Report HW1: Rudy - a small web server

Artem Sliusarenko

September 13, 2023

1 Introduction

Implemented small web server, rudy, to benchmark it using various strategies and discuss the results.

2 Main problems and solutions

Did not encounter any specific problems.

3 Evaluation

1. Test server response time (client on the same machine).

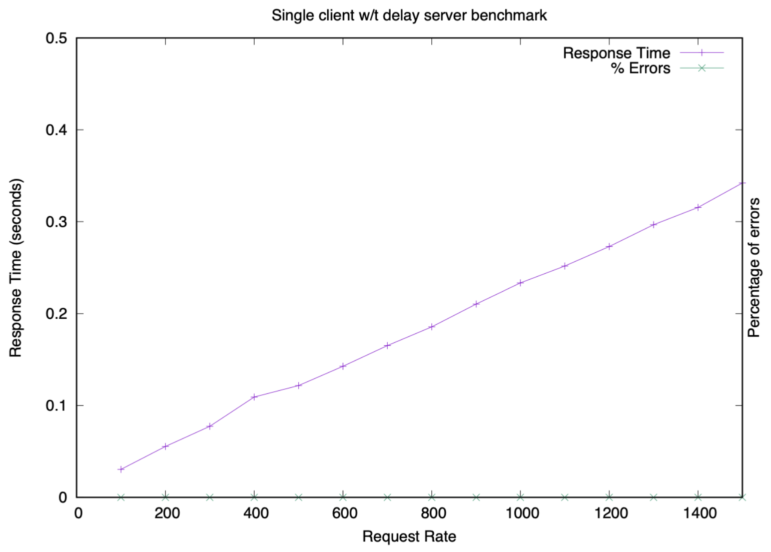

Figure 1: Server benchmark: client count (1) + without delay

By looking at the graph we can observe that as number of requests grows so does the time it takes to process them. Now we can calculate average request response time per single request: 30502 microseconds / 100 requests made = 305 microseconds per request = 0.000305 seconds. To calculate average number of RPS (requests per second) we can serve: 1 (second) / 0.000305 (single request in seconds) = 3279 RPS

2. Test server response time with artificial delay in the handling of the request (client on the same machine).

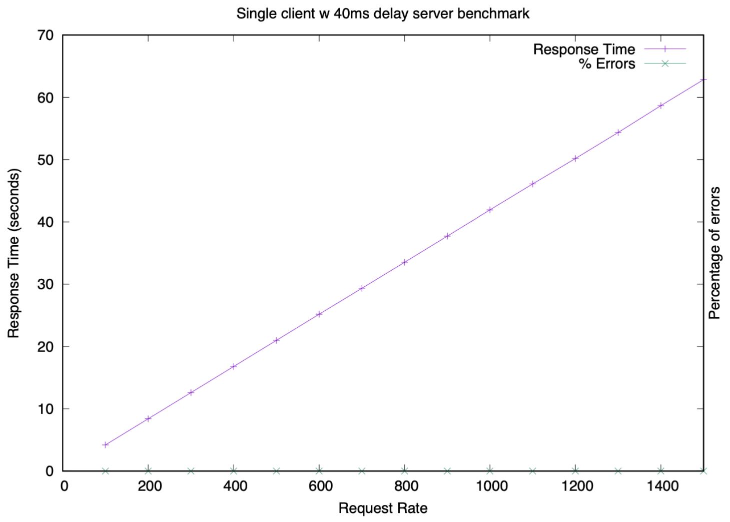

Figure 2: Server benchmark: client count (1) + with 40ms delay

Just as in section evaluation 1 we can observe that as number of requests grows so does the time it takes to process them. It is clear that delay does not disappear in the parsing overhead. It obviously slows down the parsing significantly which is clearly visible in the graph. Now we can calculate average request response time per single request: 4197233 microseconds / 100 requests made = 41972 microseconds per request = 0.041972 seconds. To calculate number of RPS (requests per second) we can serve: 1 (second) / 0.041972 (single request in seconds) = 24 RPS

