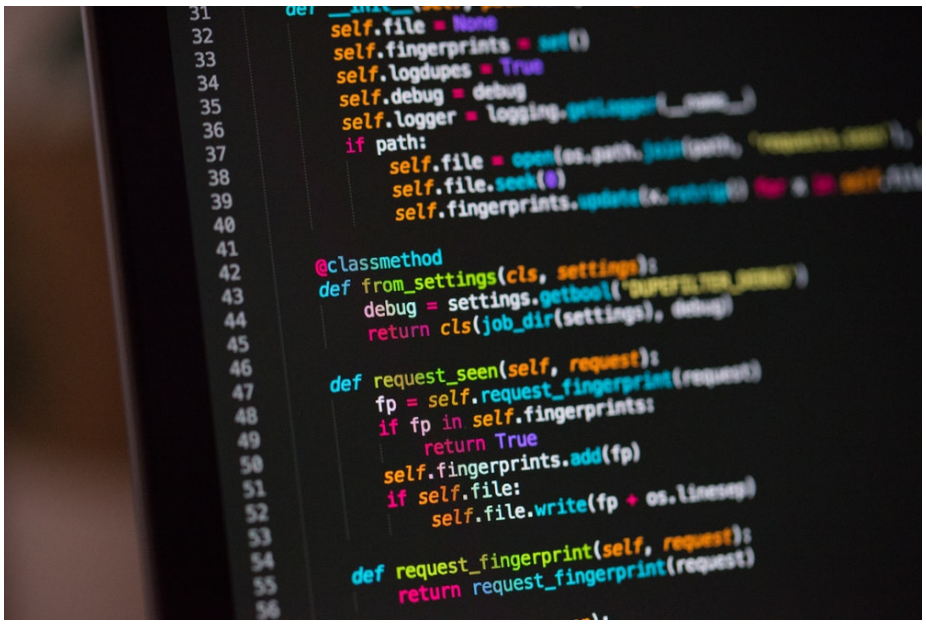
INP TOULOUSE ENSEEIHT

Évaluation du travail réalisé par le binôme complémentaire



Hamza Mouddene
Manal Hajji

23 décembre 2020

Table des matières

1 Partie technique

1.1 Architecture de la partie HDFS

La figure ci-dessous présente l'architecture complète de la partie HDFS, dans cette partie on va se restreindre dans l'explication juste sur la partie fonctionelle. Un HDFSClient qui lance les opérations un NameProvider qui prend les Requests lancés par le client pour vérifier si l'opération peut être traité, ou bien vérifier si les serveurs sont disponibles, etc... Mais l'architecutre reste plus riche que ce qui était présenté.

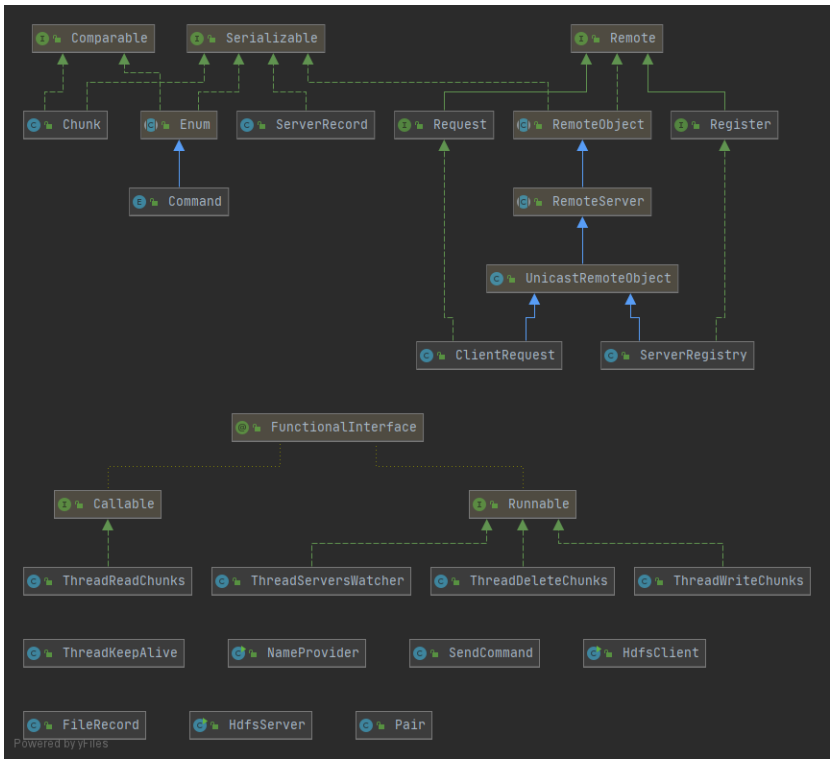


FIGURE 1 – Architecture HDFS

1.2 Présentation de la partie HDFS

Le système de fichier est composé d'un démon (HdfsServer) qui doit être lancé sur chaque machine. Une classe HdfsClient permet de manipuler les fichiers dans HDFS. Les opérations Read, Write et Delete ont été implémenté avec succès. Un jeu de tests a été fourni dans le répertoire test dans src qui testent les trois opérations précedemment citées. Plusieurs d'autres fonctionnalités ont été ajoutées très utiles pour le côté Hidoop comme récupérer le nombre

