Projet de Données Réparties
Evaluations croisées

Flageat Valentin | Gréaud Baptiste

Table des matières

Introduction **2**

1 Partie technique **2**

2 Synthèse **3**

3 Annexe **5**

Introduction

Cette étape consiste à effectuer une revue du code de l'autre binôme en effectuant des tests de fonctionnalité et de performance et en évaluant la correction, la complétude, la pertinence et la cohérence du code dans le but de proposer des améliorations.

1 Partie technique

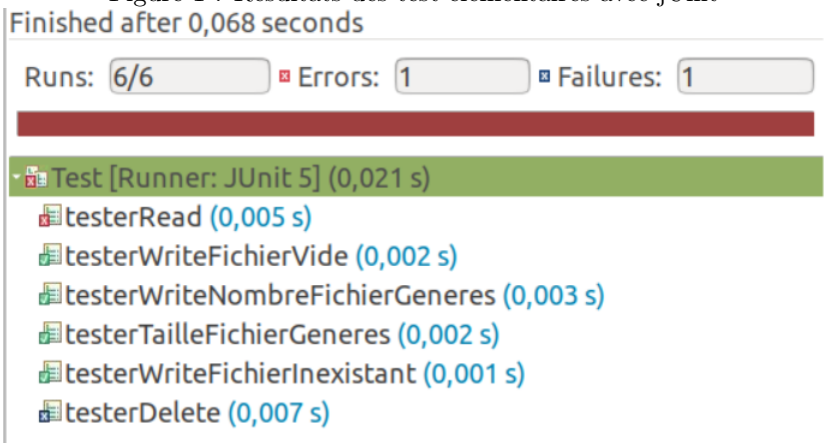
Le script src/hdfs/Test.java (voir Annexe) contient un ensemble de test sur la classe HdfsClient. Certaines fonctionnalités de la classe HdfsClient étant inutilisables, tous les tests envisagés n'ont pas été implémentés. La liste des tests est la suivante. Sont indiqués en italique les tests qui ont été implémentés dans la classe Test :

HdfsWrite	• *testerWriteNombreFichierGeneres* qui vérifie le nombre de chunks générés par HdfsWrite • *testerTailleFichierGeneres* qui vérifie que la taille des chunk générés est la bonne • *testerWriteFichierInexistant* qui teste le comportement de HdfsWrite en cas d'appel avec un fichier inexistant • *testerWriteFichierVide* qui teste le comportement de HdfsWrite en cas d'appel avec un fichier vide
HdfsRead	• *testerRead* qui vérifie la génération d'un fichier de résultat par la fonction HdfsRead • test avec un fichier inexistant • test avec un ensemble de chunks contenant un chunk vide ou erroné • test avec un fichier ouvert dans une autre application
HdfsDelete	• *testerDelete* qui vérifie la suppression de chunk par la fonction HdfsDelete • test avec un fichier inexistant • test test avec un fichier ouvert dans une autre application

Les tests implémentés traitent des fichiers dans src/hdfs/. Le chemin d'accès à ces fichiers est écrit en "dur" dans la classe Test (attribut path). Il est nécessaire de l'éditer pour pouvoir lancer les tests sur le fichier exemple.txt. La taille du fichier testé ainsi que la taille des chunks (fixée par la classe HdfsClient) ont également été écrites en "dur" (attributs tailleFichier et tailleChunk).

Voici les résultats des tests sur jUnit :

Figure 1 : Résultats des test élémentaires avec jUnit



2 Synthèse

Remarques générales : On notera que le code est bien commenté avec notamment la spécification de chaque fonction.

Correction : est ce que le produit fonctionne correctement ? Les résultats sont-ils justes ? D'après la partie précédente, on remarque que le produit ne fonctionne pas complètement. Au niveau des tests fonctionnels et de robustesse, seul la fonction Write passe les tests. Les fonctions Delete et Read ne fonctionnant pas sur les cas élémentaires, les tests n'ont pas été poussés plus loin.

