Report 1: Homework Report Template

Marcus Samuelsson

September 11, 2023

1 Introduction

In this task a small web server was created using the programming language *Erlang*. The different parts of the web server includes an *HTTP parser*, a *server* and a *benchmark* for testing. The purpose of this task is to get a better understanding of *Erlang*, as well as, getting a pictures of the difficulties that come up with distributed systems.

2 Main problems and solutions

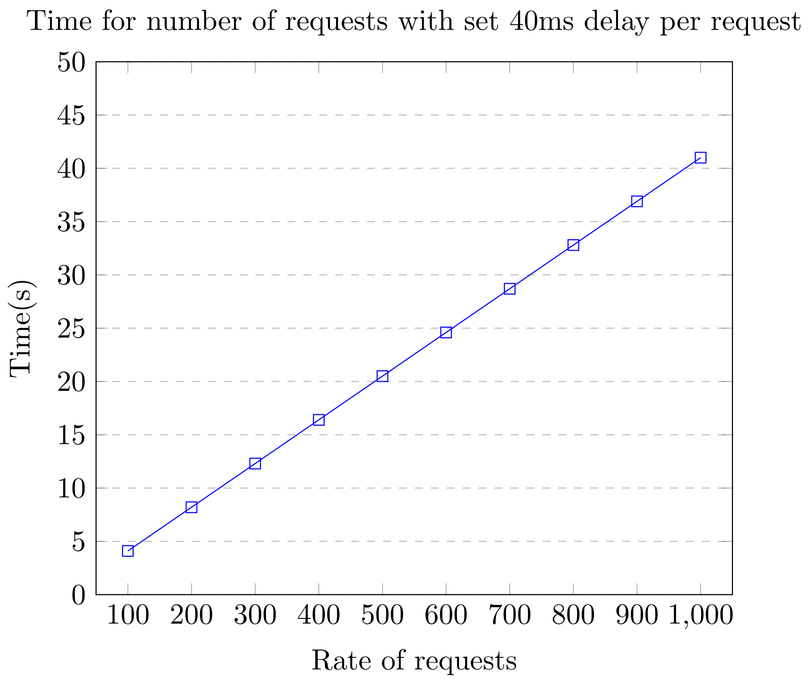
The problems that came up during this task was mostly around understanding *Erlang* and the purpose of the given functions and how to properly run the program. *Erlang* being a functional programming language, which is very different from the other more commonly used languages like *Java*, *C* and others. This made the learning curve much larger then it might be for someone used to object oriented programming. However, through reading documentation and continues testing of the code and the different parts it all started coming together.

The second issue that arose was on how to compile, run and properly give parameters when ruining the program in the *Erlang* terminal. Compiling and running the program was not as clear as other languages might have been in the documentation, which made it difficult to start testing the program so that it could become more understandable. This was eventually solved by trying different things listed in the documentation until the program was able to run. The next issue in the terminal was how to pass a *IP-address* and port as parameters, which caused issues as the *IP-address* contains punctuation's. This was only solved through trial and error until it worked.

3 Evaluation

3.1 Single shell test

Running the test from 100 to 1000 request in a with one shell sending the requests gives the results in the graph (figure 3.1) and the table 3.1.



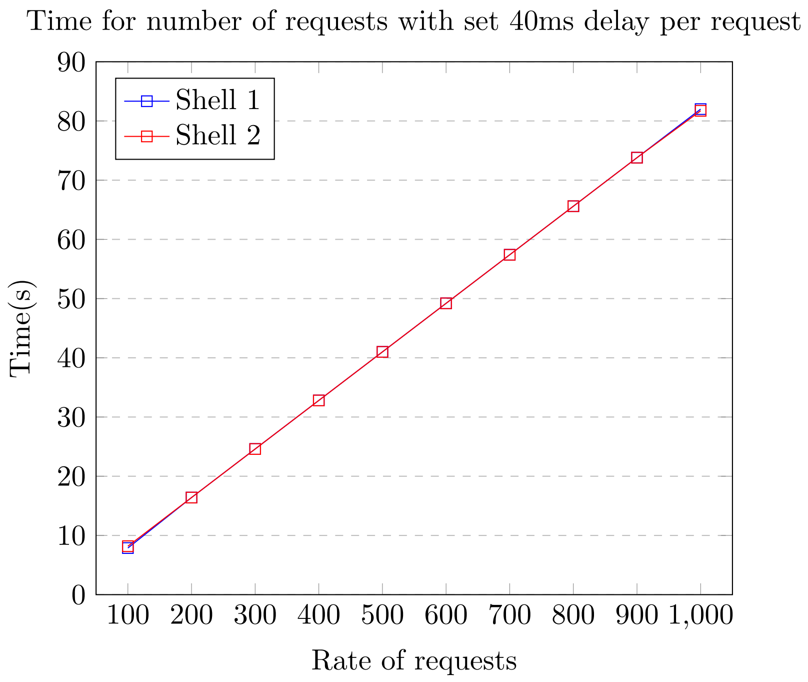
The result shows that the it increases linearly with about 4.1 seconds for each increase of 100 requests. This gives that per request the overhead 1 milliseconds per request as we have a delay of 40 milliseconds per request to simulate work. As this test was performed on multiple terminals on one computer, it could be excepted that the overhead would increase if it ran over the network to another machine.

Nr of requests	Time(s)
100	4.099806
200	8.199810
300	12.299919
400	16.399830
500	20.499955
600	24.599937
700	28.699862
800	32.799976
900	36.899891
1000	40.999922

Table 1: Some random results in a table

3.2 Two shell test

Running the test from 100 to 1000 request in a with two shell running simultaneously and sending the requests gives the results in the graph (figure 3.2) and the table 3.2.



The result here shows a clear change in the total time for the requests. All the times have doubled form Section 3.1. The two runs look very similar and do both increase linearly, the only difference is in the first test of 100 requests. The total head room for these request is 4.2 seconds given as the 40 millisecond delay for each request. However the double time is most likely caused by the fact that the program in not concurrent, which causes the program to jump between the two request. So with that in consideration, the actual overhead matches with the previous of 1 millisecond per request. This can be seen as we take away 80 milliseconds and split the rest on the two shells. The different between the first values is probably cause by the programs being started manually and not at the exact same instance.

Nr of requests	Shell 1 time(s)	Shell 2 time(s)
100	7.871741	8.183065
200	16.399880	16.397761
300	24.599932	24.599961
400	32.799995	32.799906
500	40.999819	40.999936
600	49.199940	49.199923
700	57.399932	57.399920
800	65.599891	65.599904
900	73.799981	73.799896
1000	81.999913	81.712927

Table 2: Some random results in a table

4 Conclusions

In conclusion the main issues with the task was just getting used to using a functional programming language and the confusion over some of the documentation. But this was solved by spending more time and by trial and error. The results shows that the server works linearly and holds the same overhead with increased requests. Running multiple shells at once gives similar results as with one with the biggest issue of speed being caused by the program not running concurrent. Lastly, this task gave a better understanding of Erlang and how a web server works.