Projet de Données Réparties
Service Hidoop

FLAGEAT Valentin | GRÉAUD Baptiste

Table des matières

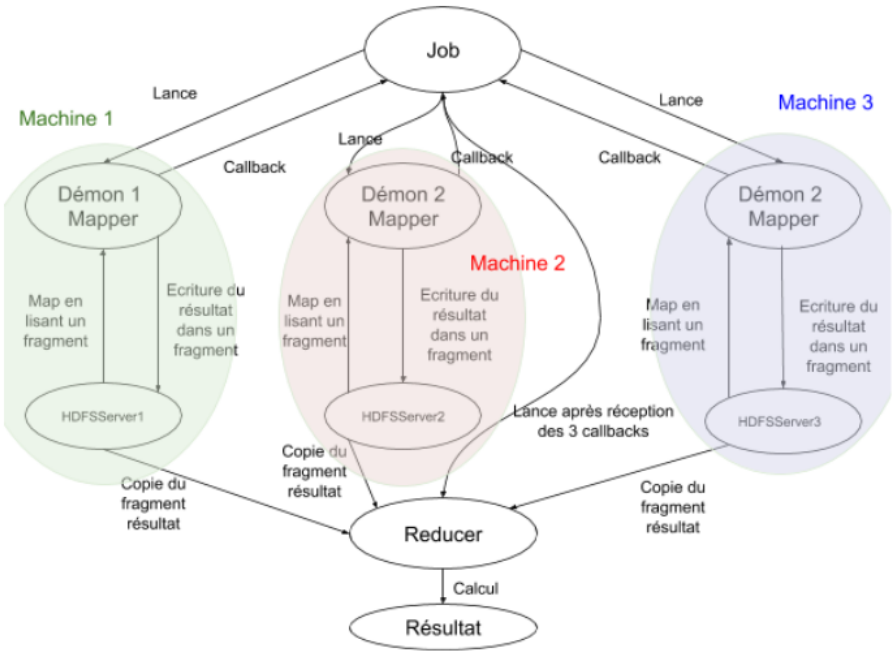
Introduction

Le schéma MapReduce permet d'effectuer en parallèle (sur une grappe (cluster) de machines) des traitements sur un grand volume de données. Les données sont découpées en fragments qui sont répartis (stockés) sur les machines de traitement. Les fragments sont alors traités en parallèle sur les différentes machines, et les résultats partiels issus du traitement des fragments sont alors agrégés pour donner le résultat final. Ainsi, une application Map-Reduce spécifie essentiellement :

- une procédure map(), qui est exécutée en parallèle sur les machines de traitement sur tousles fragments (auxquels on accède donc localement)
- une procédure reduce() qui exploite le résultat des map pour obtenir le résultat final.

L'objectif de notre travail est de créer un service Hidoop contrôlant l'exécution répartie et parallèle des traitements map, la récupération des résultats et l'exécution du reduce.

1 Architecture



2 Implémention côté serveur

Un démon (Daemon) doit être lancé sur chaque machine. Nous proposons d'utiliser RMI pour la communication entre les démons et le client. Nous créons donc une interface qui hérite de Remote et qui contient la méthode runMap qui sera appelé à distance par le client pour lancer le map. Les paramètres de la fonction runMap sont les suivants :

- Mapper m: il s'agit du programme map à appliquer sur un fragment hébergé sur la machineoù s'exécute le démon
- Format reader: il s'agit du fichier (dans un format donné par la classe de reader) sur lamachine où s'exécute le démon, contenant le fragment sur lequel doit être appliqué le map
- Format writer: il s'agit du fichier (dans un format donné par la classe de writer) sur lamachine où s'exécute le démon, dans lequel les résultats du map doivent être écrits
- CallBack cb: il s'agit d'un objet de rappel, appelé lorsque l'exécution du map est terminée

Listing 1: Daemon.java

```
package ordo;

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

import map.Mapper;
import formats.Format;

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

Voici l'implémentation de l'interface Daemon.java. On choisit pour l'instant de lancer manuellement le serveur sur chaque machine en fournissant le numéro du démon (au cas où on travaille en local sur le même port). La création du registre et l'enregistrement du serveur dans celui-ci est défini dans le main de la classe. La méthode runMap ouvre le fragment et le fichier où il écrira le résulat puis applique le map et ferme les fichiers. Dans cette version, on considère que les fragments sont déja sur les machines (en l'absence de HDFS).

Listing 2: DaemonImpl.java