Complétude : est ce que tous les points de la spécification ont été abordés et traités ? Points non traités :

- HDFS doit être composé d'un démon HdfServer qui doit être lancé sur chaque machine, or aucune classe n'est définie pour cela.
- HdfsClient utilise des sockets en mode TCP mais ne communique pas clairement avec un serveur. Il n'y a pas d'interactions Client/Serveur.
- Il n'y a donc par conséquent pas d'initiation de l'interaction par l'envoi d'un message spécifiant la commande à exécuter par le serveur

- Le code Commande n'est pas défini.

Points abordés :

- La classe HdfsClient
- Les fonctions principales Write, Read et Delete

Pertinence : le travail présenté répond-il à ce qui est demandé ? les réponses apportées sont-elles appropriées ?
L'architecture Client/Serveur n'est pas respectée.
Quelques points précis dans le code :

- (l36) Le choix de stockage du catalogue des "chunks" dans une hashmap est cohérent, cependant le type des <Key,Value> n'est pas bon. En effet, la clé est le nom du chunk et la valeur est le port de la machine contenant le chunk. Cependant, on peut aisément imaginer 2 machines différentes avec le même port.
- Fonction write : Mauvaise conception : le client doit dire au démon de faire le write, ce n'est pas à lui de le faire directement.
- (l58) La création du fichier ne dépend pas du paramètre passé dans la fonction write, ici le fichier est forcément au format FILE. Il faut pouvoir créer un fichier au format KV.
- (l107) Le port est choisi aléatoirement ce qui peut causer une exception qui arrête le programme si le port est occupé
- Fonction read : Mauvaise conception : Ce n'est pas au client de read directement sur les serveurs. Il doit envoyer une requête de read pour que le serveur lui envoie le fichier.
- Même problème pour la fonction Delete

Cohérence : le résultat obtenu a-t-il une structure, une architecture, une logique claires, les fonctions proposées et réalisées sont-elles bien complémentaires ou au contraire se recoupent-elles, font doublon ?
La structure de la classe HdfsClient est bonne, les fonctions proposées sont complémentaires et il n'y a pas de fonctions ou variables doublons. Cependant, l'architecture Client/Serveur est à revoir. Il y a quand même une cohérence intéressante avec le logiciel Hadoop : utilisation de chunks pour découper le fichier, prise en compte de futurs améliorations comme le facteur de réplication (= 1 dans cette version)

Améliorations possibles

- Changement du type de la valeur dans la hashmap du catalogue pour prendre en compte les cas d'utilisation sur un même port.
- Création d'une classe Serveur pour respecter l'architecture demandée

