

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

INSTITUT NATIONAL POLYTECHNIQUE



---

# Rapport d'évaluation du HDFS

---

Mohamed MAGHFOUR  
Iliass MOUMEN

## SOMMAIRE

<b>1</b>	<b>Partie Technique</b>	<b>1</b>
1.1	Résultats des tests . . . . .	1
1.2	Validation des performances . . . . .	1
1.3	Qualité du code . . . . .	2
<b>2</b>	<b>Synthèse</b>	<b>2</b>
2.1	Correction . . . . .	2
2.2	Complétude . . . . .	2
2.3	Pertinence . . . . .	2
2.4	Cohérence . . . . .	3
<b>3</b>	<b>Pistes d'amélioration</b>	<b>3</b>
<b>4</b>	<b>Annexe</b>	<b>4</b>
4.1	Test des fragmentations . . . . .	4
4.2	Résultats des fichiers fragments . . . . .	5

# 1 Partie Technique

## 1.1 Résultats des tests

On a commencé par lancer le serveur et tester le fonctionnement de HDFSread et HDFSwrite en partant d'un seul client. On remarque, que le fonctionnement du programme réalisé respecte le schéma présenté dans le sujet du projet. Le programme est assez pertinent au niveau de l'utilisation de la programmation concurrente. La communication entre les DataNodes(HdfsServeur) et le NameNode se fait par RMI pour envoyer les MetaDatas qui contiennent les informations sur les serveurs. De plus, La communication entre les HdfsClients et le NameNode se base sur RMI pour envoyer les différents DataNodes (serveurs) disponibles pour exécuter les différentes méthodes (Read and Write), Ainsi que les HdfsClients et les DataNodes se communiquent en utilisant les sockets pour interagir. Un bon point de plus, c'est que les NameNode peut servir plusieurs HdfsClient en même temps grâce à l'utilisation des threads.

Lors de la partie des tests plus approfondi, on a constaté les remarques suivantes :

- La demande de lecture et d'écriture d'un fichier inexistant lance une exception et interrompt le fonctionnement du programme. Ce problème peut être résolu par un message envoyé au client.
- Nous avons testé le fonctionnement de la fragmentation par des fichiers différents, le résultat était toujours positif en l'appliquant sur les différents formats (KV et Line).
- L'absence d'un fichier de configuration qui récapitule les informations génériques sur le programme à savoir les adresses des serveurs, du NameNode et les ports.

## 1.2 Validation des performances

Le code fourni fonctionne bien en vue d'ensemble. La fragmentation des fichiers sous les différents formats se fait correctement. On peut aussi lire les fragments et les détruire par l'intermédiaire du serveur et du NameNode. Pourtant, le code ne gère pas les NullPointerException.

Le système fonctionne principalement par les sockets et RMI, ce qui nécessite la bonne gestion des ouvertures et des fermetures des liens de communications.

Cela est pris en considération dans les classes `HdfsClient` et `HdfsServeur`, où il y a une seule ouverture/fermeture du fichier à traiter dans toutes les méthodes, ce qui rend le système assez performant.

### 1.3 Qualité du code

On constate que seulement les deux classes `HdfsClient` et `HdfsServer` sont commentées, mais le code en général est bien structuré et organisé. On remarque aussi que les noms des variables sont significatifs.

## 2 Synthèse

### 2.1 Correction

Après plusieurs tests, on constate que le code fourni fonctionne correctement et les résultats étaient conformes aux ce qui était prévu.

### 2.2 Complétude

Le programme fourni répond aux spécifications mentionnées dans le sujet du projet. Les grands points de la partie HDFS sont abordés et traités dans le programme fourni, à savoir la fragmentation d'un fichier sur plusieurs serveurs en maintenant sa cohérence lors de sa récupération et la gestion de l'aspect concurrent et communicant entre plusieurs serveurs, plusieurs clients et le `NameNode` simultanément.

### 2.3 Pertinence

Les points les plus pertinents après l'évaluation du code fourni sont :  
Dans l'implémentation du `NameNode`, il peut servir plusieurs clients en même temps grâce à l'utilisation des `Threads` au niveau du `NameNode`. De plus, il gère aussi les serveurs dynamiquement via RMI pour récupérer les informations des

différents DataNodes. Pour la communication entre les clients et les serveurs, le code du client se base surtout sur les sockets pour pouvoir transmettre et récupérer les résultats.

