PROJET DONNEES REPARTIES

$2^{ème}$ Année, Parcours Systèmes Logiciel

Mehdi Sensali
Faical Toubali

2020 - 2021

Contents

Evaluation

1.1 Partie Technique

1.1.1 Présentation et Résultats

L'architecture de la partie HDFS est correcte et respecte le squelette du sujet fourni. Cependant, elle n'a pas été assez développé pour être testée.

1.2 Synthèse

1.2.1 Correction

Le produit n'a toujours pas été testé. Ceci devra être réglé pour le prochain rendu.

1.2.2 Pertinence

L'ensemble du travail semble pertinent et dans la bonne direction

1.2.3 Cohérence

Comme précisé plus haut, l'architecture est très logique et suit le squelette fourni. Cependant, il est possible d'améliorer la qualité du code et d'enlever quelques redondances.

1.2.4 Points d'amélioration

Il faut mieux développer le serveur NameNode dans lequel on pourra lister l'ensemble des identifiants des serveurs qui sont en train de tourner, et ainsi gérer un plus grand nombre de fichier. L'implantation fournie pour l'instant crée les serveurs localement dans la classe HdfsClient. Il faut également mieux exploiter les méthodes de la classe format pour les lire et écrire des fichiers de différents formats, chose qui n'a pas été fait jusqu'à présent : L'implantation actuelle contourne le problème.]

Annexe

2.1 HdfsClient.java

```java
/* une PROPOSITION de squelette, incomplète et adaptable... */

package hdfs;
import formats.Format;
import formats.KV;
import formats.KVFormat;
import formats.LineFormat;

public class HdfsClient {

    private static void usage() {
        System.out.println("Usage: java HdfsClient read <file>");
        System.out.println("Usage: java HdfsClient write <line|kv> <file>");
        System.out.println("Usage: java HdfsClient delete <file>");
    }

    public static void HdfsDelete(String hdfsFname) {}

    public static void HdfsWrite(Format.Type fmt, String localFSSourceFname,
     int repFactor) { }

    public static void HdfsRead(String hdfsFname, String localFSDestFname) { }


    public static void main(String[] args) {
        // java HdfsClient <read|write> <line|kv> <file>

        try {
            if (args.length<2) {usage(); return;}

            switch (args[0]) {
              case "read": HdfsRead(args[1],null); break;
              case "delete": HdfsDelete(args[1]); break;
              case "write":
                Format.Type fmt;
                if (args.length<3) {usage(); return;}
                if (args[1].equals("line")) fmt = Format.Type.LINE;
                else if(args[1].equals("kv")) fmt = Format.Type.KV;
                else {usage(); return;}
                HdfsWrite(fmt,args[2],1);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

}
```

2.2 NameNode.java

```java
package hdfs;

import java.util.ArrayList;

public class NameNode {

  private int tailleFichier;
  private int tailleChunk;
  private int nbrChunks;
  private ArrayList<String> listChunkIDs;

  public NameNode(int tailleFichier, int tailleChunk, int nbrChunks) {
    this.tailleFichier = tailleFichier;
    this.tailleChunk = tailleChunk;
    this.nbrChunks = nbrChunks;
    this.listChunkIDs = new ArrayList<String>();
  }

  public void appendToListChunkIDs(final String LocalFSDestFname)
  {
    if (nbrChunks < 1) return;

    for (int i = 1; i <= nbrChunks; i++){
      listChunkIDs.add(LocalFSDestFname + "_chunk" + i);
    }
  }

  public void removeFromListChunkID(final String LocalFSDestNale) {
    if(listChunkIDs.contains(LocalFSDestNale)) {
      listChunkIDs.remove(LocalFSDestNale);
    }
  }

  // Getters and Setters
    ----------------------------------------------------------------------
  public int getTailleFichier() {
    return tailleFichier;
  }

  public int getTailleChunk() {
    return tailleChunk;
  }

  public int getNbrChunks() {
    return nbrChunks;
  }

  public ArrayList<String> getListChunkID() {
    return listChunkIDs;
  }

  public void setTailleFichier(final int newTailleFichier) {
    tailleFichier = newTailleFichier;
  }

  public void setTailleChunk(final int newTailleChunk) {
    tailleChunk = newTailleChunk;
  }

  public void setNbrChunks(final int newNbrChunks) {
    nbrChunks = newNbrChunks;
  }
```

```
62 }
```

2.3 HdfsServer.java

```java
package hdfs;

import java.io.IOException;
import java.io.*;
import java.net.*;

public class HdfsServer {

  private static boolean isRunning = true;


  private static void writeChunk(String FileName, String FileContent) {
    try {
      final File fileToWrite = new File("/temp/" + FileName);
      BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(fileToWrite));
      bufferedWriter.write(FileContent);
      bufferedWriter.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  private static String readChunk(String FileName) {
    String Chunk = "";

    try {
      final File fileToRead = new File("/temp/" + FileName);
      BufferedReader bufferedReader = new BufferedReader(new FileReader(fileToRead));
      String ChunkLine = bufferedReader.readLine();

      while (ChunkLine != null) {
        Chunk += ChunkLine + "\n";
        ChunkLine = bufferedReader.readLine();
      }
      bufferedReader.close();

    } catch (IOException e) {
      e.printStackTrace();
    }
    return Chunk;
  }

  private static void deleteChunk(String FileName) {
    File fileToDelete = new File("/temp/" + FileName);
    fileToDelete.delete();
  }

  public static void main(String[] args) {

    if (args.length == 0)
      return;

    final int port = Integer.parseInt(args[0]);
    try {
      ServerSocket serverSocket = new ServerSocket(port);

      while(isRunning) {

        Socket socket = serverSocket.accept();

        ObjectInputStream objInStream = new ObjectInputStream(socket.getInputStream());
        String[] message = (String[]) objInStream.readObject();
```

```
63          Commande command = Commande.valueOf(message[0]);
64
65          switch (command) {
66            case CMD_WRITE:
67              writeChunk(message[1], message[2]);
68              break;
69
70            case CMD_READ:
71              String Chunk = readChunk(message[1]);
72              ObjectOutputStream objOutStream = new ObjectOutputStream(socket.getOutputStream
    ());
73              objOutStream.writeObject(Chunk);
74              objOutStream.close();
75              break;
76
77            case CMD_DELETE:
78              deleteChunk(message[1]);
79              break;
80
81            default:
82              break;
83          }
84
85          objInStream.close();
86          socket.close();
87        }
88
89        serverSocket.close();
90
91      } catch (IOException | ClassNotFoundException e) {
92        e.printStackTrace();
93      }
94
95    }
96
97    public boolean getIsRunning() {
98      return isRunning;
99    }
100
101   public void setIsRunning(boolean isRunning) {
102     HdfsServer.isRunning = isRunning;
103   }
104 }
```