Report 1: Rudy - a small web server

Chen, Zidi

September 4, 2020

1 Introduction

In this seminar, I constructed a HTTP get method parser according to the tutorial. It identifies request-line, header, and message body from the request. Then, The Rudy server was implemented by using the parser and it implemented the sockets. There was also a test program for calculating the response time for sending a request to a server. In addition, an improved server that can deal with several concurrent requests and listen to the same socket was also implemented. The test program for the improved server was also done.

This is of great importance in distributed system, because this is a very basic and common way for communicating between machines.

2 Main problems and solutions

Although I am new for erlang, the tutorial file really helps for me. I was able to fill in the blanks in the program. The main problem for me is to how to set up a server which can handle requests concurrently. (the first bonus task)

In the improvedServer file, I set up a server which can address this problem with the help of the example rudy4 program. I can enter how many concurrent server process that I want and then a handler pool is generated. For example, if I enter 3 for the input value, then there will be 3 handler processes. When there are 3 concurrent requests coming from the port, the server can assign them to the 3 handler processes. The requests can be addressed concurrently.

3 Evaluation

With the basic server, I ran it from my machine (figure 1) and from another computer (figure 2) with the artificial delay of 20ms.

Figure 1: Test with 20ms delay from my machine

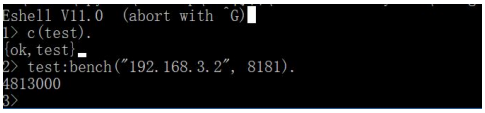

Figure 2: Test with 20ms delay from another machine

There is a big difference. I think this is because accessing to my machine from another machine needs to go through the router.

Figure 3 shows the time difference between having artificial 0ms, 20 ms, 40ms and 60ms delay in the server (request for 100 times). The time difference is quite significant. According to the figure 4, the delay and the response time might have a liner relationship.

test request without delay											
	1	2	3	4	5	6	7	8	9	10	average
time(s)	0.5	0.562	0.422	0.563	0.547	0.609	0.531	0.594	0.562	0.422	0.5312
test request with 20ms delay											
	1	2	3	4	5	6	7	8	9	10	average
time(s)	3.234	3.14	3.141	3.125	3.125	3.125	3.125	3.125	3.125	3.141	3.1406
test request with 40ms delay											
	1	2	3	4	5	6	7	8	9	10	average
time(s)	4.719	4.719	4.797	4.891	4.75	4.703	4.766	4.719	4.766	4.781	4.7611
test request with 60ms delay											
	1	2	3	4	5	6	7	8	9	10	average
time(s)	6.765	6.718	6.781	6.828	6.75	6.688	6.688	6.75	6.734	6.688	6.739

Figure 3: The response time of server with different artificial delay

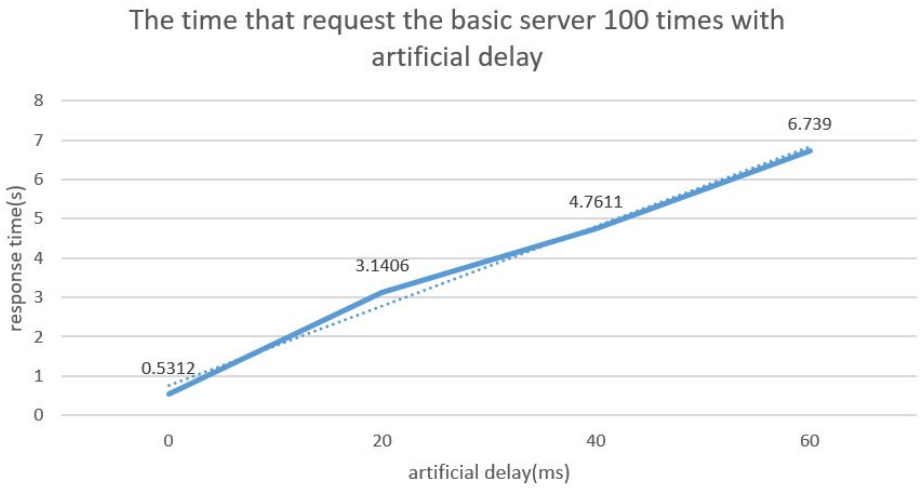

Figure 4: The chart of response time of server with different artificial delay

I have also implemented an improved server which can handle many requests at the same time, and an improved test which can request at the same time. When I used the improved test class to request the basic server with 100 concurrent processes, the basic server failed and gave many error reports. When I tested the improved server with 100 concurrent process, the result shows in the figure 5. The improved server can work like parallel servers to deal with concurrent requests.

```
2> improvedServer:start(8183,100).
true
3> improvedTest:bench(localhost,8183).
 100x10 requests in 5031000 ms
ok
```

Figure 5: Test for the improved server

4 Conclusions

From this experiment, I learned the basic knowledge of erlang, the basic idea of implementing an http parser, how to set up a simple server and how to use sockets for communication. In addition, I also learned how to set up parallel process in erlang.