Faculty of Computers, Informatics and Microelectronics

Technical University of Moldova

PAD

Laboratory work # 3

Study of HTTP protocol used in distributed systemns

Authors:
Petru Negrei
Victor Vasilica

Supervisor:
D. Ciorba

November 2014

1 Introduction

1.1 Topic

Study of HTTP protocol used in distributed systemns. Implementation of connection between client and server using HTTP methods.

1.2 Objective

Implementation of the thread safe system of processing of data.

1.3 Generic requirements

1.3.1 Report

Report will contain a short description of work done, and will present necesary information about tools, algorithms used or studied.

2 Structure

Below you can see the structure of the application, there are represented the classes used and their variables and methods. The main are client and server which the helper class *Http Handler* that server the purpose of sending the http requests to the *Main Server.*

The client will be responsible for sending *GET* request to the *Main Server*, then the *Main Server* will find the node with max connections and return the information to the *Client* Then the *Client* will establish a TCP connection necessary node. After it receive all necesary data, this data will be analyzed and shown to the user.

In order to allow multiple requests from server nodes and client, each connection will be handled in different thread and in a safe way.

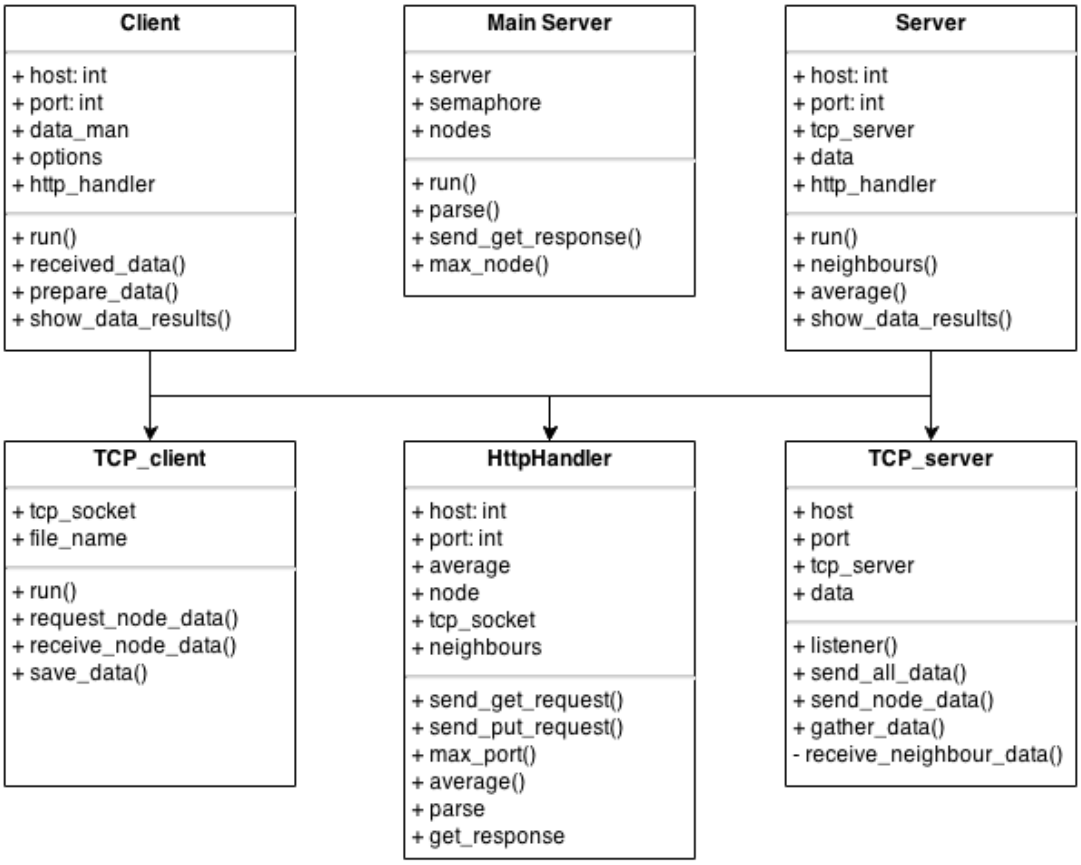

Fig. 1 Class diagram

3 Implementation

3.1 Stored data

Like in the previous laboratory work all information of servers is stored in one single *json* file (*data.json*) in order to reduce redundancy in real project each server will have separate file that will contain its specific data.

```
{
  "10000": {
    "neigthbours": [10001],
    "employers":[ ... ]
  },
  # ...
}
```

```
# ...
   "employers":[
      {"firstName":"John", "lastName":"Doe", "salary": 150, "department":
         "Marketing"},
      ...
```

```
        {"firstName":"Alex", "lastName":"Carturari", "salary": 478, "department":
           "Testing"}
    ]
# ...
```

3.2 Client Side

The client part of the application is composed of four main file that together provide functionality necesary to assure the request to server, receive, save, parse and show the response in form of the list of emplyoers from server with the maximum nodes and its neighbours.

3.2.1 HTTP handler

We don't use the UDP helper like in the previous laboratory work, instead we send the *GET* request to the *Main Server* and receive the response that will contain the port and average of the node with maximum connections.