## 2.4 Cohérence

Dans HdfsRead, on constate que la taille des chunks est calculée à partir de nombres de DataNodes qu'on a lancé, ce qui peut poser des problèmes de performances si on a par exemple un fichier qui contient une seule ligne, ou bien un fichier de grande taille et un seul serveur. Donc, pour une meilleure optimisation, il fallait donner une taille fixe aux chunks (64 ko par exemple).

Dans plusieurs méthodes de HdfsServer et HdfsClient, on a constaté qu'il y a plusieurs duplications dans la gestion des exceptions. On peut simplement résoudre ce problème par une seule "Exception e", ou bien ajouter des commentaires significatifs à chacune d'elles.

Dans Formats, la lecture et l'écriture sont bien optimisées par l'utilisation du BufferedReader et PrintWriter. Il s'agit d'une implémentation propre, simple et efficace qui sera facilement réutilisable. Ce qui est très positif.

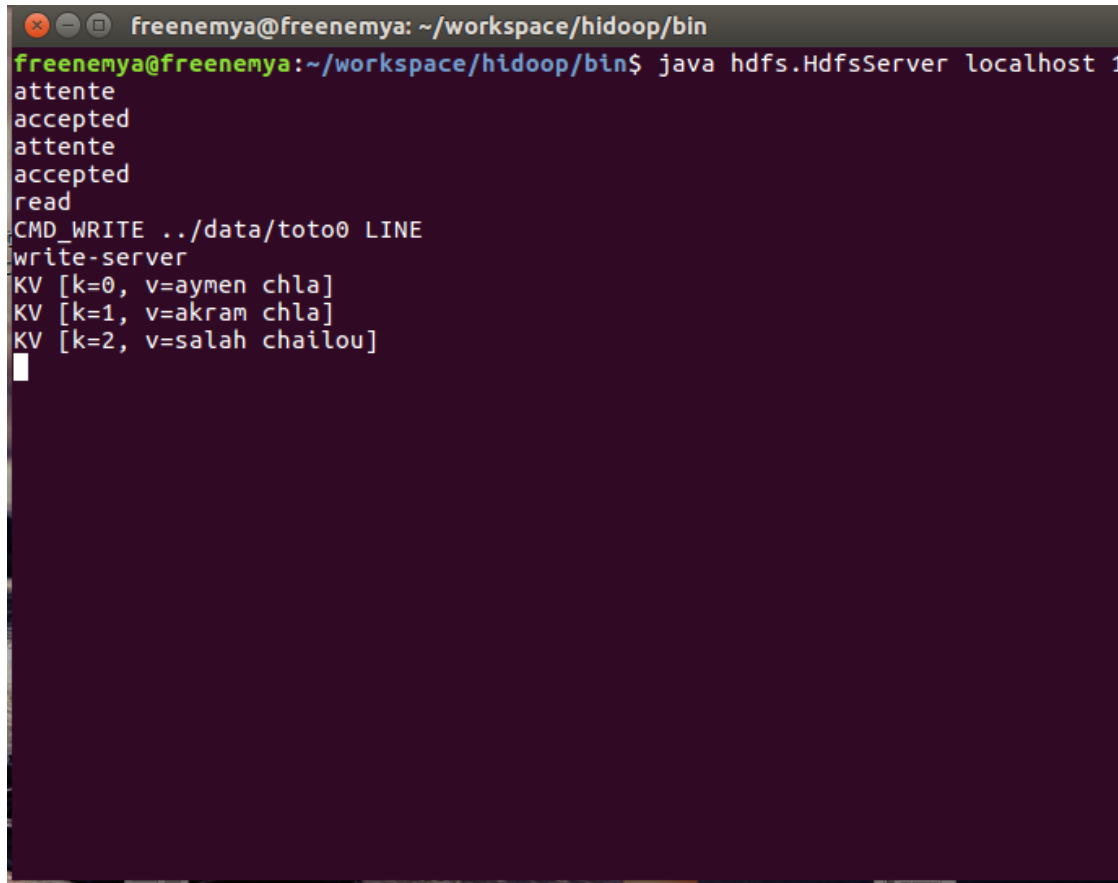
## 3 Pistes d'amélioration

Voici quelques améliorations possibles :