```
package ordo;

import map.Mapper;
import formats.Format;
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject ;
import java.rmi.registry.* ;
import java.util.*;
import java.util.regex.Pattern;
import java.net.InetAddress;
import java.lang.*;


public class DaemonImpl extends UnicastRemoteObject implements Daemon {

    private static final long serialVersionUID = 1L;

    public DaemonImpl() throws RemoteException {
    }

    public void runMap (Mapper m, Format reader, Format writer,Callback
         cb) throws RemoteException {
        System.out.println("Lancement de la tche ... ");
      System.out.println("Ouverture du fichier contenant le fragment sur
          lequel excuter le map");
        reader.open(Format.OpenMode.R);
        System.out.println("Ouverture du fichier dans lequel les
            rsultats du map doivent tre crits ");
        writer.open(Format.OpenMode.W);
        System.out.println("Lancement du map sur le fragment ...");
      m.map(reader, writer);
      System.out.println("Map termin !!!");
      System.out.println("Fermeture des fichiers en lecture et criture
          ...");
      reader.close();
      writer.close();
      System.out.println("Envoie du callback ...");
        cb.incNbMapDone();
        System.out.println("Travail termin !!!");


    }
    public static void main(String[] args) {
        int port,num;
        try {
            // dfinition du port et du numro du dmon
            port = Integer.parseInt(args[0]);
            num = Integer.parseInt(args[1]);
        } catch (Exception ex) {
            System.out.println(" Usage : java ordo/DaemonImpl <port>
```

```
                <numero du dmon>");
            return;
        }

        try {
         // Cration du registre RMI
            System.out.println("Cration du RMI ...");
            Registry registry = LocateRegistry.createRegistry(port);
            System.out.println("RMI cr !");
        } catch (Exception e) {
            System.out.println("Le RMI existe dj !");
        }

        try {
         // Enregistrement du dmon dans le registre
            System.out.println("Enregistrement du dmon dans le registre");
            DaemonImpl demon = new DaemonImpl();
            Naming.rebind("//localhost:"+port+"/DaemonImpl"+num,demon);
            System.out.println("//localhost:"+port+"/DaemonImpl"+num+"
                bound in registry");
        } catch (Exception e) {
            System.out.println("Le port sur lequel vous souhaitez vous
                connecter est occup !");
        }
    }

}
```

3 Implémentation coté client

Pour lancer un calcul parallèle, Hidoop fournit une classe Job, implantant une interface JobInterface.

Listing 3: JobInterface.java

```
package ordo;
import map.*;
import formats.*;

public interface JobInterface {
// Mthodes requises pour la classe Job
    public void setInputFormat(Format.Type format);
    public Format.Type getInputFormat();
    public void setInputFname(String fname);
    public String getInputFName();
    public void setOutputFormat(Format.Type format);
    public Format.Type getOutputFormat();
```

```
    public void setOutputFname(String Fname);
    public String getOutputFName();
    public void setResReduceFormat(Format.Type format);
    public Format.Type getResReduceFormat();
    public void setResReduceFname(String Fname);
    public String getResReduceFName();

    public void startJob (MapReduce mr);
}
```

Listing 4: Job.java

```
package ordo;

import java.rmi.*;
import java.rmi.registry.*;
import java.util.*;
import java.lang.*;
import map.*;
import formats.*;
import application.*;
import java.io.*;

public class Job implements JobInterface {

    private Format.Type inputFormat;
    private Format.Type outputFormat;
    private Format.Type resReduceFormat;
    private String inputFName;
    private String outputFName;
    private String resReduceFName;
    private int nbMaps;

    public Job(Format.Type inputFormat, String inputFName, int maps) {
      this.inputFormat = inputFormat;
      this.inputFName = inputFName;
      this.outputFormat = Format.Type.KV;
      this.outputFName = inputFName + "-tmp";
      this.resReduceFormat = Format.Type.KV;
      this.resReduceFName = inputFName + "-resf";
        this.nbMaps = maps;
    }

    public void startJob (MapReduce mr){

            System.out.println("Lancement de startJob ...");

            // Il manque ici la commande hdfs write pour envoyer les
```

```
        fragments sur les dmons
    // HdfsWrite(inputFormat, inputFName, 1);

    // Cration des formats
    Format input, output, resReduce;
    if(inputFormat == Format.Type.LINE) { // LINE
    input = new LineFormat(getInputFName());
} else { // KV
    input = new KVFormat(getInputFName());
}
resReduce = new KVFormat(getResReduceFName());
output = new KVFormat(getOutputFName());

//  rcupration  de la liste des dmons
System.out.println("Rcupration de la liste des Daemons ...");
List<Daemon> demons = new ArrayList<>();
for(int i = 0; i < getNbMaps(); i++) {
    try {
        System.out.println("On rcupre le stub de :
            //localhost:4321/DaemonImpl"+ (i+1) );
        demons.add((Daemon)
            Naming.lookup("//localhost:4321/DaemonImpl"+ (i+1) ));
    } catch (Exception e) {
        e.printStackTrace();
    }
}