de chunks d'un fichier qu'on souhaite lui appliquer un map reduce ou bien lister les fichiers que le client peut lire sur le serveur.
Les tests fournies resteront rudimentaires et c'est le cas pour les tests de perfomance qui consistent à appliquer des opérations HDFS itérativement sur des fichiers, puis on obtient des fichiers de taille significative, qu'on manipule par le futur. Ces tests réussisent avec succès sans aucun problème, sur quoi on base notre critère de validité.

1.3 Bugs et qualité de code

Malgré la structure du code qui paraît suffisemment développée, elle manque de commentaires et de spécifications , ce qui rend le code très compliqué à comprendre et peu lisible. En ce qui concerne les bugs, ils existent une dizaine de warnings non vérifiées par le compilateur comme le montre la figure ci-dessous.

```
                                    ^
  required: E
  found: Pair
  where E is a type-variable:
    E extends Object declared in class ArrayList
src/hdfs/ClientRequest.java:74: warning: [unchecked] unchecked conversion
                readingInformations.add(new Pair(chunk.getValue().getRight().getLeft(), new Pair(chunk.getKey(), serverToRequest)));
                                        ^
  required: E
  found:    Pair
  where E is a type-variable:
    E extends Object declared in class ArrayList
src/hdfs/FileRecord.java:52: warning: [unchecked] unchecked call to Pair(l,r) as a member of the raw type Pair
        chunksHash.put(hash, new Pair<>(srToStore, new Pair(chunkNumber, totalNumberChunk)));
                                                   ^
  where l,r are type-variables:
    l extends Object declared in class Pair
    r extends Object declared in class Pair
src/hdfs/FileRecord.java:52: warning: [unchecked] unchecked method invocation: constructor <init> in class Pair is applied to given types
        chunksHash.put(hash, new Pair<>(srToStore, new Pair(chunkNumber, totalNumberChunk)));
                             ^
  required: l,r
  found: ArrayList<ServerRecord>,Pair
  where l,r are type-variables:
    l extends Object declared in class Pair
    r extends Object declared in class Pair
src/hdfs/FileRecord.java:52: warning: [unchecked] unchecked conversion
        chunksHash.put(hash, new Pair<>(srToStore, new Pair(chunkNumber, totalNumberChunk)));
                                                   ^
  required: r
  found:    Pair
  where r is a type-variable:
    r extends Object declared in class Pair
src/hdfs/FileRecord.java:52: warning: [unchecked] unchecked method invocation: method put in class HashMap is applied to given types
        chunksHash.put(hash, new Pair<>(srToStore, new Pair(chunkNumber, totalNumberChunk)));
                   ^
  required: K,V
  found: String,Pair
  where K,V are type-variables:
    K extends Object declared in class HashMap
    V extends Object declared in class HashMap
src/hdfs/FileRecord.java:52: warning: [unchecked] unchecked conversion
        chunksHash.put(hash, new Pair<>(srToStore, new Pair(chunkNumber, totalNumberChunk)));
                             ^
  required: V
  found:    Pair
  where V is a type-variable:
    V extends Object declared in class HashMap
src/hdfs/HdfsServer.java:66: warning: [unchecked] unchecked cast
           storedChunk = (HashMap<String, ChunkMetadata>) yaml.load(is);
                                                          ^
  required: HashMap<String,ChunkMetadata>
  found:    Object
10 warnings
```

FIGURE 2 – Warnings lors de la compilation

Le compilateur signale un autre problème lors de l'exécution du NameProvider concernant le port utiliser pour se connecter en RMI.

