

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

INSTITUT NATIONAL POLYTECHNIQUE



Rapport Final Hadoop V.0

Partie Map/Reduce

Mohamed MAGHFOUR
Iliass MOUMEN

SOMMAIRE

1	Conception de la Solution	2
1.1	Classe Job	2
1.2	Classe Daemon	2
1.3	Classe CallBack	3
1.4	Classe MapThread	3
2	Intégration Hdfs et MapReduce	3
3	Correction et Difficultés	4
3.1	Correction	4
3.2	Difficultés	4
4	Tests et résultats	5
4.1	Lancement des Daemons	5
4.2	Exécution de la fonction Map	6
4.3	Exécution de la fonction Reduce	8
5	Conclusion	9

Table des figures

1	Lancement des Daemons	5
2	Exécution de la fonction main :MapReduce	6
3	Résultats des map	7
4	Le fichier résultat final	8

Introduction

Le but de ce projet est d'implanter une version simplifiée de la plateforme Hadoop. Cette version consiste à appliquer des fonctions MAP/REDUCE sur des fichiers distants stockés sur plusieurs serveurs HDFS. Ce rapport présente donc l'ensemble des tâches réalisées en précisant l'intérêt de chaque solution adoptée ainsi que les remarques prises en compte après l'évaluation faite par l'autre binôme.

1 Conception de la Solution

1.1 Classe Job

La classe Job est la classe de base dans notre projet. Elle permet d'instancier les fichiers avec lesquels on travaille, lancer les Maps, concaténer les résultats et appliquer Reduce.

La méthode qui gère cette étape est `startJob()`, et elle se base sur les étapes suivantes :

- Instancier les fichiers `input`, `intermédiaire`, `resultatMap`, `resultatReduce` qui servent respectivement à créer le fichier d'entrée pour chaque Daemon qui s'occupe de lui appliquer le Map, à stocker temporairement le résultat de chaque Map et de rassembler les résultats des Map dans un seul fichier pour appliquer reduce et générer le fichier `resultat` final.
- Récupérer les Daemons lancés dynamiquement pour appliquer le Map sur chaque Daemon et pour chaque fichier résultant à partir de Hdfs (Hdfs est la partie de l'autre groupe). Le lancement de Map se fait en parallèle en utilisant la programmation avec les Threads. Chaque Daemon "`i`" va stocker son résultat map dans un fichier `nomFichierInitial_inter"i"`.
- Après la fin de l'exécution de tous les maps (détectés par `callBack`), on lance la concaténation de tous les fichiers dans un seul grâce à la fonction `HdfsRead` (réalisée par l'autre groupe).
- Enfin on exécute la fonction `reduce` sur notre fichier précédent et on stocke le résultat dans le fichier `resultat`.

La récupération des adresses et des ports des Daemons se fait d'une manière dynamique pour pouvoir gérer un nombre quelconque des daemons lancées.

1.2 Classe Daemon

Le Daemon c'est le programme qui va tourner sur chaque machine et qui s'occupe d'appliquer le map. Dans cette méthode on communique directement l'adresse, le port et le nom avec le serveur `NameNode` qui va stocker les informations de chaque daemon pour en servir dans la fonction `startJob()` (savoir les daemons qui tournent à l'instant). Le rôle de Daemon est donc de lancer le `run-Map()`. Il exécute le Map sur un `Reader` et enregistre le résultat sur le `writer`.

1.3 Classe Callback

La classe Callback nous permet de savoir si les daemons ont terminé les maps. Cela revient à utiliser un sémaphore (nombre de daemons lancés). A chaque fois qu'un daemon termine on désincrémente notre sémaphore. Grâce donc à cette classe, on bloque le programme (BRIDGE) en attendant que les maps finissent.

1.4 Classe MapThread

Cette Classe garantit que le lancement des maps sur chaque daemon soit en parallèle. On utilise ici les threads (processus léger) a chaque application d'un map sur un daemon. En conséquence, on évite le fait d'attendre un daemon qu'il termine son map pour lancer le prochain.

2 Intégration Hdfs et MapReduce

L'intégration de la partie MapReduce avec Hdfs (réalisé par l'autre groupe) était un peu délicate, surtout pour la fonction Hdfsread(). L'autre groupe voulait que la méthode HdfsRead() puisse réassembler soit les fichiers après fragmentation et en même temps être capable de réassembler les résultats des maps. pour cela, on a décider d'ajouter un paramètre dans la méthode hdfsRead : booléen isMap pour savoir si celui qui a appelé la fonction s'agit d'un MapReduce et d'un appel simple de la fonction pour réassembler les fichier fragmentés. Ce problème vient de le fait que les fichiers de fragmentation sont stockés avec un FileName bien déterminé (nomFichier"i" Avec i le numéro de la fragmentation) pourtant nos fichiers intermédiaires sont stockés sous la forme (nomFichier_inter"i" Avec i le numéro du Daemon).

