PROJET DONNEES REPARTIES

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

Reda El Jai
Younes Saoudi

2020 - 2021

Contents

Chapter 1

Evaluation

Partie Technique

1.1.1 Présentation et Résultats

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

Synthèse

1.2.1 Correction

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

1.2.2 Complétude

Il manque plusieurs points de la spécification, comme les tests, l'utilisation du Sort Comparator; etc.

1.2.3 Pertinence

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

1.2.4 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.

Points d'amélioration

Il faut mieux factoriser le code et se débarrasser des doublons. Il faut surtout tester le produit et évaluer les résultats.

Chapter 2

Annexe

2.0.1 CallbackInterface.java

```java
package ordo;
import java.io.Serializable;
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface CallBackInterface extends Remote, Serializable {
    // indique que le worker a terminé l'exécution de map
  public void confirmFinishedMap() throws InterruptedException, RemoteException;

  // Indique le nombre de procéssus map non achevés
    public void waitForMapToFinish(int nb) throws InterruptedException, RemoteException;
}
```

2.0.2 CallBack.java

```java
package ordo;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.concurrent.Semaphore;

public class CallBack extends UnicastRemoteObject implements CallBackInterface {

    /**
   *
   */
  private static final long serialVersionUID = 1L;
  private Semaphore nbMapsFinished;

    public CallBack() throws RemoteException{
        super();
        nbMapsFinished = new Semaphore(0);
    }

    @Override
    public void confirmFinishedMap() {
        nbMapsFinished.release();
    }

    @Override
    public void waitForMapToFinish(int nb) {
        for(int i = 0; i < nb; i++) {
      try {
        nbMapsFinished.acquire();
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
    }
}
```

2.0.3 Job.java

```java
package ordo;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.net.InetAddress;
import java.rmi.Naming;
import java.rmi.RemoteException;
import formats.*;
import map.MapReduce;
import java.util.ArrayList;
import java.util.List;

import static hdfs.HdfsClient.HdfsRead;


public class Job implements JobInterface {

  private int numberOfReduces;
  private int numberOfMaps;
  private Format.Type inputFormat;
  private Format.Type outputFormat;
  private String inputFName;
  private String outputFName;
  private Format.Type interFormat;
  private String interFName;
  private List<String> machines; //la liste des machines sur lesquelles tournent les workers


  // Les constructeurs :

  // Constructeur vide avec les données minimums
  // Le reste à étant à remplir par l'utilisateur
  public Job() {
    this.initMachines();
    this.numberOfMaps = machines.size();
    this.numberOfReduces = 1; //Pour l'instant on se contentera de 1
  }

  // On peut aussi ajouter directement l'input
  // L'output étant à remplir par l'utilisateur
  public Job(Format.Type inputFormat, String inputFName) {
    this();
    this.inputFormat = inputFormat;
    this.inputFName = inputFName;

    this.outputFName = inputFName + "-res";
    this.interFName = inputFName + "-inter";
    this.outputFormat = inputFormat;
    this.setInterFormat(inputFormat);
  }

  // La méthode qui sera utilisée dans l'implémentation du map-reduce
    public void startJob (MapReduce mr) {
    // création des formats
    Format input, inter, output;
        if(inputFormat == Format.Type.LINE) { // LINE
      input = new LineFormat(inputFName);
      inter = new KVFormat(interFName);
      output = new LineFormat(outputFName);
    } else { // KV
```

```
63        input = new KVFormat(inputFName);
64        inter = new KVFormat(interFName);
65        output = new KVFormat(outputFName);
66      }
67
68        // récupération de la liste des workers sur l'annuaire
69        List<WorkerInterface> workers = new ArrayList<>();
70        for(int i = 0; i < this.numberOfMaps; i++) {
71          try {
72              // On va récupérer les workers en RMI sur un annuaire
73          // On vise aussi à lancer plusieurs workers dans plusieurs machine après.
74          String nomMachine = InetAddress.getLocalHost().getHostName();
75
76          workers.add((WorkerInterface) Naming.lookup(nomMachine));
77          //workers.add((Worker) Naming.lookup("//localhost/Worker1"));
78        } catch (Exception e) {
79          e.printStackTrace();
80        }
81        }
82
83        // On initialise le callback pour que les workers puissent renvoyer leurs résultats
84      CallBackInterface cb = null;
85      try {
86        cb = new CallBack();
87      } catch (RemoteException e) {
88        e.printStackTrace();
89      }
90
91      // Puis on va lancer les maps sur les différents workers
92      int ind = 0;
93      String nameInput = input.getFname();
94      String nameinter = inter.getFname();
95      for(WorkerInterface worker : workers) {
96        try {
97          /* on appelle le map sur le worker
98           on utilise le même format input et le même format output pour chacun
99           car par RMI on envoie des copies, et c'est lorsque les formats seront "open"
100          sur les différents workers, que s'effectuera le chargement des différents chunks
101         */
102
103         ((LineFormat) input).setFname(nameInput + ind);
104         ((KVFormat) inter).setFname(nameinter + ind);
105
106         ind ++;
107
108         worker.runMap(mr, input, inter, cb);
109
110       } catch (RemoteException e) {
111
112         e.printStackTrace();
113       }
114     }
115
116     // Puis on attends que tous les workers aient finis leur travail
117     try {
118       cb.waitForMapToFinish(numberOfMaps);
119     } catch (Exception e) {
120       e.printStackTrace();
121     }
122
123     // On utilise HDFS pour récupérer le fichier résultat concaténé dans resReduce
124     Format resReduce;
125     /* if(inputFormat == Format.Type.LINE) {
126       resReduce = new FormatLine("resReduceFormat");
127       System.out.println(" Ecriture du fichier intermédiaire");
```

```
    } else {
      resReduce = new FormatKV("resReduceFormat");
    }*/
    resReduce = new KVFormat("resReduceFormat");
    System.out.println("nom du fichier qu'on veut lire" + inter.getFname());
    HdfsRead(nameinter, resReduce.getFname());

      // On veut transformer ce fichier en un format local
        output.open(Format.OpenMode.R);

    // Puis on applique le reduce sur le résultat concaténé des maps
    // On stock le résultat dans l'output
    resReduce.open(Format.OpenMode.R);
    mr.reduce(resReduce, output);
    resReduce.close();

    // On extrait une liste de notre format output pour pouvoir le trier
    List<KV> listeTriee = new ArrayList<>();
      KV kv;
      while((kv = output.read()) != null) {
        listeTriee.add(kv);
    }

    // Puis on l'écrit dans le fichier de sortie
    File fOutput = new File(outputFName);
    try {
      BufferedWriter bw = new BufferedWriter(new FileWriter(fOutput));
      for(KV ligne : listeTriee) {
        bw.write(ligne.k);
        bw.write(KV.SEPARATOR);
        bw.write(ligne.v);
        bw.newLine();
      }
      bw.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  // Les méthodes utilisées pour faciliter les opérations

  public void setInputFname(String fname){
      this.inputFName = fname;
      this.outputFName = fname + "-res";
    this.interFName = fname + "-inter";
    }

    public void setOutputFname(String fname){
      this.outputFName = fname;
    }

    public void setNumberOfReduces(int tasks){
      this.numberOfReduces = tasks;
    }

    public void setNumberOfMaps(int tasks) {
      this.numberOfMaps = tasks;
    }

    public void setInputFormat(Format.Type ft){
      this.inputFormat = ft;
    this.outputFormat = inputFormat;
    this.setInterFormat(inputFormat);
    }
```

```
193     public void setOutputFormat(Format.Type ft){
194       this.outputFormat = ft;
195     }
196
197
198     public int getNumberOfReduces(){
199       return this.numberOfReduces;
200     }
201
202     public int getNumberOfMaps() {
203       return this.numberOfMaps;
204     }
205
206
207     public String getInputFname(){
208       return this.inputFName;
209
210     }
211
212     public String getOutputFname(){
213       return this.outputFName;
214   }
215
216   public Format.Type getInputFormat(){
217       return this.inputFormat;
218
219     }
220
221     public Format.Type getOutputFormat(){
222       return this.outputFormat;
223     }
224
225     public void initMachines(){
226       this.machines = new ArrayList<String>();
227         machines.add("Mehdi");
228         machines.add("Younes");
229       machines.add("Reda");
230       machines.add("Faical");
231
232   }
233
234   public Format.Type getInterFormat() {
235     return interFormat;
236   }
237
238   public void setInterFormat(Format.Type interFormat) {
239     this.interFormat = interFormat;
240   }
241 }
```