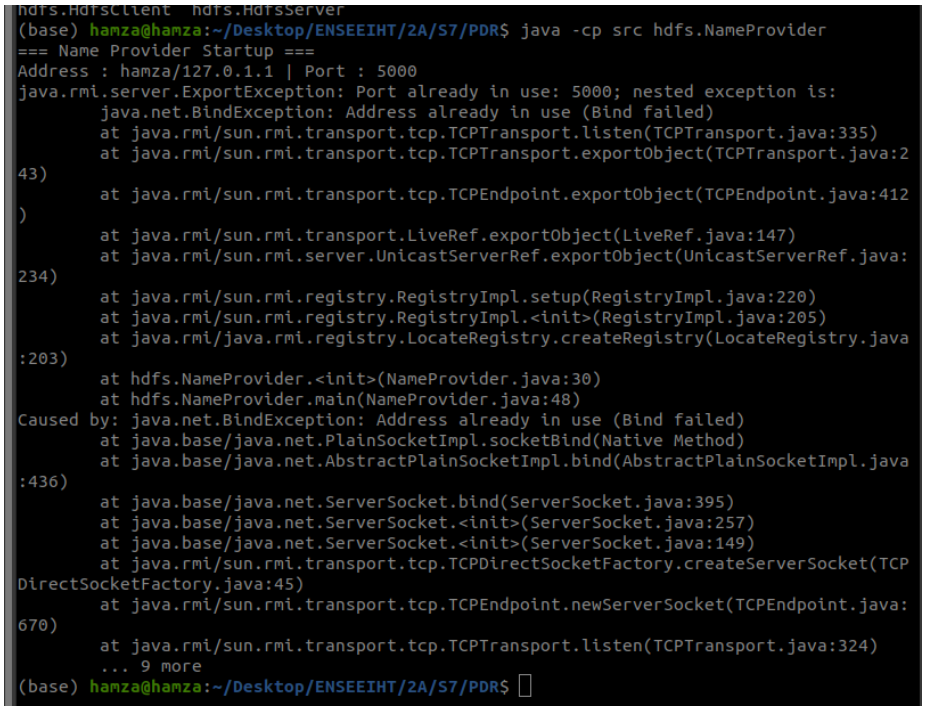


FIGURE 3 – Erreur lors de l'exécution du NameProvider

2 Synthèse

- Correction : Les résultats du produit sont relativement justes, pour confirmer la validité des résultats, il faut bien évidemment faire des tests complets et riches et ne pas se limiter sur des tests nominaux.
- Complétude : En général, tous les points de spécification sont traités ou en cours de traitement.
- Pertinence : le travail présenté répond à ce qui est demandé , les réponses apportées sont appropriées.
- Cohérence : le résultat obtenu a une structure qui respecte l'architecture de principe ainsi que la figure 1. Les fonctions proposées et réalisées sont complémentaires.
- Point d'amélioration : En générale, le produit est très bien refléchi mais le seul bémol c'est la documentation, comme le programme est un peu compliqué, faire le lien avec HDFS et HIDOOP a pris beaucoup de temps. Ainsi que le manque de tests unitaires, des tests de performances et de pertinences sont les bienvenus. On conseille plutôt d'utiliser Acoco et pitest pour faire des tests puissants et complets qui couvrent l'intégralité du code.

3 Annexes

```java
package test;
import formats.Format;
import hdfs.HdfsClient;
import hdfs.HdfsServer;
import hdfs.NameProvider;
import hdfs.Register;
import org.junit.Assert;
import org.junit.Test;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.rmi.NotBoundException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.concurrent.ExecutionException;
class testHdfsClient {
    public void testGeneral(int nb_serveurs, String nb_fichier) throws IOException, ExecutionException, InterruptedException, NotBoundException {
        Registry[] registries = new Registry[nb_serveurs];
        HdfsServer[] serveurs = new HdfsServer[nb_serveurs];
        for (int i = 0; i < nb_serveurs; i++) {
            registries[i] = LocateRegistry.getRegistry(NameProvider.NAME_PROVIDER_PORT);
            serveurs[i] = new HdfsServer((Register) registries[i].lookup("//localhost:" + NameProvider.NAME_PROVIDER_PORT + "/ServerRegistry"), "serverData", String.valueOf(i));
            serveurs[i].start();
        } //Lancement des serveurs
        String nom_destination = nb_fichier.replace(".txt","_read_result.txt");
        HdfsClient.HdfsWrite(Format.Type.LINE,nb_fichier,1); //Java HdfsClient write line nb_fichier
        HdfsClient.HdfsRead(nb_fichier,nom_destination); //Java HdfsClient read nb_fichier
        File file1 = new File(nb_fichier);
        File file2 = new File(nom_destination);
        Assert.assertTrue(file1.exists());
        Assert.assertTrue(file2.exists()); // On s'assure que le fichier généré existe
        Assert.assertEquals(file1.length(),file2.length()); //On s'assure que les deux fichiers ont la même taille
        FileInputStream fstream1 = new FileInputStream(nb_fichier);
        FileInputStream fstream2 = new FileInputStream(nom_destination);
        int i1 = fstream1.read();
        int i2 = fstream2.read();
        while (i1 != -1) {
            Assert.assertEquals(i1,i2); //On s'assure que les fichiers sont identiques, caractère pas caractère
            i1 = fstream1.read();
            i2 = fstream2.read();
        }
        fstream1.close();
        fstream2.close();
        HdfsClient.HdfsDelete(nb_fichier); //On supprime le fichier
    }
    @Test
    public void TresPetitFichiertest() throws IOException, ExecutionException, InterruptedException, NotBoundException {
        testGeneral(1,"Kilo.txt");
        testGeneral(2,"Kilo.txt");
        testGeneral(4,"Kilo.txt");
        testGeneral(8,"Kilo.txt");
        testGeneral(10,"Kilo.txt");
    }
}
```

FIGURE 4 – Read Write Delete test