Distributed Backup Service

Relatório de Projeto

Grupo 3

Tiago Luís Salgueiro dos Santos - up201808919@fe.up.pt
Marisa Daniela Quintal Oliveira - up201308594@fe.up.pt

Como correr o projeto

Abrir pasta do projeto.

Em Linux, abrir Shell e correr os seguintes comandos:

1. Sh build.sh
2. Sh rmi.sh
3. Sh peer.sh <id do peer>

De seguida seguir instruções da interface.

Todos os comandos usados para correr o *peer*/*client* são os descritos no enunciado do problema.

Implementação

O grupo optou pelo uso de *RMI* (Remote Method Invocation).

A classe *TestApp* está definida como "*Main*" do projeto, ou seja esta classe é responsável pela interação com o utilizador e por processar os argumentos inseridos pelo mesmo.

A classe *Peer* é responsável por definir endereços, criar os *Sockets* e tudo o que possibilita a comunicação entre as diferentes máquina na rede.

As classes incluídas na package *Handler* encarregam-se da gestão/processamento das mensagens. Dentro desta package encontra-se a classe *Message*, onde é feito o parsing/interpretação das mensagens enviadas e recebidas.

Todas as informações relativas aos *chunks,* métodos para manipular e obter informação sobre os *Chunks* (dados, tamanho do chunk, *id* do chunk, entre outras), encontram-se na classe *Chunk*.

A manipulação de ficheiros é feita na classe *FileManager*. A estrutura de dados usada é *ConcurrentHashMap*, devido ao seu suporte para concorrência de obtenção e atualização de dados. Uma vez que se pretende que vários peers interajam em simultâneo, consideramos que esta escolha é adequada.

FileRepository – estrutura do tipo <*String*, *FileManager*>, utilizada para manter um índice dos ficheiros que serão restaurados.

ChunkRepository – estrutura do tipo <*String*, *Chunk*> em que são armazenados os diferentes Chunks a serem processados.

A classe *FolderManager* é responsável pela criação dos diretórios necessários, bem como à gestão da memória usada. possui também o método *fileSplit,* onde os ficheiros são divididos em chunks para possibilitar o seu envio.

Na package Protocols encontram-se as classes que foram utilizadas para implementar os diferentes protocolos. Foram implementados os protocolos *Backup* e *Delete*.

Implementação do protocolo Backup

De forma a implementar o protocolo Backup, são utilizados os dados, id do ficheiro que se pretende fazer backup, um objeto do tipo Peer e o Replication Degree.

De seguida encontra-se um excerto do código da classe “Backup”:

```
public void run() {

    version = req.getVersion();
    senderID = req.getSenderID();
    fileID = req.getFileID();
    chunkNo = req.getChunkNo();
    replication = req.getReplicationDegree();
    delayLimit = 300;

    if (senderID == peer.getID()) {
            System.out.println("Ignoring backup of own files");
        return;
    }

    data = req.getBody();

    String chunkPathname = peer.getPath("chunks") + "/" + fileID;

    makeDir(peer.getPath("chunks") + "/" + fileID);

    boolean success = false;
    try {
        success = backupFile(Integer.toString(chunkNo), chunkPathname, data);
        //save to database
        peer.addChunkToFileManager(new Chunk(fileID, chunkNo, replication, data.length));
    } catch (IOException e) {
        e.printStackTrace();
    }

    if (! success) {
            System.out.println("Memory overflow");
    } else {
        sendMsgStored();
    }

    System.out.println("Finished backup!");
}
```

Implementação do protocolo Delete

De forma a implementar o protocolo Delete, são utilizados os dados, id do ficheiro que se pretende fazer backup, um objeto do tipo Peer e o Replication Degree.

De seguida encontra-se um excerto do código da classe “Delete”:

```
public void run() {
    String fileID;
    String path;
    Set<Integer> chunks;

    fileID = req.getFileID();

    if (!fileManager.chunkExists(fileID)) {
            System.out.println("Can not find chunks!");
        System.exit(-1);;
    }

    chunks = fileManager.getFileChunksKey(fileID);
    path = peer.getPath("chunks");

    for (Integer chunk : chunks) {
        try {
            Files.delete(Paths.get(path + "/" + fileID + "/" + chunk));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    try {
        Files.delete(Paths.get(path + "/" + fileID));
    } catch (IOException e) {
        e.printStackTrace();
    }

    fileManager.deleteFileBackedUp(fileID);
    System.out.println("Delete successful.");
}
```