Report: Rudy A Small Web Server

jinjia zhang

September 12, 2023

1 Introduction

This assignment aims to implement a small web server in Erlang, a language designed for building robust, distributed, and fault-tolerant systems, which provides an ideal platform for distributed systems. Its concurrency model, lightweight processes, and built-in support for networking make it an excellent choice for developing networked applications like web servers. Also, it is a functional programming language to make concurrent communication easier.

Besides, the simple HTTP parser is already available. What I did was to complete the Web server and make it runnable. And I wrote some test cases for the benchmark.

2 Main problems and solutions

The first problem is definitely the Erlang programming language. The syntax and programming mindset are quite different from what I learned before, such as C/C++. It took me much time to understand the code the assignment provided. After reading some tutorials and doing some experiments, I gradually can understand the Erlang coding pattern. As the programming time increases using Erlang, I believe that the mastery of Erlang will be better.

The second one is when I want to invoke one registered process from another node. Initially, I thought all registered processes are available among connected nodes. But, when I tried to find a specific registered process on another node, it failed. After searching for information online, I found there is a library called rpc I can use.

The third one is to implement the concurrency. The first idea is to create a new process for each incoming request, implemented in **rudy_multi.erl** file. For each incoming request, it spawns a new process and the process is monitored by the listener process. Once the handle process is broken, the listener process receives a signal.

Even though the process in Erlang is quite lightweight, as the assignment said, a better approach might be to create a pool of handlers, implemented in **rudy_poll.erl** file. The main process spawns a pool of handlers to listen and accept the same socket, because gen_tcp now allows for several processes to issue accept calls to the same listen-socket simultaneously. The different accepting processes will get connected sockets on a first-come-first-served basis. For further references see `https://erlang.org/download/otp_src_R11B-3.readme`.

3 Evaluation

The evaluation scenarios: The client and server are running on the same machine. The server is running on **127.0.0.1:8080**. There are five clients simultaneously doing HTTP requests to the server lasting 10 seconds.

And, there are two versions of the benchmark. The first one is provided by the assignment, which doesn't provide much information. The second is written in K6, which is an open-source and load-testing tool. The detailed testing results, whose suffix is txt, are also submitted for this assignment.

I use the 99% http_req_duration metric, which is the total time for the request, to measure the performance and the queries per second(QPS). The 99% is the time when the 99% of requests are completed.

	http_req_duration(99%)/ms	QPS
sequence	224.11	23.026519/s
multi_process	66.48	93.477758/s
process_poll	56.67	99.179086/s

As we can see, the sequence implementation has the worst performance as every request has to wait for the previous one to be handled. Also, the result reveals that creating a process pool is a better option for handling requests. Because the creating and deleting process is resources-intensive. Processes in Erlang are lightweight, but they still consume memory and CPU resources. When dealing with a large number of concurrent requests, this can lead to excessive memory usage and context-switching overhead, potentially causing performance bottlenecks. Reusing existing handler processes from a pool allows the application to maintain persistent connections to clients, reducing the overhead of establishing and tearing down connections for each request.

This web server only handles the HTTP GET method and doesn't support many functionalities that a web server should have. Besides that, now most Web API is RESTful, we need to have a matching mechanism to handle different paths.

4 Conclusions

I have learned the basic mindset when I program using Erlang, such as the immutable data structures and message passing. And, automation makes life easier when I use Makefile to automate the compilation. I hope I hope that as I study the course, I will have a deeper understanding of Erlang, so I can learn some very successful open-source projects, such as Rabbitmq. It also allows me to learn elixir more easily.