3. Test server response time (client on a different machine).

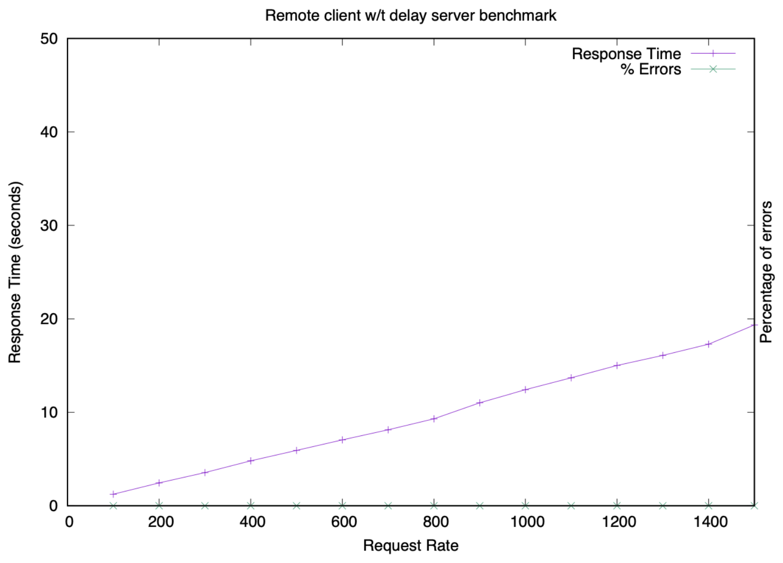

Figure 3: Server benchmark: client count (1) + without delay

Running the server benchmark from different machine shows that our parsing overhead has grown.

4. Test server response time with artificial delay in the handling of the request (client on a different machine).

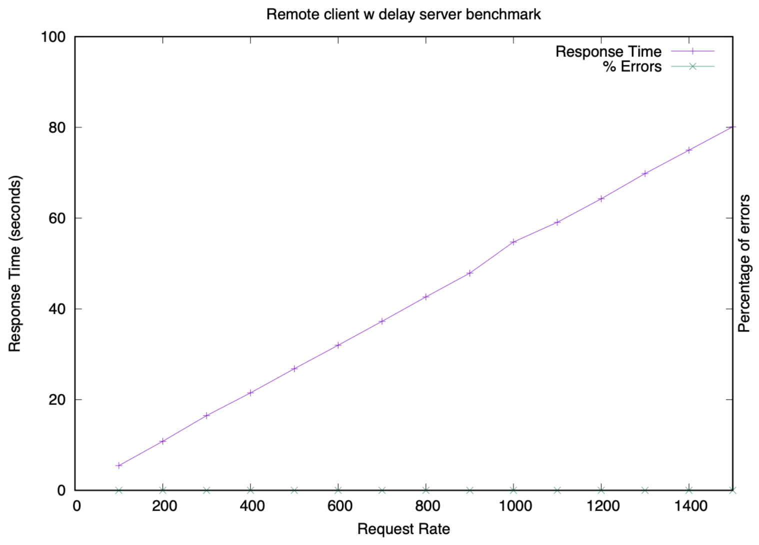

Figure 4: Server benchmark: client count (1) + with 40ms delay

Similarly to section 4 we observe rise in parsing overhead.

5. Running benchmark (with artificial request handling delay) on several machines simultaneously. I ran benchmark 3 times from both same and different machine, from where the server is running, one at a time to have a reference for a simultaneous run (Case 1 and 2). After, I ran benchmark simultaneously on both machines (Case 3 and 4). We can clearly observe that the amount of time it took server to respond to two clients simultaneously is twice as much as responding to one client request at a time.

Case	run 1	run 2	run 3
Case 1	4197070	4199507	4197814
Case 2	5581035	5441971	5358417
Case 3	8119909	8120383	7799495
Case 4	8200232	8211727	8336694

Table 1: Simultaneous run results in microseconds

4 Conclusions

We have learned that single erlang process can't process requests concurrently, therefore every request has to wait for the server to finish parsing previous request to be picked up. Such approach will prove to be inefficient in scenario with unknown number of requests. We have also learned that sending requests to remote hosts from same or different machines affects response time. Parsing overhead of the requests from different machine than the server is running on is higher than from same machine. This experiment was useful, because now we know the cons of our implementation and we can work on improving it. Throughput is one of the most important properties of the server we should look into.