3 Correction et Difficultés

3.1 Correction

Après l'évaluation de l'autre binôme, on a effectué plusieurs changements à notre code, spécialement les classes `DaemonImpl` et `Job`. La première modification consiste à lancer les Daemons d'une façon dynamique, cela était possible par l'intégration du `NameNode` dans notre classe `Job`, et stocker les différentes informations des Daemons à partir de l'intermédiaire `DataNodeInfo` qui stocke l'adresse IP, port et le nom de chaque Daemon directement dans le `NameNode`.

Pareil pour la classe `Job`, on a changé le code de telle sorte que la méthode `startJob` puisse récupérer la liste des Daemons lancés via le `NameNode` pour y appliquer la fonction `Map`.

Une autre remarque qu'on a rectifiée c'est d'ajouter des fichiers dans le dossier config contenant les informations sur le `NameNode` à savoir l'adresse IP et le port pour une meilleure synchronisation entre les deux parties du projet.

3.2 Difficultés

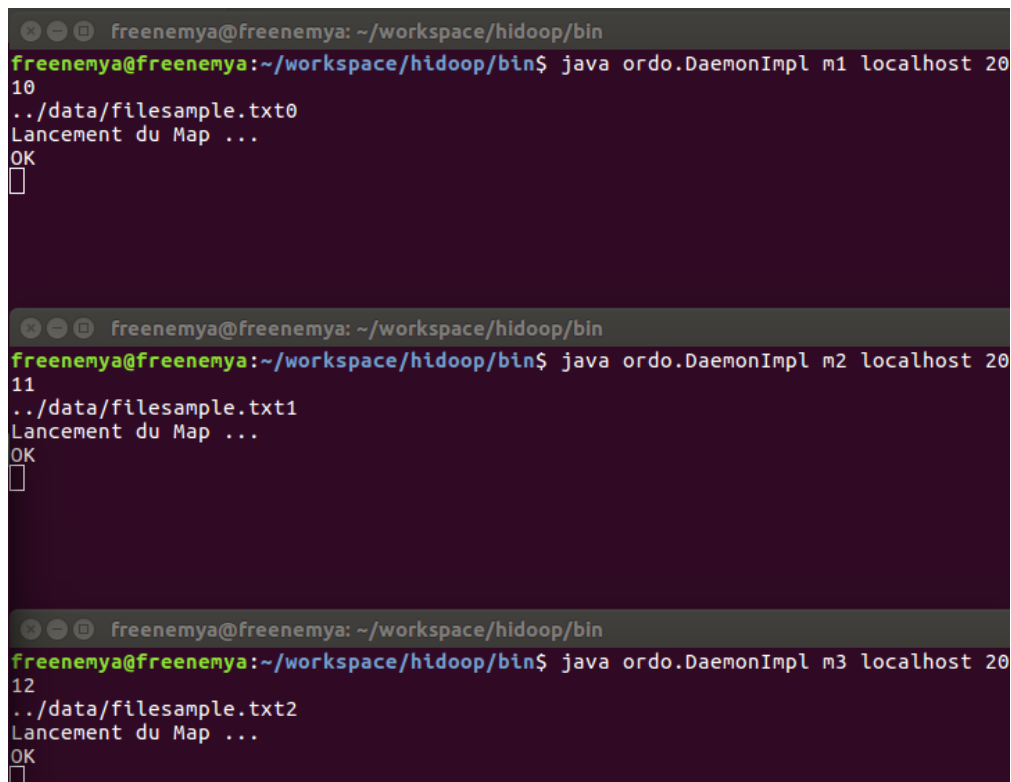
Lors de la phase de test surtout après l'intégration des deux parties du programme (avec l'autre binôme), on a rencontré beaucoup de problèmes et de bugs concernant les formats et l'intégration des méthodes de la classe `HDFS` dans notre bout de code. En effet, notre première version n'avait pas pris en considération une implémentation directe du `NameNode` pour les Daemons, ainsi que des problèmes de nomination des fichiers, et par conséquent la méthode `HdfsRead` n'a pas fonctionné correctement. Heureusement l'autre binôme ont modifié cette méthode et nous avons réussi à surmonter ce bug.

