Report 1: Rudy

Patrik Zhong

October 1, 2021

1 Introduction

In this assignment we created a very rudimentary server to show how erlang basically was built for these types of problems. Erlang uses very little synctatic sugar and has prebuilt functions to handle networking, such as the gentcp functions.

2 Main problems and solutions

Most of the code was already given, and the descriptions of how to write out the missing parts were also very clear on what we were to do. Code-wise, it took very little effort to implement the small web-server. However, managing to run the testfiles were a bit tricky for me. Our prebuilt benchfile took two arguments, the "Adress" and the "port". The port was very straightforward, as all we really had to do was to write in 8080 as our port. But the adress part was a little bit harder to interpret. I tried using my own IP adress, but had issue with the formatting since erlang disliked me using dots to format the IP. I tried also encapsulating them in char signs, but that didnt work either. It took me half an hour and then re-reading the instructions to realise that I simply had to ping localhost....

3 Evaluation

The first that I tested was how much slower it was when doubling the default 100 to 200, the time it took to process the request doubled. This can be shown in the graph below, where the amount it takes is "ish" doubled. The reason why it isn't truly linear might be due to processes taking different power is that the computer isn't in the same state all the time. There's a certain level of variability.

When running the program on another terminal (to represent running it on several machines) we saw that when requesting from the same host, the amount of time basically doubled. Running with a 40 sleep and 100 requests

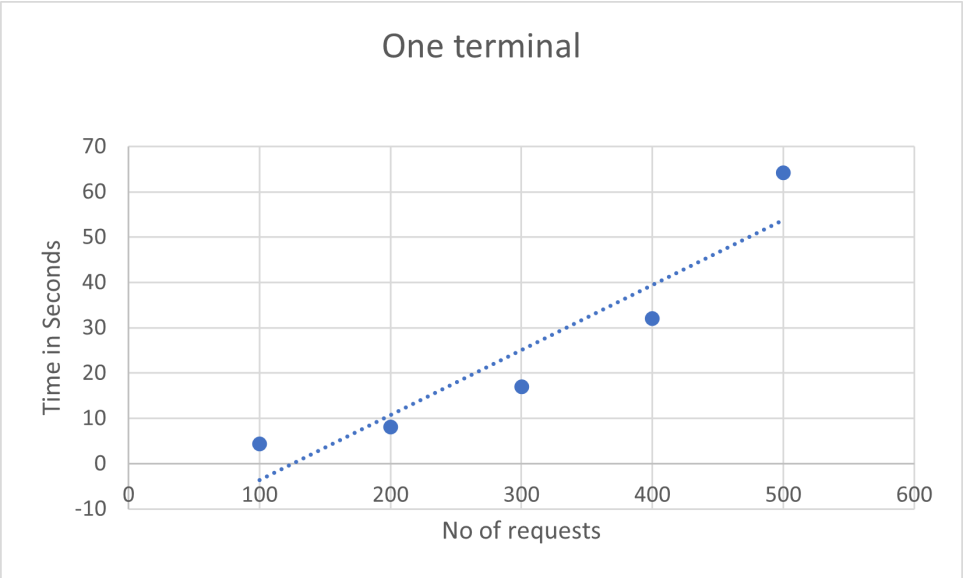

Figure 1: Results from running one terminal

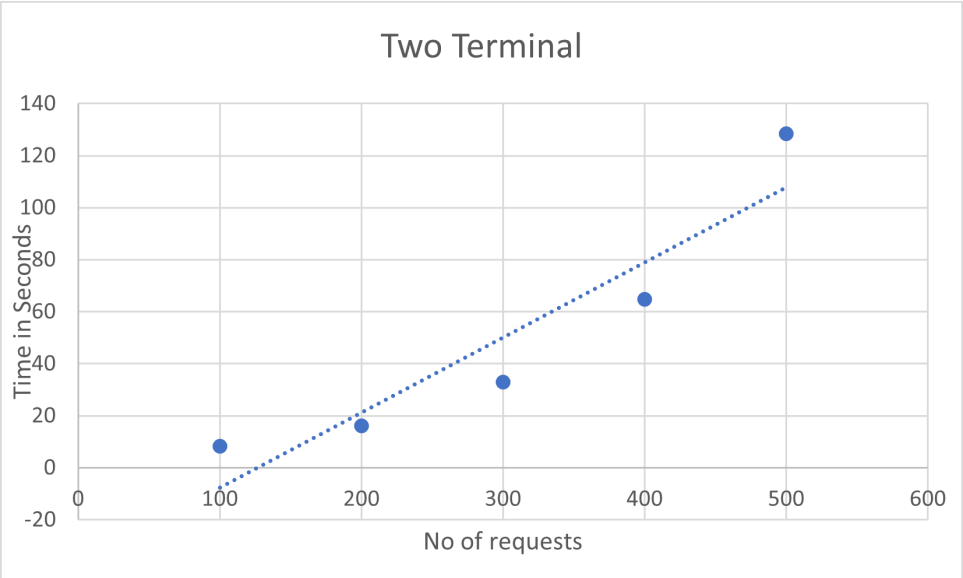

Figure 2: Results from running two terminals

was 4 seconds, but when doing that from two terminals at the same time, it came down to basically 8 seconds.

The last thing I checked was how the time differed when changing the sleep value. Since the sleep value basically works as an "artificial" time payload that just increases each request, I figured the time for each request would double, which it did, just like the previous graphs (which is why I didnt list the graph). The sleep seems to overshadow the overhead of the "send" and "request" functions.

4 Conclusions

This problem taught me how basic network connectivity works in erlang, and how easy it is to implement certain parts relative to how tricky it is in Java. It's very interesting to see how by just typing in the HTTP parts the system recognizes and allows for it.