

ÉCOLE NATIONALE SUPÉRIEURE  
D'ÉLECTROTECHNIQUE, D'ÉLECTRONIQUE, D'INFORMATIQUE,  
D'HYDRAULIQUE ET DES TÉLÉCOMMUNICATIONS

INSTITUT NATIONAL POLYTECHNIQUE



---

# Rapport final de Hidoop V0

## Partie HDFS

---

Chailou SALAH EDDINE  
Chla AYMEN

# SOMMAIRE

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Vue globale</b>	<b>1</b>
2.1	Architecture . . . . .	1
2.2	HdfsClient . . . . .	1
2.3	HdfsServer . . . . .	2
2.4	NameNode . . . . .	3
<b>3</b>	<b>Les formats des fichiers</b>	<b>3</b>
<b>4</b>	<b>Classes supplémentaires</b>	<b>4</b>
<b>5</b>	<b>Exemples d'exécutions</b>	<b>5</b>
5.1	Lancement d'HDFS . . . . .	5
5.2	Les scénarios d'exécution possibles . . . . .	6
<b>6</b>	<b>Conclusion</b>	<b>8</b>

# 1 Introduction

Ce projet consiste à développer une version minimale de la plateforme hadoop, qui est un framework open source qui repose sur Java. Hadoop prend en charge le traitement des données volumineuses (Big Data) au sein d'environnements informatiques distribués. Cette version sera nommée Hidoop et fournira les services de base en termes de gestion des accès à des fichiers fragmentés, et d'ordonnancement pour les tâches map-reduce. Durant cette phase du projet, nous nous occuperons de la réalisation d'un service HDFS permettant un accès massive concurrent à de larges volumes de données.

## 2 Vue globale

### 2.1 Architecture

Notre service (HDFS) contient les classes suivantes :

- HdfsClient : son rôle est d'établir une connexion avec un noeud centrale nommé **Name Node** afin de récupérer les adresses des **Data Nodes** nécessaires pour l'aboutissement d'une commande (Read, Write ou Delete).
- HdfsServeur : qui une fois déployé sur un serveur s'enregistre au près du noeud centrale **Name Node** pour être accessible par la suite afin d'exécuter les tâches demandés par une application utilisant HDFS.
- NameNode : est le serveur centrale appelé aussi serveur maître, il permet de coordonné la communication entre un pool de serveur HDFS et l'application, il gère aussi un catalogue de système de fichiers (méta-données et l'emplacement des fragments de chaque fichier).

### 2.2 HdfsClient

Le client HDFS commence par charger la configuration nécessaire pour son fonctionnement à partir d'un fichier de configuration contenant l'adresse IP, le port et le nom du serveur centrale (name node).

Le client HDFS pourra par la suite communiquer avec les serveurs (data nodes qui gèrent les chunks) pour exécuter des commandes en utilisant les sockets en mode de communication TCP.

Afin d'exécuter ces commandes le client HDFS doit tout d'abord contacter le Name Node en utilisant RMI. Ce dernier transmet les métadonnées nécessaires

pour l'exécution de la commande.

Chaque interaction entre le client et le serveur HDFS est initiée par l'envoi d'un objet commande spécifiant la commande à exécuter par HdfsServer.

Cette objet commande contient les champs suivant :

- Le type de la commande : Il s'agit d'une énumération qui prend la valeur (CMD\_READ, CMD\_WRITE ou CMD\_DELETE).
- une chaîne de caractère représentant le nom du chunk à récupérer ou à sauvegarder.
- Le format du fichier à manipuler. Dans notre projet nous avons manipuler deux formats LINE et KV.

La classe HdfsClient peut être utilisée depuis la ligne de commande ou directement depuis du code pour lire, écrire ou supprimer des fichiers :

- HdfsWrite(Format.Type fmt, String localFSSourceFname, int repFactor) : permet d'écrire un fichier dans HDFS. Le fichier localFSSourceFname est lu sur le système de fichiers local, découpé en fragments (soit de taille fixe, soit pour équilibrer la charge entre les machines) et les fragments sont envoyés pour être stocker sur les différentes machines, fmt est le format du fichier (Format.Type.LINE ou Format.Type.KV), repFactor est le facteur de duplication des fragments ; pour cette version il sera considéré comme valant 1 (pas de duplication).
- HdfsRead(String hdfsFname, String localFSDestFname) : permet de lire un fichier de nom hdfsFname à partir de HDFS. Les fragments du fichier sont lus à partir des différentes machines, concaténés et stockés localement dans un fichier nommé localFSDestFname. Pour simplifier nous avons adopté une convention de nommage pour les différents chunks d'un fichier dans HDFS les différents chunks porteront le nom "hdfsFname"+i avec i un entier déterminant l'ordre des chunks d'un fichier donné.
- HdfsDelete(String hdfsFname) : permet de supprimer les fragments d'un fichier stocké dans HDFS.