2.0.4 MapRunner.java

```java
package ordo;

import java.rmi.RemoteException;

import formats.Format;
import map.Mapper;

public class MapRunner extends Thread {

  WorkerInterface worker; //deamon sur lequel on va lancer le runMap
  Mapper mapper; //map à lancer
  Format reader, writer; //les formats de lecture et d'écriture
  CallBackInterface callbackInterface;

  public MapRunner(WorkerInterface worker, Mapper mapper, Format reader, Format writer,
    CallBackInterface callbackInterface){
    this.worker = worker;
    this.mapper = mapper;
    this.reader = reader;
    this.writer = writer;
    this.callbackInterface = callbackInterface;
  }

  public void run() {
    try {
      this.worker.runMap(this.mapper, this.reader, this.writer, this.callbackInterface);
    } catch (RemoteException e) {
      e.printStackTrace();
    }

  }
}
```

2.0.5 WorkerInterface.java

```java
package ordo;

import java.rmi.Remote;
import java.rmi.RemoteException;

import map.Mapper;
import formats.Format;

public interface WorkerInterface extends Remote {
  public void runMap (Mapper m, Format reader, Format writer, CallBackInterface cb) throws
    RemoteException;
}
```

2.0.6 Worker.java

```java
package ordo;

import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import formats.Format;
import map.Mapper;

public class Worker extends UnicastRemoteObject implements WorkerInterface {

  private static final long serialVersionUID = 1L;

  private String name; // Le nom de chaque Worker.

  protected Worker(String name) throws RemoteException {
    super();
    this.name = name;
  }

  @Override
    /* On appelle le map fourni en paramètre sur le reader et on écrit sur le writer
     Quand on a finit, on appelle le callback pour l'informer */
  public void runMap(Mapper m, Format reader, Format writer, CallBackInterface cb) throws
    RemoteException {
    // On ouvre le formats sur le démons, pour récupérer les chunks
    reader.open(Format.OpenMode.R);
    writer.open(Format.OpenMode.W);

    m.map(reader, writer);

    reader.close();
    writer.close();
    try {
      cb.confirmFinishedMap();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }

  /* Lancement du worker dans la machine
    */
  public static void main(String args[]) {
    try {
      WorkerInterface worker = new Worker(args[0]);
      // On l'enregistre auprès du serveur de nom, qu'il faudra avoir lancé au préalable !
            //String nomMachine = InetAddress.getLocalHost().getHostName();
            //Naming.rebind("//" + "localhost/" + ((WorkerImpl) worker).getName(), worker);
            //Registry registry = LocateRegistry.createRegistry(8887);
            //registry.rebind("//localhost:8887",  worker);
            Naming.rebind("//localhost:8888/" + ((Worker) worker).getName(), worker);

    } catch (Exception e) {
      e.printStackTrace();
    }
  }

    public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
```

```
62    }
63
64 }
```