Report : Rudy a web server

Vasigaran Senthilkumar

September 15, 2021

1 Introduction

The development of a small web server, Rudy using Erlang is discussed in the report. Various modules developed in the code include the http parser for parsing http get requests and the server module which uses in-built gen tcp module for socket communication. The report also provides in sights on improving the throughput of server after various performance evaluations. The effort has covered basics of Erlang functional programming, Usage of TCP Socket API's, Understanding the Process of a server, concurrent programming. These topics hold the key to get started with fundamentals of distributed programming.

2 Main problems and solutions

2.1 HTTP Parsing using tail recursion

In the http parser module, for parsing the requests, requesturi/2, a tail recursive function is implemented with an extra temporary variable. It aims to eliminate the stacking of operation by reducing them as they happen.

```
request_uri([C|R0],A) ->
request_uri(R0,[C|A]).
```

2.2 Increasing throughput

When Rudy runs as single process the throughput is very low. Hence Multiple handler processes can be run in parallel to increase the throughput. parallelhandlers/2 function is used to spawn handler/1 function in parallel. Now all the handlers will simultaneously wait for incoming connection from a single socket. When a client connection is made, request is sent to handler.

```
if N==0->
   ok;
   true->
   spawn_link(fun()->handler(Listen,Pid) end),

   parallel_handlers(Listen,N-1,Pid)
```

2.3 Spawn link

Spawning parallel handlers created exceptions while closing the server. The reason behind was, only the caller process was getting killed when stopping the server. The created parallel process resulted in exceptions blocks inside the handler function call. In order to avoid this scenario, spawn link function had to be used instead of spawn inside the parallelhandler/2 function block. In a spawn link function, a link is created between the calling process and the new process, atomically. If either process is killed, the rest will also be killed.

3 Evaluation

While the server was up and running in one machine, another machine was able to access the server with IP and Port, without crashing of the server. A benchmark program was run to generate requests and measure the time it takes to receive the requests.

3.1 Single client thread sent 100 requests to Rudy

In the test case, single client thread sent 100 requests to Rudy. For various number of handler processes, the results were measured for the time taken to respond to the requests. The results are plotted in figure 1.

3.2 4 client threads sent multiple number of requests each to Rudy

In the test case, 4 client threads sent multiple number of requests to Rudy. For various number of handler processes, the results were measured for the time taken to respond to the requests. The results are plotted in figure 2.

3.3 Multiple client requests with 200ms delay

In the test case, multiple client threads with multiple request each was tested with a delay of 200 ms in the server code. The results are plotted in figure 3.

