Mestrado Integrado em Engenharia Informática e Computação

Sistemas Distribuídos 2019/2020

Project 2 - Distributed Backup Service for the Internet

Luis Couto 201204994

Amr Abdalrahman 201903073

Raul Viana 201208089

May 2020

Overview

In our project, we have developed a centralized approach using TCP. We replicate full files and we use thread-based concurrency. Fault-tolerance is achieved avoiding points of failure as described in each protocol.

It is implemented in Java and and we use only Java SE. It supports *backup*, with *replication degree* option, *restore* of files, *delete* of files and *space reclaim* in peers.

We have two packages:

- Main, with the following classes:
  - Peer: which implements the peer;
  - PeerThread: represents the threads for each connection made with peer;
  - PeerDelete: refers to the delete protocol executed by peer;
  - Server: represents the central server which manages the network;
  - ServerThread: implements the threads for the server operations;
  - TestApp: represents the interface to interact with the program;
- Utils, with the following classes:
  - FileCopy: it is a representation of the files in the system, as seen by the server;
  - FilesMap: implements the needed operations to manage the FileCopy objects in the system, consisting mainly in an Hashtable and all the operations to manage it;
  - FileUtil: contains the needed code to operate files in the system itself;
  - Network: represents network of peers as seen by the the central server and consists mainly in an ArrayList and operations to manage it;

To run the code, after files compilation:

- Start Server: java Main.Server <Port>
- Start Peer(s): java Main.Peer localhost <Server Port> <Peer Port/>

Protocols

- **Backup**
  - TestApp sends “BACKUP” command to requested Peer (TCP)

```
case "BACKUP":
      (...)
      output.writeUTF(command + " " + opt1 + " " + opt2);
```

[TestApp.java, lines 49-53]

  - PeerThread is created, asks Server for peers and transfer data to given peers

```
//Establish connection with peer(s)
Socket peerSocket = new Socket("localhost",
Integer.parseInt(peersList[i]));
//Transfer data
DataOutputStream peerOutput = new
DataOutputStream(peerSocket.getOutputStream());
peerOutput.writeUTF("SAVE " + String.valueOf(myPort) + " " +
sendF.getName() + " " + words[2] + " " + sendFile);
(...)
```

[PeerThread.java, lines 93-144]

  - Peers getting “SAVE” command save the file in system, assuring allowed storage capacity and inform Server of saved file

```
case "SAVE":
(...)
InputStream is = clientSocket.getInputStream();
(...)
FileOutputStream file_os = new FileOutputStream(recFile);
(...)
if(Peer.currentStorageInBytes+counter>Peer.maxAllowedStorageInBytes) {
(...)
servOutput.writeUTF("SAVED " + myPort + " " + file_id + " " + repDeg);
(...)
```

[PeerThread, lines 237-332]

  - Upon receiving Info about saved files, the server updates the Files Map

```
case "SAVED":
      if(!(FilesMap.exists(words[2]))) (...)
```

[ServerThread.java, lines 148-166]

    - To execute Backup protocol:

java Main.TestApp localhost <Peer port> BACKUP <File> <replication degree>

- **Restore**
  - TestApp sends “STARTRESTORE” command to requested Peer (TCP)

```
case "RESTORE":
            String fileName = args[3];
            output.writeUTF("STARTRESTORE"+" "+fileName);
            break;
```

[TestApp.java, lines 55-57]

  - The Peer that received the “STARTRESTORE” message from TestApp will request the server to send the ports of all peers that store the specified file.

```
            String serverResponse=servInput.readUTF();
            String[] allPeers=serverResponse.split(" ");
```

[PeerThread.java, lines 165-167]

  - After receiving the ports, it will send a “RESTORE” command to the port of the first peer and wait for 4 seconds. If no reply was sent, it will move to the port of the next peer and do the same request and wait for 4 seconds until some peer sends the file data.

```
      for(int i=0;i<allPeers.length;i++) {
            if(peerDidReply)
                  break;
            (...)
            output.writeUTF(command);
            output.flush();
            socket.setSoTimeout(4000);
            peerDidReply=true;
         try {
           int fileSize=input.readInt();
           fileContents=new byte[fileSize];
           input.readFully(fileContents);
         }catch(Exception e){peerDidReply=false;}
      }
      (...)
```

[PeerThread.java, lines 177-221]

  - The PeerThread that received a “RESTORE” command will check if file exists, if so sends it to the peer that requested the file(TCP)

```
   case "RESTORE":
```

```
(...)
clOutput.write(fileContents);
clOutput.flush();
break;
```

[PeerThread.java, lines 232-251]

- ServerThread responds to client with the IP's and ports of the other clietns (TCP)

```
case "RESTORE":
        (...)
            output.writeUTF(responseToClient);
            output.flush();
```

[ServerThread.java, lines 60-75]

To execute Restore protocol:

java Main.TestApp localhost <Peer port> RESTORE <File>

- **Delete**
  - TestApp sends "DELETE" command to requested Peer (TCP)

```
case "DELETE":
        String file = args[3];//file
        output.writeUTF(command + " " + file);
        break;
```

[AppTest.java, lines 56-59]