## 2.3 HdfsServer

HdfsServer est un démon qui est déployé sur tout les serveurs du cluster HDFS. Une fois lancés, ces derniers chargent le fichier de configuration nécessaires à leur fonctionnement devront par la suite s'enregistrer au près du Name Node pour être visibles est accessibles à partir du client HDFS.

Les serveurs HDFS communiquent ensuite avec le client HDFS via les sockets pour exécuter les commandes demandées.

Chaque interaction entre le client et le serveur HDFS est initié par l'envoi d'un objet commande spécifiant la commande à exécuter par HdfsServer.

## 2.4 NameNode

Le NameNode gère les méta-données du système de fichiers : pour chaque fichier, une structure analogue à un i-nœud regroupe les informations utiles à la gestion du fichier.

Cette structure comporte en particulier :

- Le nom du fichier
- la taille du fichier
- le format du fichier
- la taille des blocs (chunks)
- la liste des identifiants de blocs (chunk handles) constituant le fichier
- le facteur de duplication des blocs du fichier : chacun des blocs devront être en effet dupliqué et ses copies devront être distribuées sur différents Data-Nodes. Pour cette version il ne sera pas pris en compte (pas de duplication)

Le NameNode implémente plusieurs méthodes qui peuvent être utilisées à distance en utilisant RMI depuis le HdfsClient et le pool de serveurs HDFS.

Parmi ces méthodes on peut citer :

- `addMetaDataFile(MetadataFile metadata)` et `MetadataFile getMetaDataFile(String fileName)` qui permettent de gérer les métadonnées.
- `List<DataNodeInfo> getDaemons()` et `addDaemon(DataNodeInfo info)` qui permettent d'enregistrer ou de récupérer les démons en cours d'exécution.

## 3 Les formats des fichiers

Il existe deux formats utilisés, chaque format est définie dans une classe, avec des méthodes qui permettent la lecture/écriture en assurant la cohérence. Un fichier de structures ne peut pas être coupé au milieu d'une structure. Dans notre cas, on ne considère que 2 formats : le format texte (LINE) et le format Clé-Valeur (KV), mais d'autres formats peuvent être envisagés et introduits par la suite. Une classe implantant un format implante l'interface `Format`. Le format KV correspond au format des données intermédiaires produites durant l'exécution de l'application MapReduce

## 4 Classes supplémentaires

Dans cette partie, nous introduiront des classes secondaires mais néanmoins nécessaires au bon fonctionnement du système HDFS.

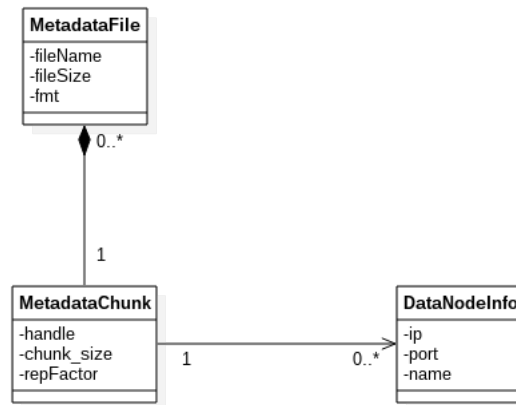


FIGURE 1 – Gestion métadata

**MetadataFile** Cette classe sert à stocker les métadonnées des fichiers enregistrés sur HDFS elle contient les attributs suivants :

- Le nom du fichier
- la taille du fichier
- le format du fichier
- une liste de type **MetadataChunk**

**MetadataChunk** Sert à stocker les métadonnées des chunks enregistrés sur les data nodes elle contient les attributs suivants :

- Le nom du chunk
- la taille du chunk
- le facteur de réplication
- un attribut de type **DataNodeInfo**

**DataNodeInfo** Cette classe contient les informations nécessaires pour communiquer avec les data nodes à savoir l'adresse ip et le port.

**Commande** Cette classe regroupe les informations nécessaires pour initier l'interaction entre **HdfsClient** et **HdfsServer**.

## 5 Exemples d'exécutions

Dans cette section nous présenterons quelques cas concrets de l'utilisation d'HDFS en ligne de commande.