// Initialisation du callback
System.out.println("Initialisation du Callback ...");
Callback cb = null;
try {
    cb = new CallbackImpl(getNbMaps());
} catch (RemoteException e) {
    e.printStackTrace();
}

// Lancement des maps sur les dmons
System.out.println("Lancement des maps ...");
for(int i = 0; i < getNbMaps(); i++) {
    Daemon d = demons.get(i);

    // On change le nom des Formats en rajoutant un numro pour
        que les fragments aient des noms diffrents pour chaque
        Daemon
    Format inputTmp;
      if(inputFormat == Format.Type.LINE) { // LINE
        inputTmp = new LineFormat(input.getFname() + "" + i);
    } else { // KV
        inputTmp = new KVFormat(input.getFname() + "" + i);
    }
```

```
            Format outputTmp = new KVFormat(output.getFname() + "" + i);

           // on appelle le map sur le dmon
           MapRunner mapRunner = new MapRunner(d, mr, inputTmp,
               outputTmp, cb);
           mapRunner.start();
        }
        System.out.println("OK\n");

        // Puis on attends que tous les dmons aient finis leur travail
        System.out.println("Attente du callback des Daemons ...");
        try {
           cb.waitMapDone();
        } catch (RemoteException e) {
           e.printStackTrace();
        }
        System.out.println("OK\n");

           // Il manque ici la commande hdfs read pour rassembler les
               fragments de rsultats intermdiaires
           // HdfsRead(inputFName,outputFName);

        // On peut alors lancer le reduce

        System.out.println("Ouverture du fichier contenant la
            concatnation des rsultats des map");
        output.open(Format.OpenMode.R);
        System.out.println("Ouverture du fichier dans lequel les
            rsultats du reduce doivent tre crits ");
        resReduce.open(Format.OpenMode.W);
        System.out.println("Lancement du reduce ...");
        mr.reduce(output, resReduce);
        System.out.println("Reduce termin !!!");
        System.out.println("Fermeture des fichiers en lecture et
            criture ...");
        output.close();
        resReduce.close();
    }


    public void setInputFormat(Format.Type format){
        this.inputFormat = format;
    }

    public Format.Type getInputFormat() {
      return this.inputFormat;
    }

    public void setInputFname(String fname){
        this.inputFName = fname;
```

```
 }

 public String getInputFName() {
   return this.inputFName;
 }

 public void setOutputFormat(Format.Type format){
     this.outputFormat = format;
 }

 public Format.Type getOutputFormat() {
   return this.outputFormat;
 }

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

 public String getOutputFName() {
   return this.outputFName;
 }

 public void setResReduceFormat(Format.Type format){
     this.resReduceFormat = format;
 }

 public Format.Type getResReduceFormat() {
   return this.resReduceFormat;
 }

 public void setResReduceFname(String fname){
     this.resReduceFName = fname;
 }

 public String getResReduceFName() {
   return this.resReduceFName;
}

 public void setNbMaps(int maps){
     this.nbMaps = maps;
 }

 public int getNbMaps() {
   return this.nbMaps;
}


 public static void main(String[] args) {
     System.out.println("------------------------------------------------------------------");
   System.out.println("-------------------- Interface Hidoop
```

```
    ---------------------------");
  System.out.println("-----------------------------------------------------------------------");
  Scanner sc = new Scanner(System.in);
  String path = System.getProperty("user.dir");
  String pathParent = path.substring(0,path.lastIndexOf("/"));
  String data = pathParent+"/data";
  String application = path + "/application";
  File repertoireD = new File(data);
  String[] liste = repertoireD.list();
  String str;

  do {
      System.out.println("Veuillez choisir le nom du fichier parmis
          ceux disponibles (Si le fichier que vous cherchez n'est
          pas propos, vrifiez qu'il est prsent dans le dossier
          data) :");
      for (int i = 0; i < liste.length; i++) {
          System.out.println(liste[i]);
      }
      str = sc.nextLine();
  } while ( !(new File(data+"/"+str)).exists());

  int n = 3;
  Format.Type f = Format.Type.LINE;
  if (n == 1) {
      f = Format.Type.LINE;
  } else if (n==2) {
      f = Format.Type.KV;
  } else {
      System.out.println("Slectionnez le format de ce fichier :");
      System.out.println("1 - LINE");
      System.out.println("2 - KV");
      n = sc.nextInt();
      sc.nextLine();
  }

  int m;
  do {
      System.out.println("Veuillez choisir un nombre de machines
          utiliser pour cette opration (entre 1 et 3):");
      m = sc.nextInt();
      sc.nextLine();
  } while (m<1 || m>3);