Améliorations	Gain
Ajouter une nouvelle classe qui traite la communication par les sockets. cette classe gère les différentes parties qui se répète à chaque fois.	Augmenter la clarté du code, et éviter d'écrire à chaque fois le même code dans les différentes.
Ajouter des fichiers dans le dossier config contenant les différentes informations à propos des adresses et des ports.	Stocker les paramètres de configuration du programme.
Dans la fonctionnalité de fragmentation, il vaut mieux se baser sur une taille fixe du chunk pour diviser notre fichier au lieu de se baser sur son nombre de lignes et le nombre de serveurs disponibles.	optimiser l'utilisation des serveurs disponibles, laisser d'autres serveurs disponibles pour servir d'autres clients.

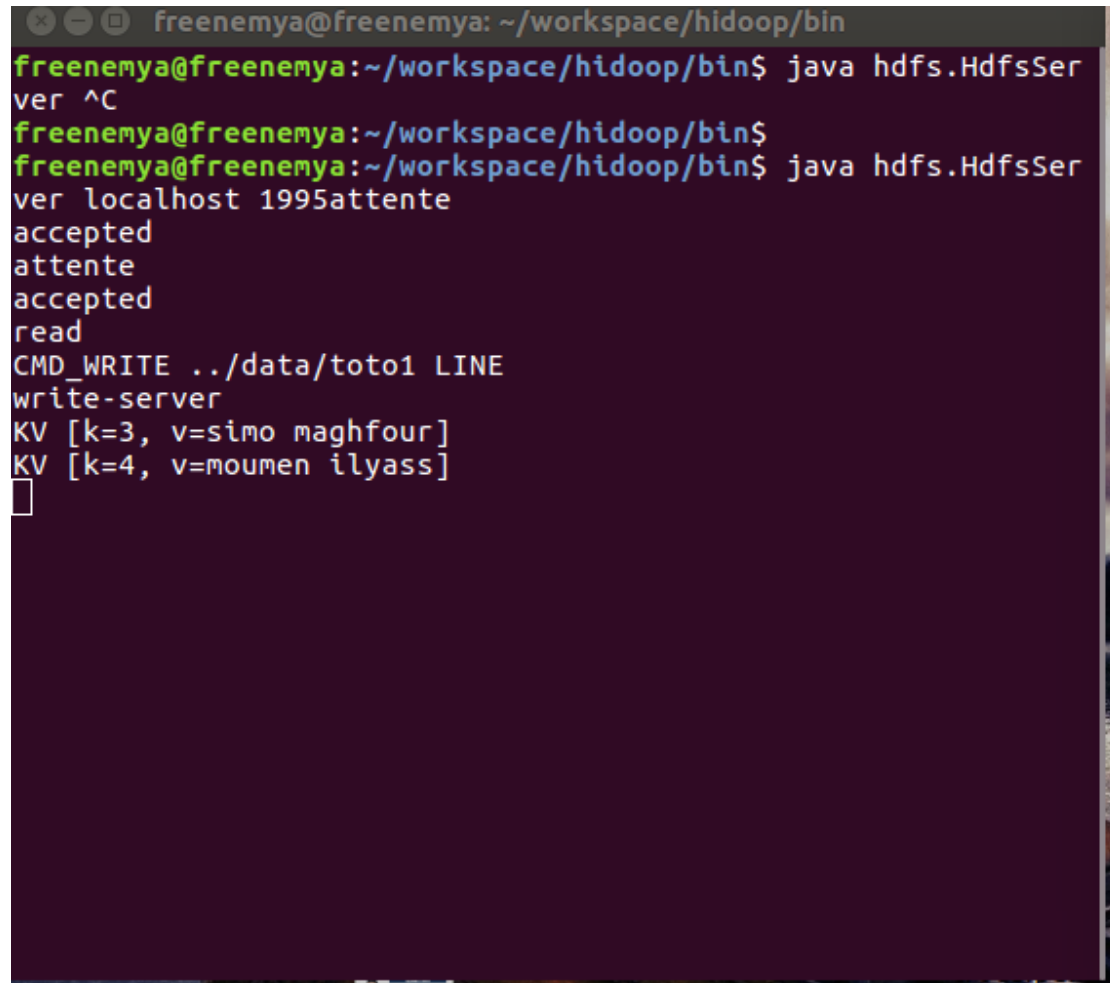
## 4 Annexe

### 4.1 Test des fragmentations



```
freenemya@freenemya: ~/workspace/hadoop/bin
freenemya@freenemya:~/workspace/hadoop/bin$ java hdfs.HdfsServer localhost 1
attente
accepted
attente