### 5.1 Lancement d'HDFS

Pour l'utilisation du HDFS on commence tout d'abord par lancer le RmiRegistry suivi du nameNodeImpl qui sera accessible via RMI. De la même manière on déploie les DataNodes sur le pool de serveurs via la commande (HdfsServer ip port). Après avoir lancé ces derniers un client HDFS pourra alors communiquer avec le nameNode et les dataNodes pour exécuter les commandes d'écriture de lecture ou de suppression de fichiers :

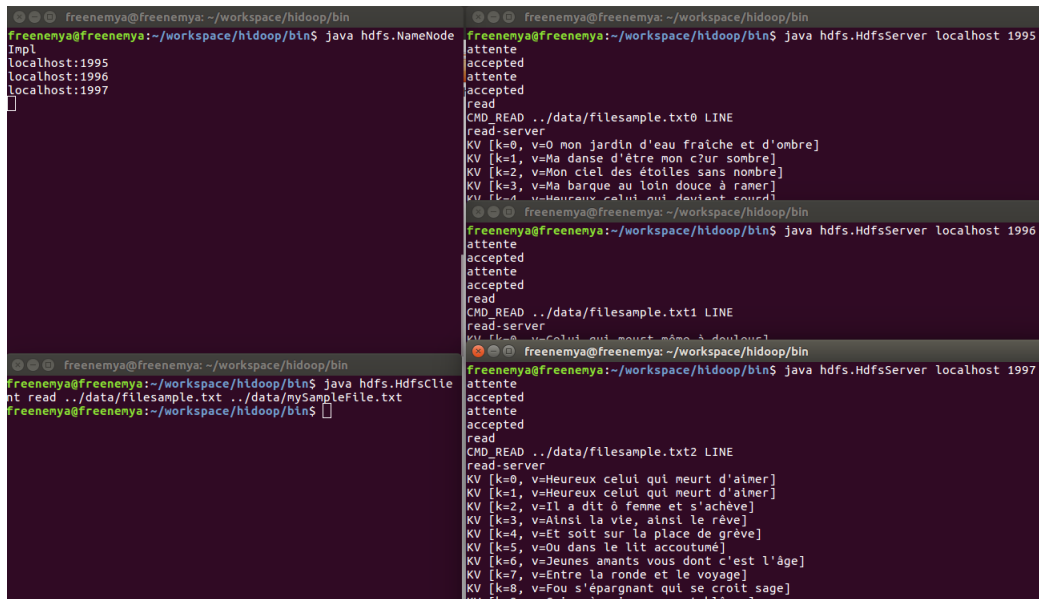
- `java HdfsClient read <file> <destfile>`
- `java HdfsClient write <line|kv> <file>`
- `java HdfsClient delete <file>`

Où `file` est le nom du fichier, `destfile` est le nouveau nom du fichier lu à partir de HDFS et finalement `line/kv` sont les formats utilisés pour les fichiers.

## 5.2 Les scénarios d'exécution possibles

Voici les commandes possibles :