Job j = new Job(f,str,m);

  String mr;
  do {
      System.out.println("Veuillez saisir l'application Map/Reduce
          appliquer sur le fichier parmis ceux disponibles (Si
```

```
                l'application que vous cherchez n'est pas propose,
                vrifiez qu'elle est prsente dans le dossier
                application):");
            File repertoireA = new File(application);
            liste = repertoireA.list();
            for (int i = 0; i < liste.length; i++) {
                if (liste[i].contains("java")) {
                    System.out.println(liste[i].substring(0,liste[i].lastIndexOf(".")));
                }
            }
            mr = sc.nextLine();
        } while ( !(new File(application+"/"+mr+".java")).exists());

        /*
        MapReduce map = null;

        try {
          map = (MapReduce) Class.forName(mr).newInstance();
      } catch (Exception e) {
            e.printStackTrace();
        }
        */

        MapReduce map = new MyMapReduce();

      j.startJob(map);

    }
}
```

Nous utilisons 7 attributs pour gérer le format et le noms du fragment, du fichier de résultat temporaire du map,du fichier du résultat du reduce et du nombre de maps à effectuer. Le main propose une interface utilisateur dans le terminal pour choisir ces données.

Cette classe permet de lancer notamment la méthode startJob prenant en paramètre :

- MapReduce mr: correspond au programme MapReduce à exécuter en parallèle (voir modèlede programmation)

Cette méthode commence par créer les formats qui passeront en pramètre du runMap et du reduce. Ensuite, elle récupère la liste des démons, elle instancie un objet Callback (voir plus bas pour les explications), elle lance ensuite la méthode mapRunner sur un objet de la classe MapRunner (voir plus bas) . On attend ensuite le callback de tous les démons puis on lance le reduce.

La classe MapRunner hérite de la classe Thread. Elle sert à lancer un thread qui va lancer la méthode runMap. Cela permet de lancer les maps en parallèle sur les démons.

Listing 5: MapRunner.java

```
package ordo;

import java.rmi.RemoteException;
import formats.Format;
import map.Mapper;

public class MapRunner extends Thread {

   Daemon deamon; //deamon sur lequel on va lancer le runMap
   Mapper m; //map  lancer
   Format reader, writer; //les formats de lecture et d'criture
   Callback cb;

   public MapRunner(Daemon deamon, Mapper m, Format reader, Format
       writer, Callback cb){
      this.deamon = deamon;
      this.m = m;
      this.reader = reader;
      this.writer = writer;
      this.cb = cb;
   }

   public void run() {
      try {
         this.deamon.runMap(this.m, this.reader, this.writer, this.cb);
      } catch (RemoteException e) {
         e.printStackTrace();
      }

   }
}
```

La classe CallbackImpl, implémentation de Callback, permet de savoir quand est-ce que les démons ont fini leur travail. Pour cela, on utilise les outils de synchronisation wait() et notify(). On a une méthode incNbMapDone() qui va être appelé par les démons lorsqu'ils auront fini leur map et qui a pour effet d'incrémenter la variable nbMapsDone. Le client va utiliser la méthode waitMapDone() qui attend que nbMapsDone == nbMaps.

Listing 6: Callback.java

```
package ordo;

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

public interface Callback extends Remote {
```

```
   public void incNbMapDone () throws RemoteException;
   public void waitMapDone() throws RemoteException;
}
```

Listing 7: CallbackImpl.java

```
package ordo;

import java.rmi.*;
import java.rmi.server.UnicastRemoteObject ;
import java.util.*;
import java.lang.*;

public class CallbackImpl extends UnicastRemoteObject implements
    Callback {

   private static final long serialVersionUID = 2674880711467464646L;
   private int nbMapDone;
   private int nbMaps;
   private final Object lock = new Object();

   public CallbackImpl(int maps) throws RemoteException {
      this.nbMapDone = 0;
      this.nbMaps = maps;
   }

   public void incNbMapDone () throws RemoteException {
      this.nbMapDone ++;
      synchronized(lock){
         lock.notify();
      }

   }

   public void waitMapDone() throws RemoteException {
      while (this.nbMapDone < nbMaps) {
         try {
            synchronized(lock){
               lock.wait();
            }

         } catch (final InterruptedException e) {
            e.printStackTrace();
         }
      }
   }
}
```

Conclusion

Cette version permet donc d'effectuer un MapReduce en local ou sur différentes machines sous réserve que les fragments soient présents sur les machines et que les résultats des maps soient mis à la main dans un fichier temporaire pour effectuer le reduce. Les fichiers doivent être mis dans le répertoire source.