- *initialize* - initialize all necesary variables and a TCP socket.
- *send_get_request* - send *GET* request and receives the response from *Main Server*.
- *send_put_request* - send *PUT* request in the case of server node.
- *max_port* - returns the node with max connections.
- *average* - computes the total average of all nodes.

```
class HttpHandler

  attr_accessor :host, :node, :tcp_socket, :port, :neighbors, :average

  def initialize host, port, neighbors=0, average=0
    @host = host
    @neighbors = neighbors
    @average = average
    @node = {}
    @port = port
    @tcp_socket = TCPSocket.new(host, 8000)
  end

  def send_get_request
    tcp_socket.puts "GET / HTTP/1.0\r\n\r\n"
    parse get_response
    tcp_socket.close
  end

  def send_put_request
    response = "#{port}:#{neighbors}:#{@average}"
    header = [ "PUT / HTTP/1.0",
             "HOST: #{host}",
```

```
                "Content-Type: text/plain",
                "Content-Length: #{response.bytesize}",
                "Connection: Keep-Alive" ].join "\r\n"

      tcp_socket.puts header + "\r\n\r\n" + response
      p "sent put request"
      tcp_socket.close
    end

    def max_port
      @node["port"]
    end

    def average
      @node["average"]
    end

    private

    def parse response
      @node = JSON.parse response
    end

    def get_response
     response = ''
      while line = tcp_socket.recv(1024)
        response += line
        line =~ /%--%/ ? break : response += line
      end
      headers,body = response.split(/\r\n\r\n/)
      body.gsub "%--%", ""
    end

  end
```

3.2.2 TCP protocol Data manipulation

It were implemented in the previous laboratory work.

3.3 Main Server

The most important function of the main server is to receive different request, and depending on type and parameters of request to perform certain actions.

The *run* function is the function that receives requests from nodes or client in a thread safe way and depending of the type of request it acess different methods.

```
  # ...
    def run
      loop do
       Thread.start(server.accept) do |client|
```

```
        @semaphore.synchronize do
          request = client.gets
          if request =~ /^GET.*/
            send_get_response(client)
          else
            request += client.read
            @nodes << request
          end
          client.close
        end
      end
    end
  end
  # ...
```

3.3.1 Main

The main class *Client* contains all methods that are necesary to start and perform necesary work, it also has UDP and TCP functionality by using the *classes* with the same name. Below there is a short description of methods that it has, for the implementation you can see the full code.

- In the *contructor* we save provided options and setup http handler.
- *receive_data* - we setup a tcp server, send requests to servers and save response to a file.
- *prepare_data* - read the json file and initiate the data manipulation class.
- *show_data_results* - show data received from server, based on option from console.

```
class Client
  # ...
  def initialize options
    @options = options
    @http_client = HttpHandler.new HOST, 8000
  end

  def run
    # ...
  end

  private

    def receive_data
      tcp_client = TcpClient.new(HOST, FILE_NAME, http_client.max_port)
      tcp_client.run
    end

    def prepare_data
      data = read_json_file
      @data_man = DataManipulation.new(data, http_client.average)
    end
```

```
    def show_data_results
      case options[:show]
      when "default" then data_man.show_all
      when "filtered" then data_man.show_filtered
      else
        puts "Invalid option"
      end
    end

    def read_json_file
      JSON.parse File.read(FILE_NAME)
    end

end

# input arguments from terminal
options = {}
options[:show] = "default"
OptionParser.new do |opts|
  opts.banner = "Usage: ruby client.rb [options]\n\n"
  opts.on("-s", "--show [SHOW]", "show data by method") do |s|
    options[:show] = s || "default"
  end
end.parse!
```

One of its most important methods is 'run' method, here we first send get request to the *Main Server* and then we establish a tcp connection with the node with maximum neighboars, receiving necesary information from it, parse it and show it in the console.

```
# ...
  def run
    http_client.send_get_request
    receive_data
    prepare_data
    show_data_results
  end
  # ...
```

The client actions are started with the following line, it receive the actions that need to be applied on the data.

```
Client.new(options).run
```

3.4 Servers

The *Main server* part was done by Vasilica Victor but a part of functionality of thread safe implementation was done working together, (pair programming) in order to assure the integrity of the data.

4 Conclusion

In this laboratory work we built a distributed system that different http requests to send information from nodes to main server and from *main server* to *client*, each request doing its own predefined task.
We build a helper class that handles all required requests *GET* and *PUT*, and receives the responses from the *Main Server*. Depending on the request *Main Server* will perform different action, in the *GET* case it will return the information about node with maximum connection, when *PUT* it will add the information in an array.
Thus we learned how to build a small distributed system with a collection of objects using different http requests.

Link to Repository: `https://gitlab.ati.utm.md/petru.negrei/lab3/tree/wip/petru`

5 References

- Ruby Socket `http://www.ruby-doc.org/stdlib-1.9.3/libdoc/socket/rdoc/Socket.html`
- Ruby TCP Socket `http://www.ruby-doc.org/stdlib-1.9.3/libdoc/socket/rdoc/TCPSocket.html`
- Ruby Mutex `http://wwwi.ruby-doc.org/core-2.1.4/Mutex.html`
- Ruby JSON `http://www.ruby-doc.org/stdlib-2.0.0/libdoc/json/rdoc/JSON.html`
- Ruby OptionParser `http://ruby-doc.org/stdlib-2.1.0/libdoc/optparse/rdoc/OptionParser.html`