

## Évaluation du travail réalisé par le binôme complémentaire

```
31 def __init__(self, request):
32     self.file = None
33     self.fingerprints = set()
34     self.logdupes = True
35     self.debug = debug
36     self.logger = logging.getLogger(__name__)
37     if path:
38         self.file = open(os.path.join(path, "requests.log"),
39                         "a")
40         self.file.seek(0)
41         self.fingerprints.update(self.request.headers)
42
43 @classmethod
44 def from_settings(cls, settings):
45     debug = settings.getbool("SUPERFINGER_DEBUG")
46     return cls(job_dir(settings), debug)
47
48 def request_seen(self, request):
49     fp = self.request_fingerprint(request)
50     if fp in self.fingerprints:
51         return True
52     self.fingerprints.add(fp)
53     if self.file:
54         self.file.write(fp + os.linesep)
55
56 def request_fingerprint(self, request):
57     return request_fingerprint(request)
```

Hamza Mouddene

Manal Hajji

23 décembre 2020



## Table des matières

<b>1</b>	<b>Partie technique</b>	<b>1</b>
1.1	Architecture de la partie HDFS . . . . .	1
1.2	Présentation de la partie HDFS . . . . .	1
1.3	Bugs et qualité de code . . . . .	2
<b>2</b>	<b>Synthèse</b>	<b>3</b>
<b>3</b>	<b>Annexes</b>	<b>4</b>

# 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 fonctionnelle. Un HDFSClient qui lance les opérations un NameProvider qui prend les Requests lancés par le client pour vérifier si l'opération peut être traitée, ou bien vérifier si les serveurs sont disponibles, etc... Mais l'architecture reste plus riche que ce qui était présenté.

