Faculty of Computers, Informatics and Microelectronics

Technical University of Moldova

PAD

Laboratory work # 6

Building a Proxy with Shared Databases.

Authors:
Petru Negrei
Victor Vasilica

Supervisor:
D. Ciorba

December 2014

1 Introduction

1.1 Topic

Build a Proxy that will link the client with the main server and assure the integrity of data among servers.

1.2 Generic requirements

1.2.1 Task

Develop a system of distributed heterogeneous data, centralized in one node type warehousing.

1.2.2 Report

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

2 Structure

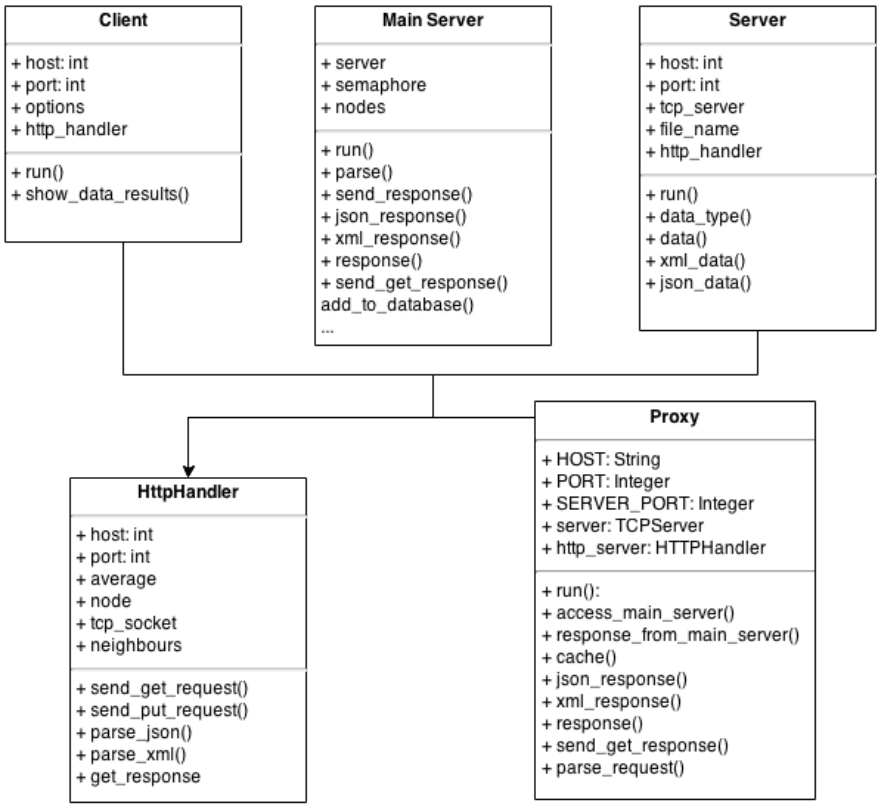

Fig. 1 Class diagram

Above you can see the structure of the application, (the structure remained the same) there are represented the classes used and their variables and methods. The application is composed of three

main clasess *Client* and *Server* and *Main Server*, and now also the *Proxy* class.

The *Client*, *Server*, *Proxy* classes use the *HttpHandler* helper class in order to send the necesary requests to the *Main Server* and also recieve the response from it.

The requests are handled and analyzed on *Main Server* in different threads and saved on a single database, the database doesn't duplicate request, if it receives a data that already that is in database it will update the fields of it.

3 Implementation

3.1 Proxy

Here I will explain the structure of the *Proxy* class. The description for other class in the system you can find in the previous laboratory work.
The most important function of the *Proxy* is to receive *GET* request from client, and to pass this requests to the server part. Then it receives the response from main server and sends it back to the client. At this level it is also implemented caching of the request, which was implemented toghether with Victor.
The *run* function is the function that receives requests from client in a thread safe way and depending if the requsts was already in the cache to send the cached response or to access main server and cached and send response from it to the client.

```
# ...
 def run
   loop do
    Thread.start(server.accept) do |client|
      http_header = client.gets("\r\n\r\n")

      unless redis.exists(http_header)
        @http_server = HttpHandler.new HOST, SERVER_PORT
        access_main_server(http_header)
      end

      send_get_response(client, redis.get(http_header))
      client.close
    end
   end
  end
# ...
```

Below I will describe each function utility.

- *initialize* - create a TCPServer to listen to the clients request, and also *Redis* object to store the cache of the requests.
- *access_main_server* - to send request to the main server and call the *cache* method.
- *response_from_main_server* - contains the response from server in different formats.

- *cache* - save the cache key and value with a given interval of time.
- *json_response* - returns the json response.
- *xml_response* - returns the xml response.
- *response* - the method that returns the data depending of the user request.
- *send_get_response* - sends the response in the right format with the right data.
- *parse_request* - return the type and url from the client request.

```
# ...
class Proxy
  HOST = 'localhost'
  PORT = 8001
  SERVER_PORT = 8000

  attr_reader :server, :http_server, :redis

  def initialize
    @server = TCPServer.new HOST, PORT
    @redis = Redis.new
    @redis.flushall
  end

# ...

def access_main_server http_header
  puts "new data"
  resp = response_from_main_server(http_header)
  cache(http_header, resp, 60)
  resp
end

def response_from_main_server http_header
  url, _, type = parse_request(http_header)
  http_server.send_get_request("/" << url, type)
  (type == 'json') ? json_response : xml_response
end

def cache key, value, expires_at
  redis.set(key, value)
  redis.expire(key, expires_at)
end

# ...
```

3.2 Server Side

The server part was done by Vasilica Victor and the caching functionality was implemented together.

4 Conclusion

After implementing this laboratory work, I learn how to build a Proxy that will allow client to communicate with different main Servers, and to receive response from them. I implemented a caching system that will optimize the requests received and send the response faster without accessing the server.
Link to Repository: `https://gitlab.ati.utm.md/petru.negrei/lab6`

5 References

- Redis `http://redis.io/`
- 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`