Report 1: HTTP server

Simon Cathébras

September 6, 2012

1 Introduction

During the preparation of this seminar, we did an implementation of an HTTP server in Erlang. This server was made following instructions provided in the subject. During this preliminary study, we studied HTTP protocol, the use of a socket API, and how to build a parser in Erlang.

This seminar has highlighted the structure of a node as a server in a the distributed systems' field. This is actually the basis of all distributed systems, that's why studying an example of node is really important.

2 Main problems and solutions

The problematic was not very difficult to understand. What was somehow difficult was Erlang itself. The main problem was when we had finished to implement the Rudy server. When we ran it, there was an unexplainable error. We solved this in reading the part of Erlang documentation dealing with the error report. We found that the stack of functions call was given in the error report. So far, it was easy to find that `http.erl` was not compiled.

Except this erlang's "trick", another main problem was that we could not directly run the benchmark on the server. Because, we did have only one machine. To get rid of this problem, we inspired ourself in the *Hello Erlang* seminar. We started two nodes in the same machine. One of them was `client`, which executed the bench program, and the other was `server`, who ran the HTTP server. Once again, Erlang made things easy: we just had to launch the benchmark with the following arguments: `bench('127.0.0.1',8080)`, and it worked.

3 Evaluation

In this section, we will report Rudy's behaviour concerning requests' computing. First, here are the results of 3 benchmark's run.

Time elapsed	$request.s^{-1}$
4341181	23,03
4293843	23,28
4396452	22,75

Table 1: Number of requests by seconds

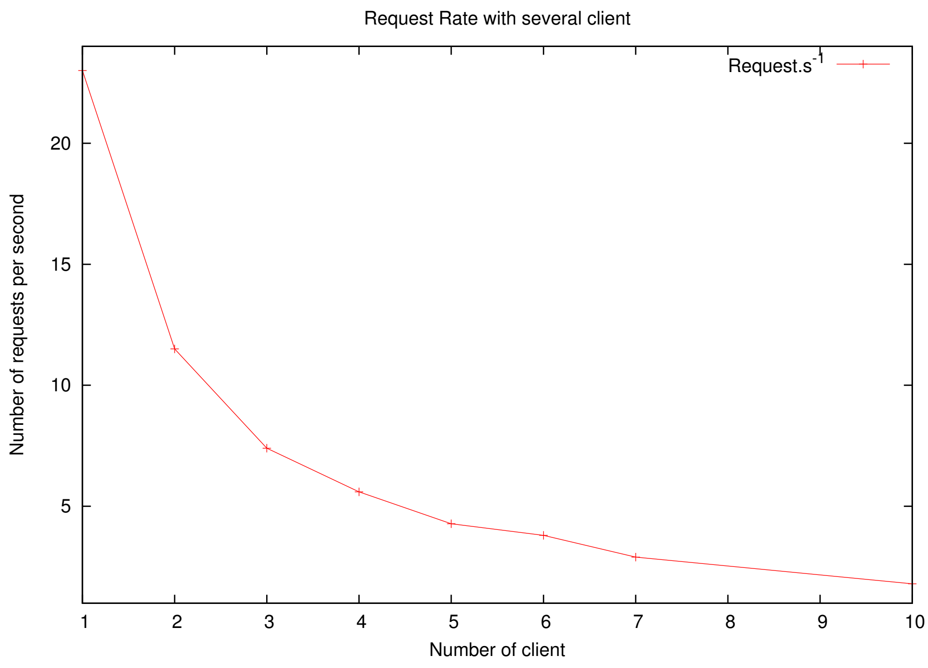

Figure 1: Requests per second depending the number of clients

Observing those results, we can assume that our server can compute approximately 23 requests per seconds. In addition, we can observe that the artificial delay we added in the handling of each request is significant when we compare it to the parsing time. We are computing actually 946 requests by seconds if we do not add this delay. Now, let's see what append if we increase the number of clients.

To test Rudy's behaviour with several clients, we modified a little bit the `test.erl` file. We have add a **start** `(Host, Port, NbClient)` function, which starts `NbClient` process of `bench(Host, Port)`.

In this graphic, we displayed the mean of requests rate from the client side. Observing this graphic, we can assume that, right now, Rudy is not able to handle properly several concurrent connexions.

4 Conclusions

First, we learned a little about how to build a simple parser in a functional language like Erlang. Something which is actually not obvious. In addition, in this preliminary work, we discovered that one of the main issues of distributed systems is the handling of concurrency. which give us something to start with in the next seminar.