accepted
read
CMD_WRITE ../data/toto0 LINE
write-server
KV [k=0, v=aymen chla]
KV [k=1, v=akram chla]
KV [k=2, v=salah chailou]
```

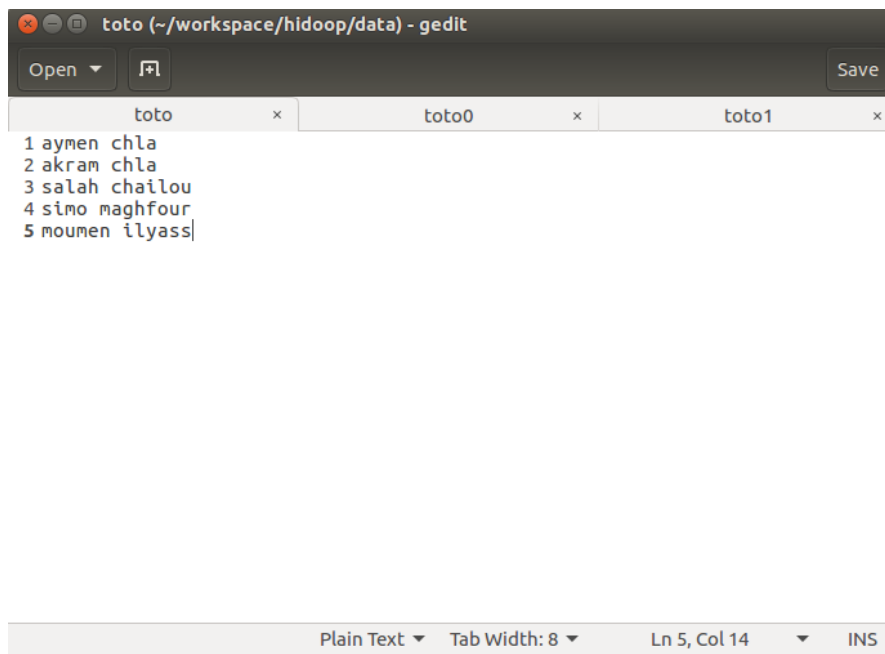
FIGURE 1 – Test des fragmentations

A terminal window with a dark background and light-colored text. The prompt is 'freenemya@freenemya: ~/workspace/hadoop/bin'. The user enters 'java hdfs.HdfsServer ^C', then 'java hdfs.HdfsServer localhost 1995', and then 'attente'. The output shows 'accepted', 'attente', 'accepted', 'read', 'CMD\_WRITE ../data/toto1 LINE', 'write-server', 'KV [k=3, v=simo maghfour]', and 'KV [k=4, v=moumen ilyass]'.

```
freenemya@freenemya: ~/workspace/hadoop/bin
freenemya@freenemya:~/workspace/hadoop/bin$ java hdfs.HdfsSer
ver ^C
freenemya@freenemya:~/workspace/hadoop/bin$
freenemya@freenemya:~/workspace/hadoop/bin$ java hdfs.HdfsSer
ver localhost 1995attente
accepted
attente
accepted
read
CMD_WRITE ../data/toto1 LINE
write-server
KV [k=3, v=simo maghfour]
KV [k=4, v=moumen ilyass]
█
```