- **HdfsClient read fileName** : Cette commande permet de récupérer les méta-données correspondantes au fichier fileName, puis il lit les différents chunks de fichier et stocke le résultat dans un fichier.



```

freemyma@freemyma:~/workspace/hadoop/bin$ java hdfs.Namenode
Inpl
localhost:1995
localhost:1996
localhost:1997
[]

freemyma@freemyma:~/workspace/hadoop/bin$ java hdfs.HdfsServer localhost 1995
attente
accepted
attente
accepted
read
CMD_READ ../data/filesample.txt0 LINE
read-server
KV [k=0, v=0 mon jardin d'eau fraîche et d'ombre]
KV [k=1, v=Ma danse d'être mon cœur sombre]
KV [k=2, v=Mon ciel des étoiles sans nombre]
KV [k=3, v=Ma barque au loin douce à ramer]
KV [k=4, v=Heureux celui qui devient sourd]

freemyma@freemyma:~/workspace/hadoop/bin$ java hdfs.HdfsServer localhost 1996
attente
accepted
attente
accepted
read
CMD_READ ../data/filesample.txt1 LINE
read-server
KV [k=0, v=Heureux celui qui meurt d'aimer]
KV [k=1, v=Heureux celui qui meurt d'aimer]
KV [k=2, v=Il a dit ô femme et s'achève]
KV [k=3, v=Ainsi la vie, ainsi le rêve]
KV [k=4, v=Et soit sur la place de grève]
KV [k=5, v=Ou dans le lit accoutumé]
KV [k=6, v=Jeunes amants vous dont c'est l'âge]
KV [k=7, v=Entre la ronde et le voyage]
KV [k=8, v=Fou s'épargnant qui se croit sage]
KV [k=9, v=Entre la ronde et le voyage]

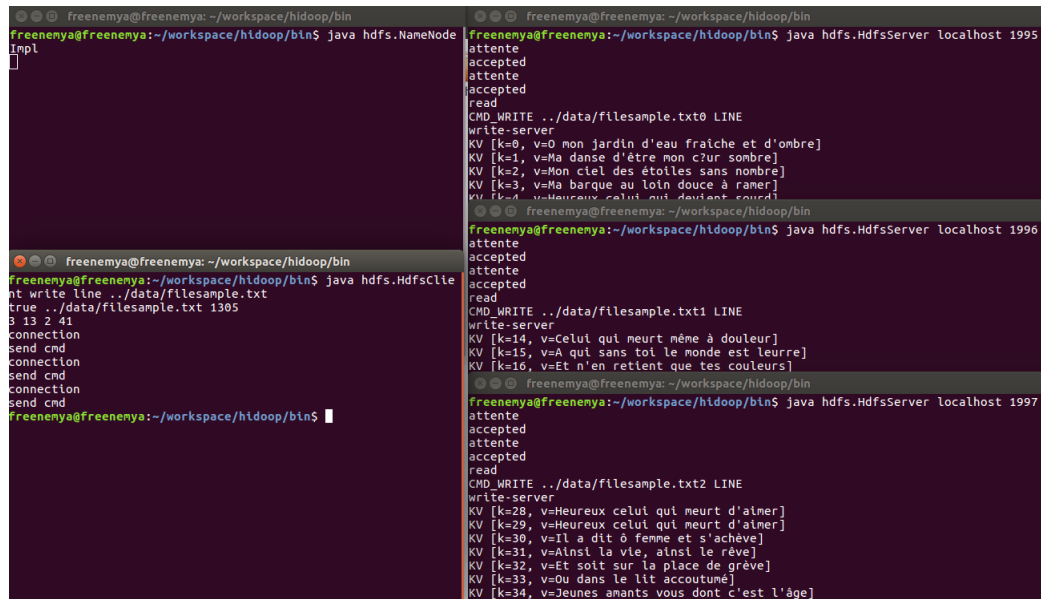
freemyma@freemyma:~/workspace/hadoop/bin$ java hdfs.HdfsClient read ../data/filesample.txt ../data/mysamplefile.txt
freemyma@freemyma:~/workspace/hadoop/bin$