3 Annexe

Listing 1: La classe de test HdfsTest.java

```
package hdfs;
import org.junit.*;
import formats.Format;
import formats.KV;
import formats.KVFormat;
import formats.LineFormat;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;
import java.io.File;
import java.io.IOException;



/**
 * Classe de test de HDFS
 * @version $Revision$
 */
public class Test {
   HdfsClient hdfsClient;

   // nom d fichier pour les tests
   public final static String path =
       "/home/bgreaud/Anne_2/ProjetDonneesReparties/Hidoop/src/hdfs/";
   public final static String nomFichier = path + "filesampleLINE.txt";
       // fichier fourni dans le dossier data du projet
   public final static String nomFichier1 = path + "filesampleKV.txt";
       // fichier correspondant  filesample.txt-res dans le dossier data
   public final static String nomFichier2 = path + "exemple1.txt"; //
       fichier vide
   public final static String nomFichier3 = path + "tmp.txt";
   public final static String nomFichier4 = path + "res.txt";

   public final int tailleFichier = 1305, tailleChunck = 1024;

   @Before
   public void setUp() {
      // Construire un HdfsClient
      hdfsClient = new HdfsClient();
   }

   // ** TESTS SUR WRITE ** //
   // Test du nombre de fichier  gnrs  par HdfsWrite
   @org.junit.Test
```

```
public void testerWriteNombreFichierGeneres() {
   int compteur;

   // Nettoyage des ventuels fichiers  dj    gnrs
   File temp;
   for (int i = 0 ; (temp = new File(nomFichier + i)).exists() ; i++)
       {
      temp.delete();
   }

   //  Gnration  de fichiers de la fonction HdfsWrite - mode LINE
   hdfsClient.HdfsWrite(Format.Type.LINE, nomFichier, 1);

   // On compte le nombre de fichiers  gnrs  par le write
   for (compteur = 0 ; ((new File(nomFichier + compteur))).exists() ;
       compteur++);

   // Les fichiers   gnrs   font 1024 octets, on devrait donc obtenir
       2 fichiers avec le fichier exemple de 1305 octets
   assertEquals(compteur, 2);


   // Nettoyage des ventuels fichiers  dj    gnrs
   for (int i = 0 ; (temp = new File(nomFichier1 + i)).exists() ;
       i++) {
      temp.delete();
   }

   //  Gnration  de fichiers de la fonction HdfsWrite - Mode KV
   hdfsClient.HdfsWrite(Format.Type.KV, nomFichier1, 1);

   // On compte le nombre de fichiers  gnrs  par le write
   for (compteur = 0 ; ((new File(nomFichier1 + compteur))).exists()
       ; compteur++);

   // Les fichiers   gnrs   font 1024 octets, on devrait donc obtenir
       2 fichiers avec le fichier exemple de 1305 octets
   assertEquals(compteur, 2);
}


// Test de la taille
@org.junit.Test
public void testerTailleFichierGeneres() {
   long nbFichiersGeneres = this.tailleFichier / this.tailleChunck +
       1;
   boolean booleenTest = true;

   // Nettoyage des ventuels fichiers  dj    gnrs
   File temp;
```

```
   for (int i = 0 ; (temp = new File(nomFichier + i)).exists() ; i++)
       {
      temp.delete();
   }

   //  Gnration  de fichiers de la fonction HdfsWrite - mode LINE
   hdfsClient.HdfsWrite(Format.Type.LINE, nomFichier, 1);

   // On compte le nombre de fichiers  gnrs  par le write
   for (int i = 0 ; i < nbFichiersGeneres ; i++) {
      if (i != nbFichiersGeneres - 1) {
         if ((new File(nomFichier + i).length() !=
              (this.tailleChunck))) booleenTest = false;
      } else {
         if ((new File(nomFichier + i).length() != this.tailleFichier
              % this.tailleChunck)) booleenTest = false;
      }
   }
   assertTrue(booleenTest);
}


// Test avec un fichier inexistant
@org.junit.Test
public void testerWriteFichierInexistant() throws
    java.io.FileNotFoundException{
   String fichierInexistant = "a2ddk78SHaizbdyt76ksozie099999999.txt";
   (new File(fichierInexistant)).delete();
   // Appel de la fonction HdfsWrite sur le fichier inexistant
   hdfsClient.HdfsWrite(Format.Type.LINE, fichierInexistant, 1);
   // Il n'y a pas d'exception   rcuprer , on se contente donc de
       vrifier que l'appel  HdfsWrite n'a rien  gnr
   assertFalse((new File(fichierInexistant + '0')).exists());
}

// Test write fichier vide
@org.junit.Test
public void testerWriteFichierVide() {
   // Appel de la fonction HdfsWrite sur le fichier vide
   hdfsClient.HdfsWrite(Format.Type.LINE, nomFichier2, 1);
   // On  vrifie  qu'il ne  gnre  pas de fichiers
   assertFalse(new File(nomFichier2 + 0).exists());
   assertFalse(new File(nomFichier2 + 1).exists());


}

@org.junit.Test
public void testerRead(){
   // Appel de la fonction HdfsRead sur le fichier tmp.txt
```

```
      hdfsClient.HdfsRead(nomFichier3, nomFichier4);
      // On vrifie que le fichier res.txt existe
      assertTrue((new File(nomFichier4)).exists());
   }

   @org.junit.Test
   public void testerDelete(){
      // Nettoyage des ventuels fichiers  dj    gnrs
           File temp;
           for (int i = 0 ; (temp = new File(nomFichier + i)).exists() ;
               i++) {
              temp.delete();
           }
      // On appelle la fonction HdfsWrite en mode LINE
      hdfsClient.HdfsWrite(Format.Type.LINE, nomFichier, 1);
      // On appelle la fonction HdfsDelete pour supprimer les fichiers
          gnrs par le write
      hdfsClient.HdfsDelete(nomFichier3);
      // On vrifie que les fichiers ont bien t  supprims
      assertFalse((new File(nomFichier3 + '0')).exists());
      assertFalse((new File(nomFichier3 + '1')).exists());
   }


}
```