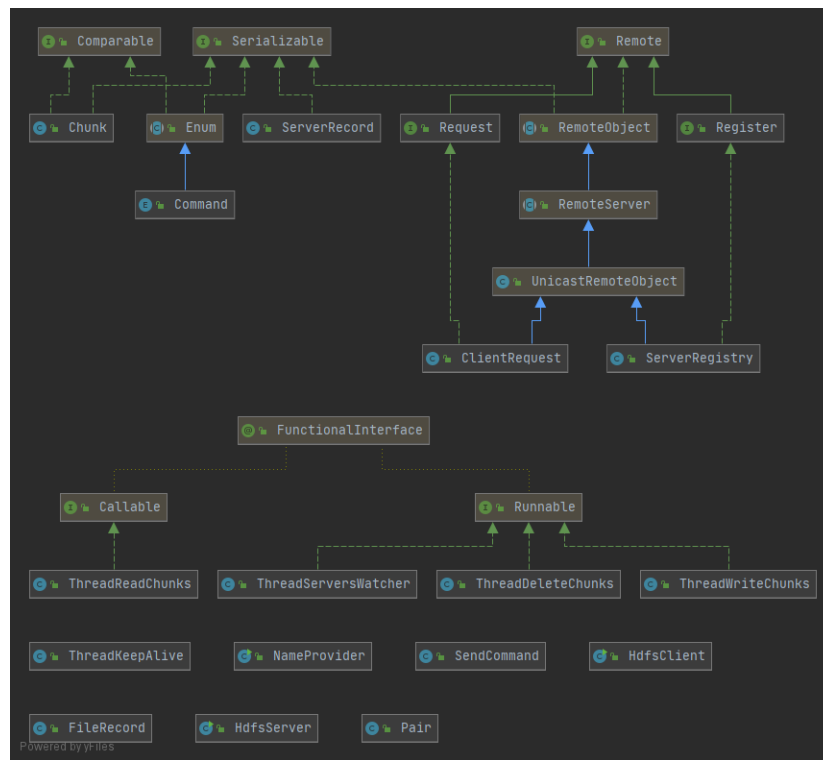


FIGURE 1 – Architecture HDFS

## 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 précédemment 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 fournis resteront rudimentaires et c'est le cas pour les tests de performance 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 réussissent 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 suffisamment 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.

```

hdfs.HdfsClient hdfs.HdfsServer
(base) hamza@hamza:~/Desktop/ENSEEIH/2A/S7/PDR$ java -cp src hdfs.NameProvider
=== Name Provider Startup ===
Address : hamza/127.0.1.1 | Port : 5000
java.rmi.server.ExportException: Port already in use: 5000; nested exception is:
    java.net.BindException: Address already in use (Bind failed)
    at java.rmi/sun.rmi.transport.tcp.TCPTransport.listen(TCPTransport.java:335)
    at java.rmi/sun.rmi.transport.tcp.TCPTransport.exportObject(TCPTransport.java:2
43)
    at java.rmi/sun.rmi.transport.tcp.TCPEndpoint.exportObject(TCPEndpoint.java:412
)
    at java.rmi/sun.rmi.transport.LiveRef.exportObject(LiveRef.java:147)
    at java.rmi/sun.rmi.server.UnicastServerRef.exportObject(UnicastServerRef.java:
234)
    at java.rmi/sun.rmi.registry.RegistryImpl.setup(RegistryImpl.java:220)
    at java.rmi/sun.rmi.registry.RegistryImpl.<init>(RegistryImpl.java:205)
    at java.rmi/java.rmi.registry LocateRegistry.createRegistry(LocateRegistry.java
:203)
    at hdfs.NameProvider.<init>(NameProvider.java:30)
    at hdfs.NameProvider.main(NameProvider.java:48)
Caused by: java.net.BindException: Address already in use (Bind failed)
    at java.base/java.net.PlainSocketImpl.socketBind(Native Method)
    at java.base/java.net.AbstractPlainSocketImpl.bind(AbstractPlainSocketImpl.java
:436)
    at java.base/java.net.ServerSocket.bind(ServerSocket.java:395)
    at java.base/java.net.ServerSocket.<init>(ServerSocket.java:257)
    at java.base/java.net.ServerSocket.<init>(ServerSocket.java:149)
    at java.rmi/sun.rmi.transport.tcp.TCPDirectSocketFactory.createServerSocket(TCP
DirectSocketFactory.java:45)
    at java.rmi/sun.rmi.transport.tcp.TCPEndpoint.newServerSocket(TCPEndpoint.java:
670)
    at java.rmi/sun.rmi.transport.tcp.TCPTransport.listen(TCPTransport.java:324)
    ... 9 more
(base) hamza@hamza:~/Desktop/ENSEEIH/2A/S7/PDR$

```

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 réfléchi mais le seul bémol c'est la documentation, comme le programme est un peu compliqué, faire le lien avec HDFS et HADOOP 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

```

1 package test;
2 import formats.Format;
3 import hdfs.HdfsClient;
4 import hdfs.HdfsServer;
5 import hdfs.NameProvider;
6 import hdfs.Register;
7 import org.junit.Assert;
8 import org.junit.Test;
9 import java.io.File;
10 import java.io.FileInputStream;
11 import java.io.IOException;
12 import java.rmi.NotBoundException;
13 import java.rmi.registry.LocateRegistry;
14 import java.rmi.registry.Registry;
15 import java.util.concurrent.ExecutionException;
16 class TestHdfsClient {
17     public void testGeneral(int nb_serveurs, String nb_fichier) throws IOException, ExecutionException, InterruptedException, NotBoundException {
18         Registry[] registries = new Registry[nb_serveurs];
19         HdfsServer[] serveurs = new HdfsServer[nb_serveurs];
20         for (int i = 0; i < nb_serveurs; i++) {
21             registries[i] = LocateRegistry.getRegistry(NameProvider.NAME_PROVIDER_PORT);
22             serveurs[i] = new HdfsServer((Register) registries[i].lookup("//localhost:" + NameProvider.NAME_PROVIDER_PORT + "/ServerRegistry"), "serverData", String.valueOf(i));
23             serveurs[i].start();
24         }
25         String non_destination = nb_fichier.replace(".txt", "_read_result.txt");
26         HdfsClient.HdfsWrite(format.Type.LINE, nb_fichier, i); //java HdfsClient write line nb_fichier
27         HdfsClient.HdfsRead(nb_fichier, non_destination); //java HdfsClient read nb_fichier
28         File file1 = new File(nb_fichier);
29         File file2 = new File(non_destination);
30         Assert.assertTrue(file1.exists());
31         Assert.assertTrue(file2.exists());
32         Assert.assertEquals(file1.length(), file2.length()); //On s'assure que les deux fichiers ont la même taille
33         FileInputStream fstream1 = new FileInputStream(nb_fichier);
34         FileInputStream fstream2 = new FileInputStream(non_destination);
35         int i1 = fstream1.read();
36         int i2 = fstream2.read();
37         while ((i1 != -1) {
38             Assert.assertEquals(i1, i2); //On s'assure que les fichiers sont identiques, caractère par caractère
39             i1 = fstream1.read();
40             i2 = fstream2.read();
41         }
42         fstream1.close();
43         fstream2.close();
44         HdfsClient.HdfsDelete(nb_fichier); //On supprime le fichier
45     }
46     @Test
47     public void TestPetitFichierTest() throws IOException, ExecutionException, InterruptedException, NotBoundException {
48         testGeneral(1, "kilo.txt");
49         testGeneral(1, "kilo.txt");
50         testGeneral(1, "kilo.txt");
51         testGeneral(1, "kilo.txt");
52         testGeneral(1, "kilo.txt");
53     }
54 }

```

FIGURE 4 – Read Write Delete test