  - PeerThread sends "DELETE" command to server, receives a list of peers and initiates a thread for each peer (TCP)

```
case "DELETE":
            (...)
            servOutput.writeUTF(askDelete);
            String response = servInput.readUTF();

            for(int i = 0; i < peersToContact.length; i++){
                new PeerDeleteThread(peersToContact[i],
words[1]).start();
            }
            break;
```

[PeerThread.java, lines 246-257]

  - PeerDeleteThread sends "DELETEFILE" command to every peer (TCP)

```
(...)
```

```
DataOutputStream peerOutput = new DataOutputStream(peerSocket.getOutputStream());
            peerOutput.writeUTF("DELETEFILE " + filename);
(...)
```

[PeerDeleteThread.java, lines 21-30]

- PeerThread receives “DELETEFILE” command and deletes file (TCP)

```
case "DELETEFILE":
            File fot = new File (String.valueOf(myPort) + "/" + "backedup files/" + words[1]);
            if(fot.delete()) System.out.println("File " + words[1] + " in " + myPort + " deleted!");
            else System.out.println("File " + words[1] + " not deleted");
            break;
```

[PeerThread.java, lines 258-266]

To execute Delete protocol:

java Main.TestApp localhost <Peer port> DELETE <File>

- **Space Reclaim**
  - TestApp sends “RECLAIM” command to requested Peer (TCP)

```
case "RECLAIM":
            output.writeUTF(command + " " + args[3]);
            break;
```

[AppTest.java, lines 60-62]

  - PeerThread updates allowed space deletes the necessary files and informs the server about the files that got deleted, (TCP)

```
Peer.maxAllowedStorageInBytes=allowedSpace*1000;
```

```
(...)
if(allowedSpace == 0) {
   deletedFiles = utils.FileUtil.deleteAllFiles(path);
}
else{
  detedFiles    = utils.FileUtil.deleteFiles(allowedSpace,myPort);
}
(...)
```

```
servOutput.writeUTF(messageToServer);
}
break;
```

[PeerThread.java, lines 333-363]

- ServerThread receives “RECLAIM” command, removes the peer from the hashmap (that stores each file name and what peers has that file). Next, server checks replication degree, if not as requested sends “BACKUP” command to peers that has the file(TCP)

```
case "RECLAIM":
                    for(int i=2;i<words.length;i++) {
                        FilesMap.setAchievedReplicationDegree(words[i],
FilesMap.getAchievedReplicationDegree(words[i])-1);
                        FilesMap.removePeerFromFileCopy(words[i], peer);
                    (...)
                     if  (FilesMap.getAchievedReplicationDegree(words[i])
< FilesMap.getdesiredReplicationDegree(words[i])) {
                    (...)
                    for(int j=0;j<Network.peers.size();j++) {
                           if (Network.peers.get(j).equals(peer) == false
&& FilesMap.getPeers(words[i]).contains(Network.peers.get(j)) == true) {
                peerSocket        =        new        Socket("localhost",
Integer.parseInt(Network.peers.get(j)));
        backupOutput                          =                          new
DataOutputStream(peerSocket.getOutputStream());
                    (...)
                    backupOutput.writeUTF(backupCommand);
                    (...)
                break;
```

[ServerThread.java, lines 88-136]

To execute Reclaim protocol:

java Main.TestApp localhost <Peer port> RECLAIM <Space>

Concurrency Design

The concurrency in our project is supported by threads. We decided to create a thread whenever section we expected the code to be more slow.

- Peer and Server classes creates a new thread for each TCP connection:

```
while (true) {
   Socket mySocket = myServerSocket.accept();
   new PeerThread(mySocket, port, serverName, serverPort).start()
   }
```

[Peer.java, lines 68-74]

```
while (true) {
      Socket socket = serverSocket.accept();
      System.out.println("New client connected in " +
socket.getRemoteSocketAddress().toString());
       new ServerThread(socket).start();
```

[Server.java, lines 61-66]

- PeerThread creates a new thread for deletion of files in peers:

```
for(int i = 0; i < peersToContact.length; i++) {
    new PeerDeleteThread(peersToContact[i], words[1]).start();
          }
```

[PeerThread.java, lines 254-256]

Fault-tolerance

In our project, we tried to manage points of failure along each protocol implementation. As main efforts, we have implemented replication degree in backup protocol. The system tries to achieve the requested replication degree on the available peers on the network, if it isn't achieved it tries to backup the file up to 5 times, with an interval of 1 second, in case some peer is down, or enters the system meanwhile.

The peer checks for replication degree in Server when it saves a file:

```
Socket serverSocket2 = new Socket(serverName, serverPort);

new
DataOutputStream(serverSocket2.getOutputStream()).writeUTF("CHECKREPDEG "
+ myPort + " " + file_id + " " + repDeg + " " + fp + " " + peer_id);
```

[PeerThread.java, lines 332-334]

The server confirms if replication degree is achieved for that file and if not, it repeats the backup protocol in in “initiator peer” for backup

```
case "CHECKREPDEG"

(...)

peerOutput.writeUTF("BACKUP " + fp + " " + rd );
```

[ServerThread.java, lines 192-220]