UMONS
Université de Mons

Networks

Proxy Cache Server

Faculté des Sciences

Networks - Project 1 : Proxy cache server
Baptiste Calin
Frédéric Jamar

Networks – Project 1
Implementation of a proxy cache server

Issue:

As part of the networking course, a group of two people was asked to create a proxy cache server in Java. As a reminder, a proxy server is an intermediate agent that a HTTP client, like a browser, can contact when it needs to request a web site. The caching feature reduces traffic by saving web files for a defined time and returns them to the client as needed. This occurs only if all downloaded data were not modified since the server downloaded them. If not, the server executes a request to the distant web server and saves again files locally.

Requirements and technical constraints:

[REQ_01]
The proxy cache server shall be started without IDE.

[REQ_02]
The user shall be able to configure the listening port of the proxy cache server.

[REQ_03]
The proxy cache server shall support client HTTP versions such as 1.0 and 1.1, HTTPS is out of scope.

[REQ_04]
Only the GET method and static websites shall be compatible.

The proxy cache server shall support:

[REQ_05]
- Get and decode a HTTP request from a client
- [REQ_06]
- If the requested web page is available locally, the server shall transmit it to the client.
  [REQ_07]
- If the requested web page is not available locally, the server shall forward the request to the web server, get and decode the response, saving files locally and forward the response to the client. This feature is in reality a cache system with persistence.

[REQ_08]
The server is able to handle multiples client connections in the same time.

[REQ_09]

The server implements a request/response pipelining system to optimize the response for the client.

Note: The solution shall be implemented only with socket handling and low-level classes.

Tools and architecture:

The IDE chosen is IntelliJ. IntelliJ IDEA is the most advanced and complete IDE to develop an application in Java. Moreover, the community version is free for a non-commercial use.

The project is hosted on the free and open source distributed version control system GitHub public under https://github.com/bacalin1982/proxy .

To implement the solution, the Java SE Development Kit 8 is used.

Implemented features

Client-Proxy communication

This feature is at first handled by the Server Class who listen by thread new clients. Then, for each client, in the Client Class, the request is captured by input stream, decoded, logged and transmitted to the cache evaluation. If the cache evaluation is negative or positive, the Client Class is able to give the correct HTTP response to the client.

This communication is adaptive to the client request. Indeed, if the client asks for HTTP 1.0 or 1.1, the server is able to answer with the same protocol version.

Proxy-Server communication

If the cache evaluation is negative, the server is able to forward the client's request to the distant web server and handle the answer to make new cache files and forward everything to the client. See, in Client class the begin of the Proxy-Server communication. About the request/response from the web server, see the run method in Request class:

```
[...]
HttpRequest httpRequest = HttpRequestBuilder.makeHttpRequest(this.clientRequest);
//get response from server
String host = httpRequest.getHost();
Socket serverSocket = new Socket(host, 80);
System.out.println(Constants._I+ Constants.WEB_SERVER_CON.replace("%1", host));
int nbrWaiting = 10;
S:
while (!serverSocket.isClosed()) {
    try {
[...]
```

Competing connections

This feature is implemented by accepting clients with thread method in Server Class. The server is able to handle multiple connections simultaneously:

```
[...]
/*
* [REQ_08]
* The server is able to handle multiples client connections in the same time.
* */
// Server is waiting infinitely a new Thread client with its opened socket.
while(true){
    // Each time a client is accepted, a new client is defined.
    new Client(this.serverSocket.accept()).start();
}
[...]
```

Pipelining

The feature is implemented with a new Thread for each client request in Client class with the help of Request class:

```
/*
* [REQ_09]
* The server implements a request/response pipelining system to optimize the
response for the client.
* Thread solution
* */
//response does not exist in cache
System.out.println(Constants._I+Constants.CLIENT_RES_NO_CACHE.replace("%1",
httpRequest.getHost()));
serverThread = new Request(this.clientSocket, this.clientRequest,
this.clientOutputStream);
serverThread.start();

while(!serverThread.isInterrupted()){
    continue;
}
```

Persistence

Persistence feature can be found in class Cache class with the following method:

```
/*
* [REQ_07] Cache/persistence part
* If the requested web page is not available locally, the server shall forward the
request
* to the web server, get and decode the response, saving files locally and forward
the response
* to the client. This feature is in reality a cache system with persistence.
* */
public void initialize()
[...]
```

Each time the Proxy is started, the initialize method is called. The method checks if there is one or more existing directory saved in cache folder. If any, the server loads each response/request saved under a map of JAXB objects.

JAXB is a specification that maps an XML document to a set of classes and vice versa by means of serialization/deserialization operations named marshalling/unmarshalling.

Caching

In the Client class, a part of code checks if the HTTP request is already in cache and compute if it is always valid. If yes, the server takes the JAXB object, deserializes and forwards it to the client:

```
/*
* [REQ_06]
* If the requested web page is available locally, the server shall transmit it to
the client.
* */
//response exist in cache
System.out.println(Constants._I+Constants.RESPONSE_IS_ALR_CACHED.replace("%1",
httpRequest.getHost()));
```

```
System.out.println(httpResponse.toString());

List<byte[]> serverResponseList = httpResponse.getServerResponseList();
for (int i = 0; i <= serverResponseList.size(); i++) {
    clientOutputStream.write(serverResponseList.get(0));
}
clientOutputStream.flush();
```

The method verifies if the cache-control contains an authorization for caching by the web server. Indeed, the Proxy Server is able to detect if the web server does not allow to cache file thanks to the cache-control value in HttpResponse class:

```
//Check property Cache-Control
String cacheControl = getParam(CACHE_CONTROL);
if(cacheControl == null || cacheControl.indexOf("no-cache") == -1){
    System.out.println(Constants.YES);
    System.out.println(Constants._S+CACHE_CONTROL+"="+cacheControl);
    return true;
}else{
    System.out.println(Constants.NO);
}
```

The application is able to detect if the cache informations are always up to date with the 'isValid" method in HttpResponse class.

Saving cache

Saving cache feature can be found in Cache class of the project under the following method:

```
[...]
public void saveToFile(HttpRequest httpRequest, HttpResponse httpResponse)
[...]
```

The cache is saved in "cache" folder, next to the executable file. Each folder corresponds to a request and responses for a web page or a file. If a client asks for example http://www.perdu.com, a MD5 print is calculated on the complete HTTP request String in order to have a unique folder for each request/response couple. If another client asks for the same web page or file, the MD5 print will be the same. Thanks to that, the server can forward the saved response to the second client.

Future changes requests

The first feature that the Proxy Cache Server can receive for the next release is the verbose or non-verbose mode for console text messages. Indeed, this first release always displays a lot of messages such as complete HTTP requests, HTTP responses and even properties of cache-control sent by the web server. It could be annoying to do not have the possibility to mute the application messages.

The second change request could be an option to apply some blocking filter for kids. This feature can be implemented with a XML file that contains forbidden DNS. Each time a client

requests a forbidden host, the proxy shall display a custom web page to inform that this host was blocked by the proxy cache server.

The third change request could be a full cache encryption method. The client will be no more able to edit request or response XML files with an editor.

User guide

The server is runnable without IDE thanks to the JAR file. At least one argument is mandatory to start the application. Use: Proxy –p <server_port> -d <directory_cache>. If the port argument is not a number, a default 8081 port will be set instead.

Example of use: Proxy –p 8080 –d cache

The application was tested with following web sites:

- www.perdu.com
- www.meme.com
- http://httpwg.org/specs/
- http://httpd.apache.org/docs/2.2/fr/