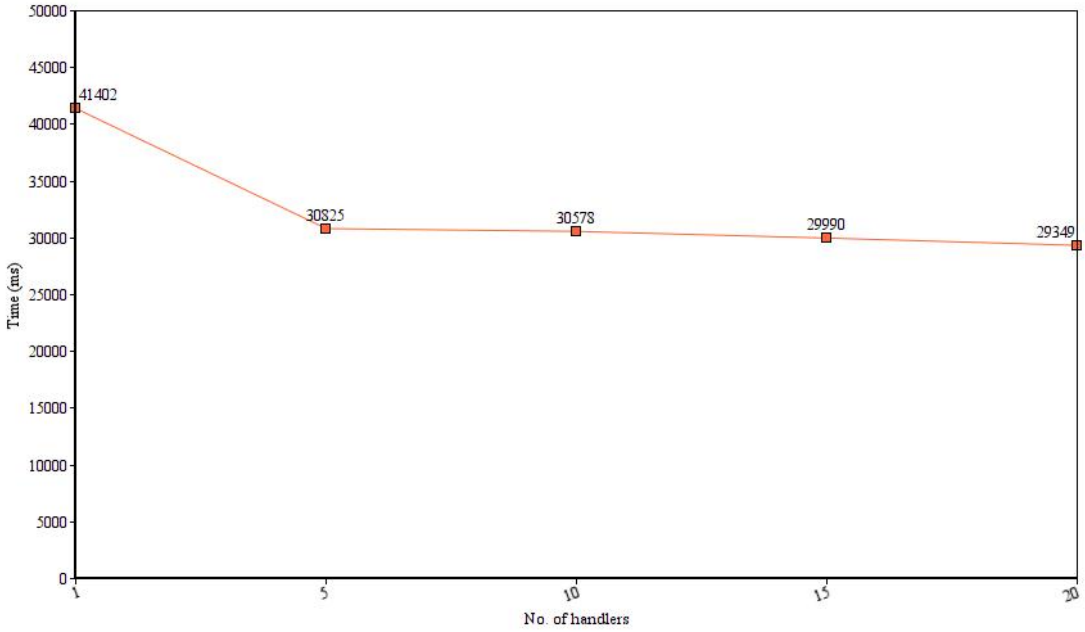

Figure 1: Single client thread sent 100 requests

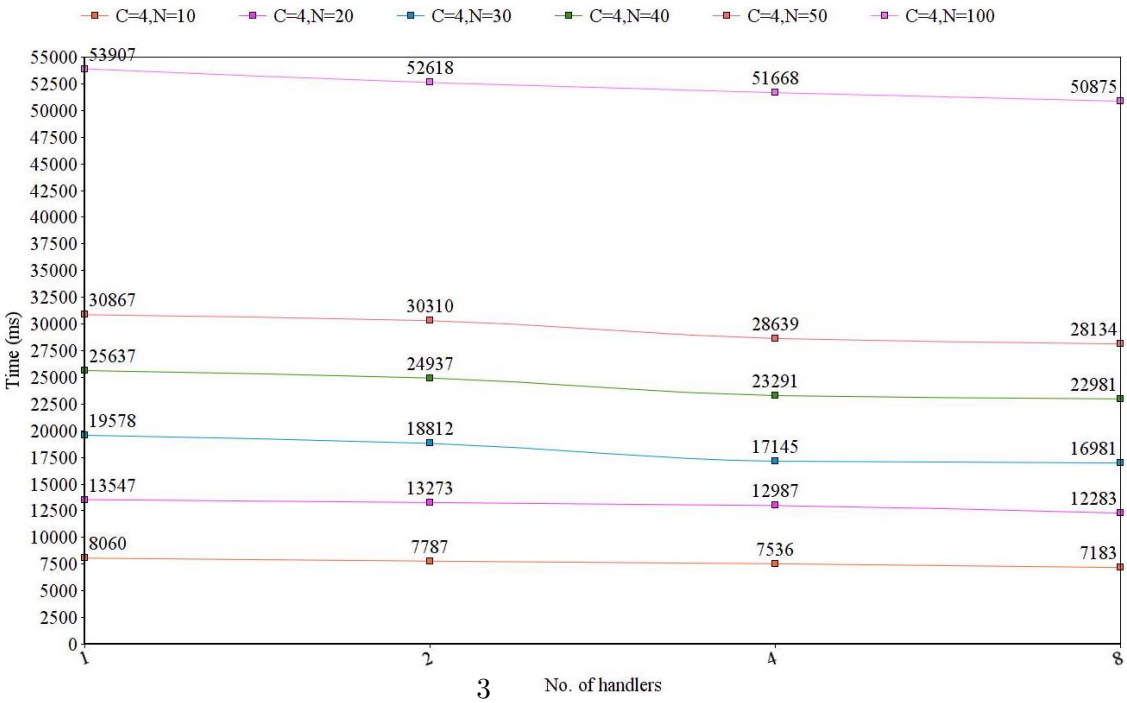

Figure 2: 4 client threads sent multiple number of requests
C - No of Client threads, N - No of requests

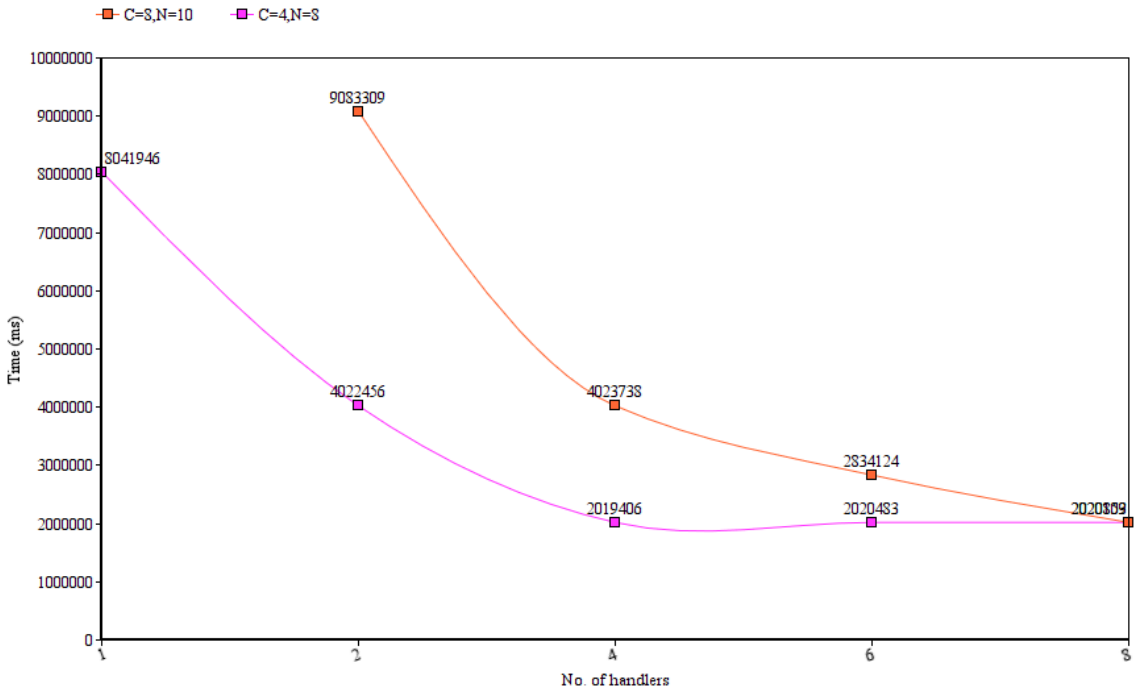

Figure 3: Multiple client requests with 200ms delay

3.4 Number of requests received per second

Condition	time(microseconds)	Requests per second
with 40ms delay	4197431	24
without any delay	41402	2300

Table 1: Requests received per second

4 Conclusions

The results from figure 1 and figure 2 show only slight improvement in the throughput through parallel processing. A 200ms sleep was introduced in the server and tested. The increase in the throughput was significant as time taken for the processes reduced with number of handlers from figure 3. The tests were done on a mac os. So 200ms delay had to be introduced as the mac os sometimes interferes when running benchmarks. Thus throughput of server has been increased by letting each request be handled concurrently.