```

FIGURE 2 – HDFS Read

- **HdfsClient write line fileName** : Cette commande permet d'enregistrer le fichier fileName de format LINE dans les DataNodes disponibles, les informations relatives à ces derniers sont chargées par le client, qui les envoie des chunks un par un jusqu'à la fin du fichier, ces chunks ont pour nom "filename" suivi du i avec i le numéro de chunk. Après l'enregistrement, les méta-données sont mises à jour. Un message "ce fichier n'existe pas" est affiché si le fichier est introuvable en local.





```

freenemya@freenemya: ~/workspace/hadoop/bin
freenemya@freenemya:~/workspace/hadoop/bin$ java hdfs.NameNode
Impl
[]

freenemya@freenemya:~/workspace/hadoop/bin$ java hdfs.HdfsClient
nt write line ../data/filesample.txt
true ../data/filesample.txt 1305
3 13 2 41
connection
send cmd
connection
send cmd
connection
send cmd
freenemya@freenemya:~/workspace/hadoop/bin$

freenemya@freenemya:~/workspace/hadoop/bin$ java hdfs.HdfsServer localhost 1995
attente
accepted
attente
accepted
read
CMD_WRITE ../data/filesample.txt0 LINE
write-server
KV [k=0, v=0 mon jardin d'eau fraîche et d'ombre]
KV [k=1, v=Ma danse d'être mon c?ur sombre]
KV [k=2, v=Mon ciel des étoiles sans nombre]
KV [k=3, v=Ma barque au loin douce à ramer]
KV [k=4, v=Mon cœur palpitant meurt d'attente]

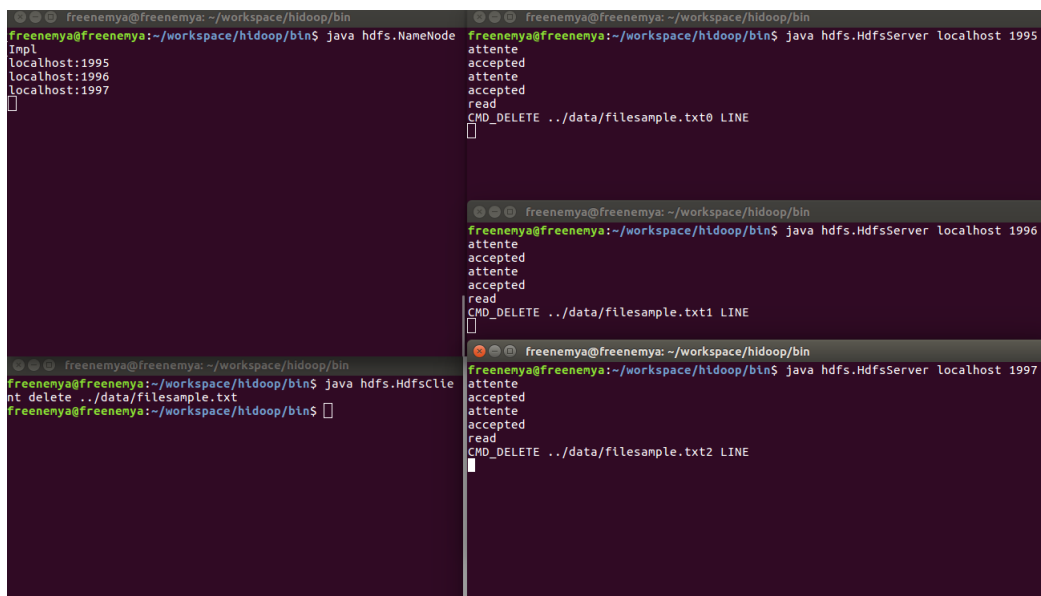
freenemya@freenemya:~/workspace/hadoop/bin$ java hdfs.HdfsServer localhost 1996
attente
accepted
attente
accepted
read
CMD_WRITE ../data/filesample.txt1 LINE
write-server
KV [k=14, v=Celui qui meurt même à douleur]
KV [k=15, v=A qui sans toi le monde est leurre]
KV [k=16, v=Et n'en retient que tes couleurs]

freenemya@freenemya:~/workspace/hadoop/bin$ java hdfs.HdfsServer localhost 1997
attente
accepted
attente
accepted
read
CMD_WRITE ../data/filesample.txt2 LINE
write-server
KV [k=28, v=Heureux celui qui meurt d'aimer]
KV [k=29, v=Heureux celui qui meurt d'aimer]
KV [k=30, v=Il a dit ô femme et s'achève]
KV [k=31, v=Ainsi la vie, ainsi le rêve]
KV [k=32, v=Et soit sur la place de grève]
KV [k=33, v=Ou dans le lit accoutumé]
KV [k=34, v=Jeunes amants vous dont c'est l'âge]

```

FIGURE 3 – HDFS Write

- **HdfsClient delete fileName** : Cette commande permet de supprimer tous les chunks dans les différents DataNodes relative au fichier fileName, et mettre à jour les méta-données après la suppression. Un message "ce fichier n'existe pas" est affiché si le fichier est introuvable dans HDFS.



```

freenemya@freenemya: ~/workspace/hadoop/bin
freenemya@freenemya:~/workspace/hadoop/bin$ java hdfs.NameNode
Impl
localhost:1995
localhost:1996
localhost:1997
[]

freenemya@freenemya:~/workspace/hadoop/bin$ java hdfs.HdfsClient
nt delete ../data/filesample.txt
freenemya@freenemya:~/workspace/hadoop/bin$

freenemya@freenemya:~/workspace/hadoop/bin$ java hdfs.HdfsServer localhost 1995
attente
accepted
attente
accepted
read
CMD_DELETE ../data/filesample.txt0 LINE
[]

freenemya@freenemya:~/workspace/hadoop/bin$ java hdfs.HdfsServer localhost 1996
attente
accepted
attente
accepted
read
CMD_DELETE ../data/filesample.txt1 LINE
[]

freenemya@freenemya:~/workspace/hadoop/bin$ java hdfs.HdfsServer localhost 1997
attente
accepted
attente
accepted
read
CMD_DELETE ../data/filesample.txt2 LINE
[]

```

FIGURE 4 – HDFS Delete

## 6 Conclusion

Après avoir eu l'évaluation de notre binôme complémentaire de la version V0 de notre projet, nous avons pu résoudre les problèmes d'intégrations et améliorer cette première version en rajoutant un fichier de configuration.