HW1 Report, Rudy: A Small Web Server

David Fischer

September 1, 2025

1 Introduction

As part of this report a HTTP/1.1 server partially implementing the RFC 2616[1] specification was created in the Erlang programming language. The server is capable of correctly parsing and responding to HTTP requests, handling multiple clients concurrently, and acting as a file server with functional encoding. xxx todo how related to dist system, why important

2 Main problems and solutions

2.1 Scope

As the HTTP/1.1 specification is quite broad building up scope creep beyond the goals of the original assignment was a concern. Work was limited to objectives specifically mentioned in the assignment description. At time of submission the server currently supports the following features:

- Listening on a specified port and delegating requests to one or many handler processes
- Parsing HTTP *GET* and *POST* requests (URI, request headers, and body)
- Responding with one of three HTTP status codes: 200 OK, 404 Not Found, or 500 Internal Server Error
- Serving static files from the working directory and its subdirectories with a limited number of supported media types
- Support for the Accept-Enconding request header, specifically gzip

2.2 Erlang

todo

[1] `https://www.ietf.org/rfc/rfc2616.txt`

3 Evaluation

Testing the baseline implementation of *Rudy*, including the artificial 40ms delay, with the given benchmark program measures 100 requests in 4.916 seconds, or ≈20 requests per second. Since the benchmark runs sequentially, the regular parsing overhead can be calculated as 9,16 ms meaning the artificial delay is over four times the length of a regular request.

Additionally, while the artificial delay isn't noticeable with a single instance of the benchmark running, starting a second run from another machine doubles the request time while the benchmarks overlap as can be seen in Figure 1.

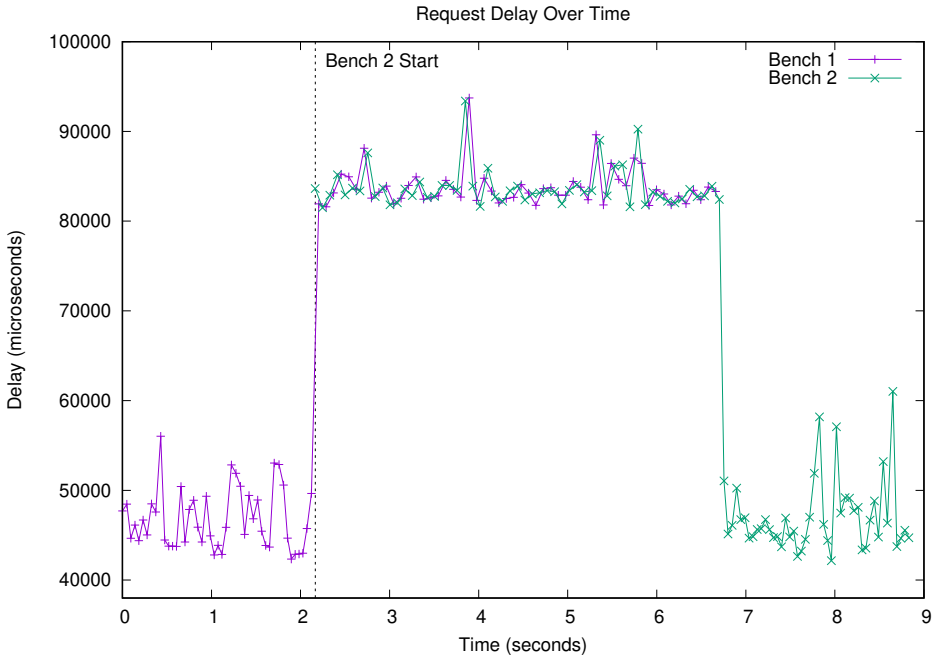

Figure 1: Single Process Handler

This outcome is expected, as a single handler process will always be blocked while already responding to a request during which all new requests are waiting to be picked up. To solve this and increase throughput, Erlangs support for having multiple processes listen to the same socket was utilized. Figure 2 shows the same two sequential benchmarks running against an instance of *Rudy* with two handler processes running, resulting in no spike in response delay.

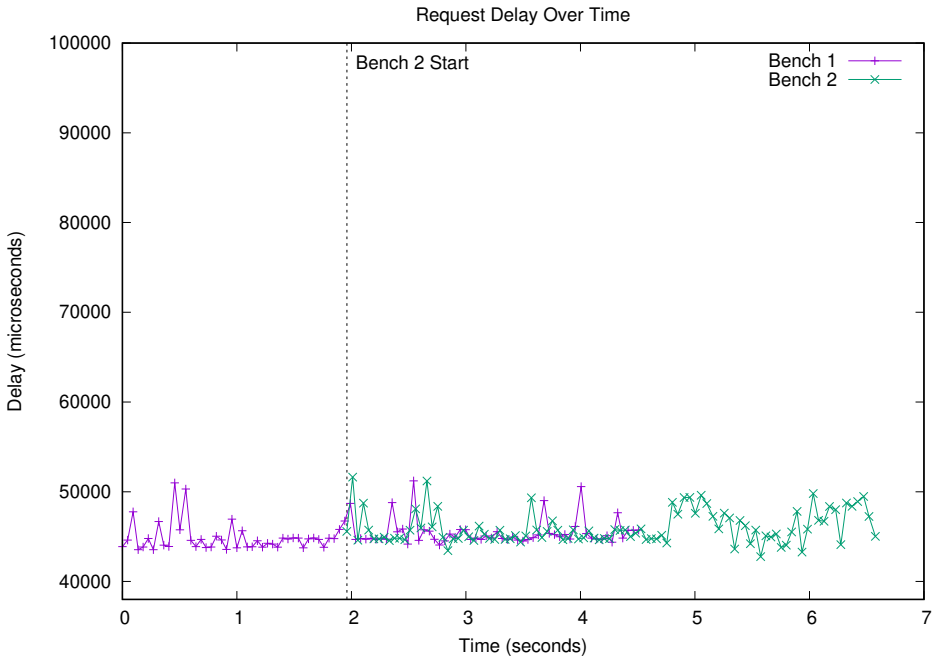

Figure 2: Multi-Process Handler

3.1 File Server Functionality

The implementation of *Rudy* supports serving gzip compressed files including a limited selection of MIME types. xxx

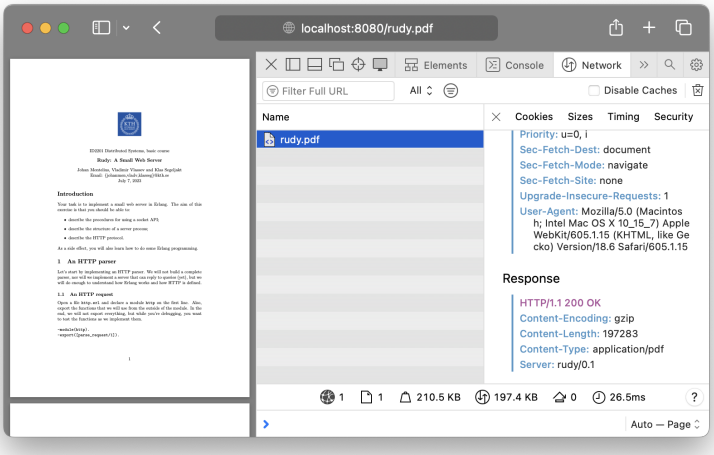

Figure 3: *Rudy* serving a gzipped PDF file to a browser

1. one
2. two

4 Conclusions

xxx What have you learnt from the problem presented? Was it useful?