HW1 : Rudy - a small web server

[redacted]

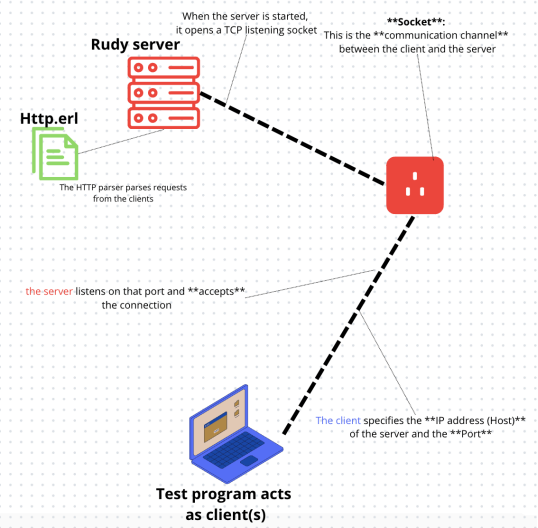
September 5, 2024

1 Introduction

*We worked through the development and testing of a **rudimentary HTTP server** using Erlang. The task involved implementing a server that can handle incoming HTTP requests, simulate server-side delays, and benchmark its performance.*

The main topic related to distributed systems covered in this seminar is building a concurrent HTTP server that can handle multiple client requests at one time using Erlang. This is important for distributed systems as it allows servers to manage high volumes of client requests without crashing, ensuring possibilities for scalability. Understanding how to handle concurrency, network communication, and performance which is essential for creating scalable and reliable distributed systems.

Very basic visual representation of the structure of the task and its components.



2 Main problems and solutions

When testing the functionality of my server, I configured the server to listen on my WLAN IP address instead of localhost, even though I was accessing it from the same machine. This allowed me to simulate the connection as if it were coming from another machine.

While I didn’t have access to another machine with Erlang, I was able to retrieve the page “http://192.168.1.31:8088/sofie” from a web browser on a different device, which validated the server's response.

Encountered errors: I encountered one issue when trying to start a new port reusing a prior port nr. I encountered the error `eaddrinuse`, indicating that the port was already in use. This issue occurred since the port hadn't been properly released. To resolve this: I made sure to properly stop the server using the `rudy:stop/0` function, which ensured the port was freed.

3 Evaluation

When doing experiments to test the servers request handling, I accessed the server from the different clients, testing a 10ms delay and a 40ms delay with first a single client sending 100 requests, and then with 3 clients sending 100 requests each simultaneously. See results:

40ms delay

```
  timer:sleep(40),  % Simulate a 40ms delay
```

When sending requests from one client at a time, the amount of ms it takes to run the 100 requests is:

```
5> test:bench("192.168.68.55", 8088).
4292623
```

$$\frac{100\,\text{requests}}{4.292623\,\text{seconds}} \approx 23.30\,\text{requests/second}$$

Running from three clients at a time, the amount of ms i takes to run 100 requests / client is:

$$\frac{100\,\text{requests}}{10.321831\,\text{seconds}} \approx 9.69\,\text{requests/second}$$

$$\frac{100\,\text{requests}}{11.105069\,\text{seconds}} \approx 9.00\,\text{requests/second}$$

$$\frac{100\,\text{requests}}{10.661644\,\text{seconds}} \approx 9.38\,\text{requests/second}$$

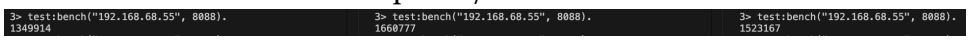
10ms delay

```
  timer:sleep(10),  % Simulate a 10ms delay
```

When sending requests from one client at a time, the amount of ms it takes to run the 100 requests is:

$$\frac{100\,\text{requests}}{1.216112\,\text{seconds}} \approx 82.23\,\text{requests/second}$$

Running from three clients at a time, the amount of ms i takes to run 100 requests / client is:



$$\frac{100\,\text{requests}}{1.349914\,\text{seconds}} \approx 74.08\,\text{requests/second}$$

$$\frac{100\,\text{requests}}{1.660777\,\text{seconds}} \approx 60.20\,\text{requests/second}$$

$$\frac{100\,\text{requests}}{1.523167\,\text{seconds}} \approx 65.65\,\text{requests/second}$$

4 Conclusions

When we tested the server with multiple clients sending requests at the same time, the performance for each individual client dropped, but the server could still handle the overall load effectively. For example:

- With a 40ms delay and only one client, the server processed 23.3 requests per second. This is close to the maximum the server can handle with such a delay.
- When three clients sent requests at the same time, the server processed around 9–10 requests per second for each client. The total throughput for the server was around 28 requests per second.

This means that the server is capable of handling multiple clients at the same time, but the requests per second per client decrease as more clients connect.