Report 1: Rudy - a small web server

Yining Hou

September 11, 2023

1 Introduction

In this report, I present a small web server called Rudy as well as some possible improvements to the basic server.

In this seminar, the main topic is about how to handle requests from clients in distributed systems. Efficiency is important when the server manages multiple parallel requests.

2 The Rudy Server

Set up a rudimentary server and access it. Use a benchmark program to do some experiments.

- `rudy.erl`: the Rudy server which sends clients' requests to an http parser and reply to the client.
- `http.erl`: the http parser mentioned above. It parses the request into `{Request Line, Headers, Body}`.
- `test.erl`: a small benchmark program which generates requests and calculates the time it takes.

Run the test program to measure the time it takes to response to 100 requests. For testing, 40ms of waiting is added to each reply.

```
1> rudy:start(8000).
true
2> test:bench(localhost, 8000).
4216578
```

The total waiting time is 0.04 * 100 = 4s. The artificial delay is significant. To calculate the time each request takes:

$$\frac{TotalTime}{TotalRequests} = \frac{216578}{100} = 0.002s$$

To calculate how many requests we can serve per second:

$$\frac{1}{0.002} = 500 requests/s$$

3 Improve Rudy

Implement some improvements to the server and compare it to the original one through some tests.

3.1 Increasing Throughput

Create a new process to handle each incoming request, so it will reduce waiting time.

```
spawn(fun() -> request(Client) end),
```

To test the improved rudy in a parallel context, send each request concurrently.

```
spawn(fun() -> request(Receiver, Host, Port) end)
```

Create a Receiver to handle completed request and errors.

```
receiver(N, Pid) ->
    if N == 0 -> Pid ! all_done;
       true ->
            receive
                done -> receiver(N-1,Pid);
                error -> ...
```

Add a third parameter to customize the number of requests. Instantiate the receiver before sending requests. Finish the timer when all requests are done.

```
bench(Host, Port, N) ->
    P = self(),
    Receiver = spawn(fun()->receiver(N,P) end),
    Start = erlang:system_time(micro_seconds),
    run(N, Receiver, Host, Port),
    receive
        all_done -> ok
    end,
    Finish = erlang:system_time(micro_seconds),
    Finish - Start.
```

As Figure 1 shows, when the number of requests increases, the improved rudy responses faster significantly.

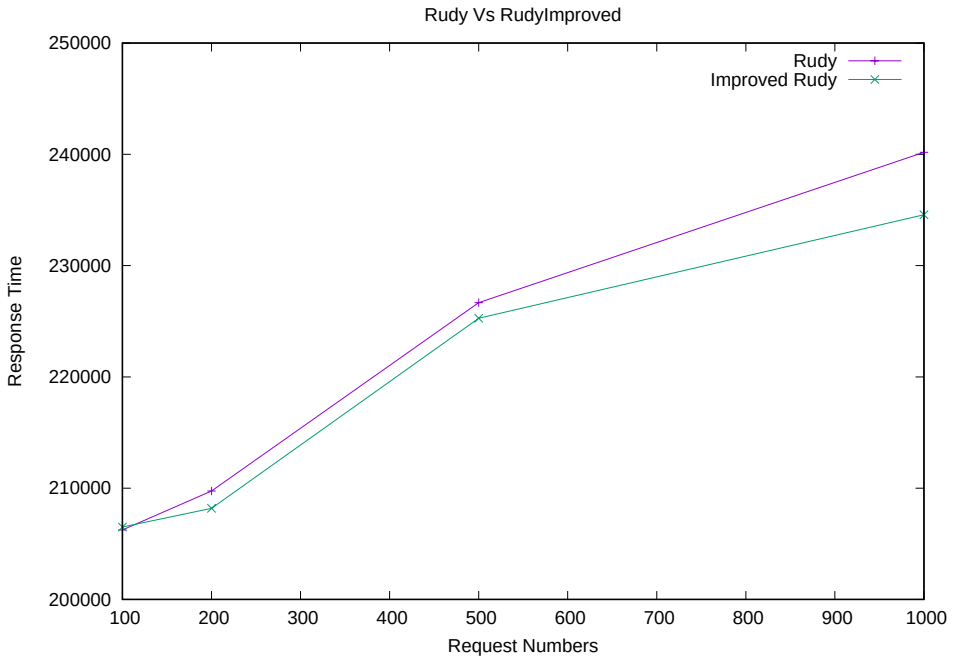

Figure 1: Response Time

3.2 Deliver Files

Use `string:tokens(Str, " ")` to separate the file path and read the file. Response with a reply header including status, type, size and show the file data.

```
get_file(Path) ->
    case file:read_file(Path) of
        {ok, Data} -> Response = response(Data);
        ...

response(Data) ->
    io_lib:fwrite(
    "HTTP/1.0 ~s\n
    Content-Type: text/html\n
    Content-Length: ~p\n\n~s",
    [Status, size(Data), Data])
    ...
```

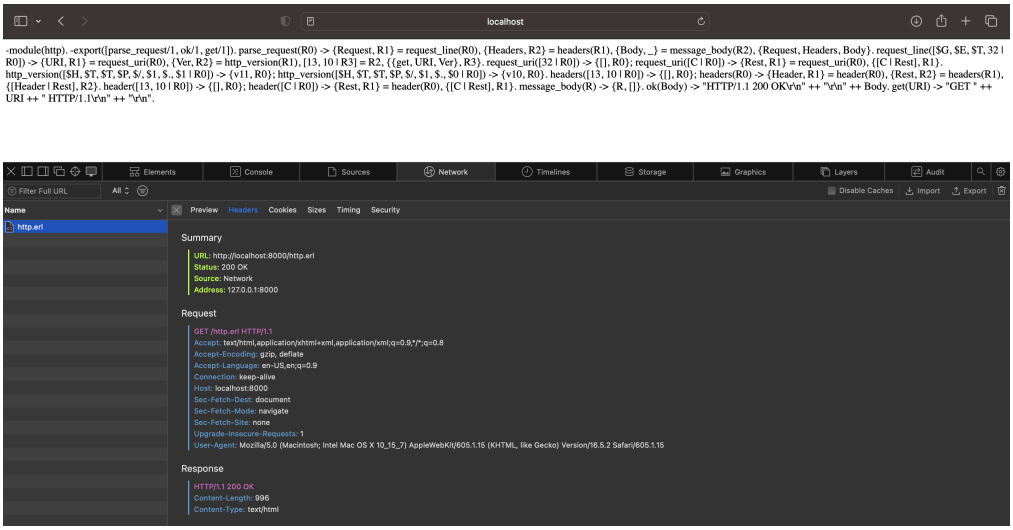

Figure 2: Deliver File

4 Conclusions

Rudy works well and some improvements have been implemented.

I experienced to build a server from scratch and knew the structure of a server process. I also learned how to do some Erlang programming, which is useful for me to study distributed systems.