms of responsiveness.

III.B Efficient Policies Comparison

Because of the detachment form a arrival rate distribution and page space limitations, I was only interested in the average response time of each server, instead of plot of each response time per host. A classical mean function was used between all measurements. Figure 1, depicting these results, was generated with `chart.py`.

The responsiveness weight values are those depicted in table 1, but adjusted so that the first hosts weight is with 10% less, because the original value badly flooded that path. As a rule of thumb, when a path to a server is severely flooded, all servers in that region become unresponsive. This is why both hosts in the US region fall behind in the round robin/random scheduling and why ASIA second hosts starts lagging for the last experiment.

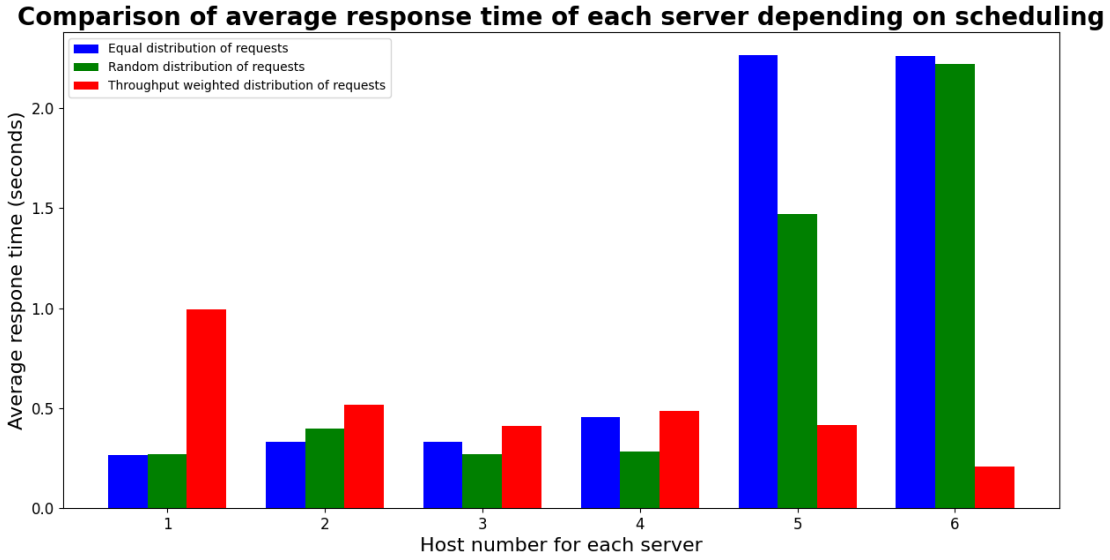

Figure 1: Plot of average response times for each host and for each scheduling function. As it can be seen only the thrid option minimizes our time cost evaluation and shows that an nonadaptive solutions isn't perfect.

Bibliography

[1] J. Corbet, "Large receive offload." 2007. Available: https://lwn.net/Articles/243949/

[2] M. Cotton and L. Vegoda, "Special use ipv4 addresses," 2010.

[3] W. Herrin, "Anycast TCP architecture." 2017. Available: https://bill.herrin.us/network/anycasttcp.html

[4] D. Kegel, "The C10K problem." 8 May 1999. Available: https://lwn.net/Articles/243949/