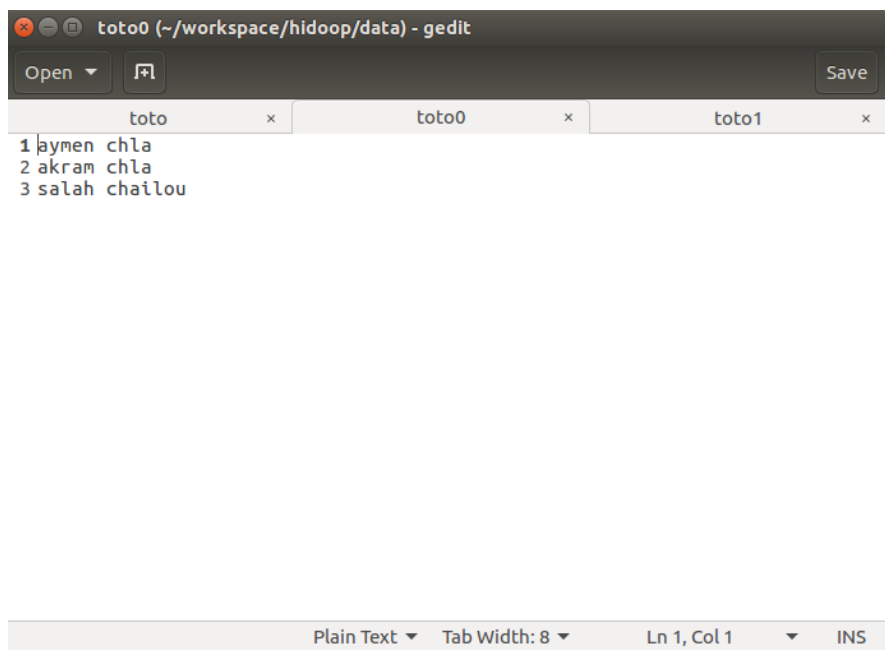
FIGURE 2 – Test des fragmentations

## 4.2 Résultats des fichiers fragments



```
toto (~/.workspace/hadoop/data) - gedit
Open Save
toto x toto0 x toto1 x
1 aymen chla
2 akram chla
3 salah chailou
4 simo maghfour
5 moumen ilyass|
Plain Text Tab Width: 8 Ln 5, Col 14 INS
```

FIGURE 3 – Le fichier source



```
toto0 (~/.workspace/hadoop/data) - gedit
Open Save
toto x toto0 x toto1 x
1 aymen chla
2 akram chla
3 salah chailou
Plain Text Tab Width: 8 Ln 1, Col 1 INS
```

FIGURE 4 – Premier fragment



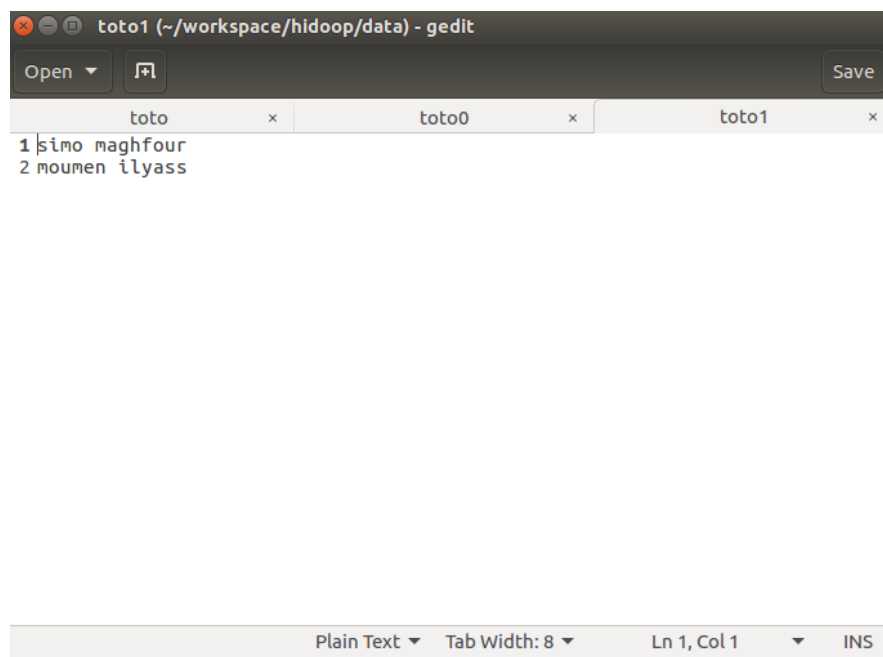


FIGURE 5 – Deuxième fragment