Report: Rudy A Small Web Server

Lakshmi Srinidh Pachabotla

September 11, 2024

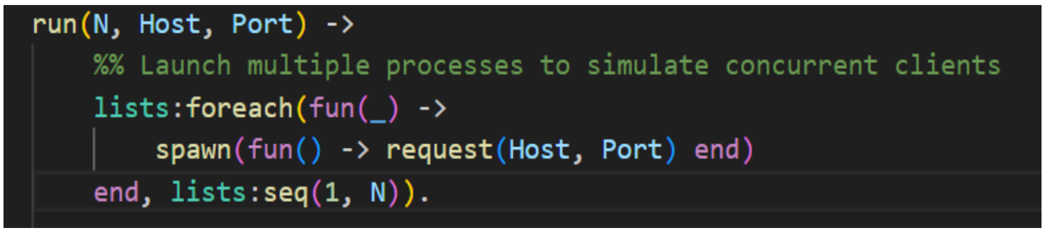
1. Introduction

A small web server, named "Rudy," is implemented using the Erlang programming language. Erlang is a functional language which effectively used for creating distributed and concurrent systems. The complexities of managing socket connectivity and interpreting HTTP requests in the Erlang programming language are also learned through this assignment. The task also entails enhancing the server's functionality to accommodate increasingly intricate use cases, such as file delivery, multi-threading, and concurrent request processing. Building a basic web server that can process HTTP GET requests and provide basic replies is the task at hand.

2. Main Problems and solutions

The main problems faced during the implementation of the assignment are:

- ➢ The first problem would be about the Erlang programming language, as it is new and a bit different from the others like C, C++, etc. At times, it will be like a deadlock situation and it needs some time to understand and then execute it. Although, it was a bit difficult in the starting, but later after going through some tutorials and notes, and then implementing it was fun like starting the server and compiling the codes.
- ➢ The second problem with the assignment is that, implementing the concurrency i.e. Multi-Threaded Performance. There was no problem in implementing the sequential. But, for the implementation of concurrency, the system displayed some errors. There were no errors in the test.erl (multi-threaded performance) code. After browsing the internet and reading some materials, this problem was solved finally with the help of replacing the below code in the test.erl code, the concurrency was possible.



```
run(N, Host, Port) ->
    %% Launch multiple processes to simulate concurrent clients
    lists:foreach(fun(_) ->
        spawn(fun() -> request(Host, Port) end)
    end, lists:seq(1, N)).
```

Figure 1: Replacement for the code in the test.url file

3. Evaluation

The evaluation of the Rudy is based on its functionality and performance. A few important factors are assessed to determine the server's efficiency and constraints.

- A sequential implementation which has a delay of 40ms:

S. No.	*No. of Request*	*Time Taken (in seconds)*
1	200	9.62
2	400	21.23
3	600	31.62
4	800	39.97
5	1000	48.26

Table 1: sequential implementation which has a delay of 40ms

As per the above table, time taken is calculated by "Time in Seconds = Total Time in Microseconds / 1,000,000". "Throughput = Total Number of Requests / Time in Seconds". After calculating the average throughput is 99.35 / 5 = 19.87. Hence, on average, the throughput is 19.87 with a delay of 40ms.

- A sequential implementation that has no delay:

S. No.	*No. of Request*	*Time Taken (in seconds)*
1	200	0.67
2	400	1.31
3	600	1.99
4	800	2.49
5	1000	2.74

Table 2: sequential implementation which has no delay

As per the above table, time taken is calculated by Time in Seconds = Total Time in Microseconds / 1,000,000. Throughput = Total Number of Requests / Time in seconds. After calculating the average throughput is 1591.58 / 5 = 318.31. Hence, on average, the throughput is 318.31 with no delay.

4. Conclusions

Based on the above results, when there is a delay of 40ms in the network, the average throughput is 19.87 units per unit of time. And with no delay, the throughput is 318.31 units per time. The comparison between the two scenarios clearly shows that network delay significantly impacts the throughput in a sequential request scenario. Even a relatively small delay of 40 microseconds can reduce throughput substantially when scaled over a large number of requests. Therefore, optimizing the network for minimal delays pr implementing a more efficient, parallel, or asynchronous request-handling approach can vastly improve throughput and overall performance.