4 Tests et résultats

Cette partie présente les résultats de cette version Hidoop. Les deux parties sont lancées sur une machine (les Hdfs, les Daemons, NameNode ...) mais la structure avec laquelle notre programme est fait garantit le bon fonctionnement aussi sur plusieurs machines sans problème grâce au NameNode qui stocke les informations sur chaque machine (communication entre Daemon/DataNode et le NameNode).

4.1 Lancement des Daemons

On lance 3 Daemons pour tester notre partie du programme à partir de trois fragments du fichier initiale "filesample.txt" fait par l'autre binôme.



```
freemyma@freemyma: ~/workspace/hidoop/bin
freemyma@freemyma:~/workspace/hidoop/bin$ java ordo.DaemonImpl m1 localhost 20
10
../data/filesample.txt0
Lancement du Map ...
OK
[ ]

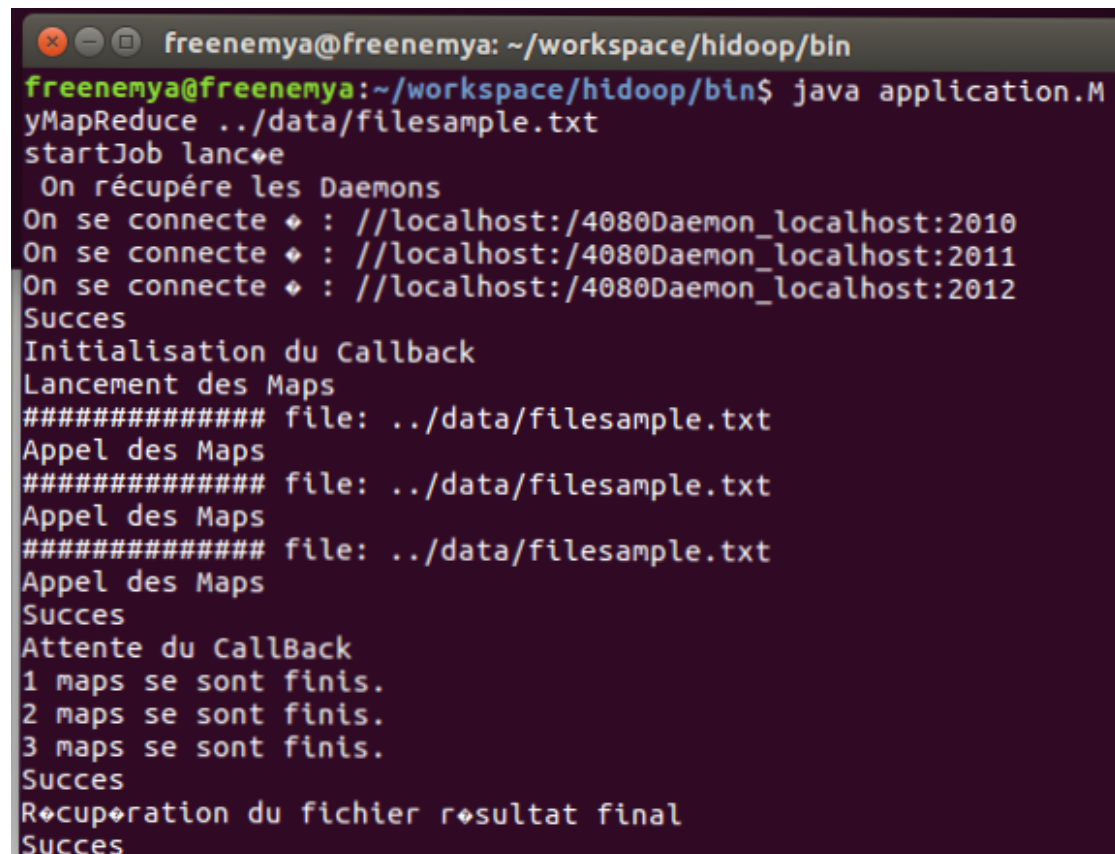
freemyma@freemyma: ~/workspace/hidoop/bin
freemyma@freemyma:~/workspace/hidoop/bin$ java ordo.DaemonImpl m2 localhost 20
11
../data/filesample.txt1
Lancement du Map ...
OK
[ ]

freemyma@freemyma: ~/workspace/hidoop/bin
freemyma@freemyma:~/workspace/hidoop/bin$ java ordo.DaemonImpl m3 localhost 20
12
../data/filesample.txt2
Lancement du Map ...
OK
[ ]
```

FIGURE 1 – Lancement des Daemons

4.2 Exécution de la fonction Map

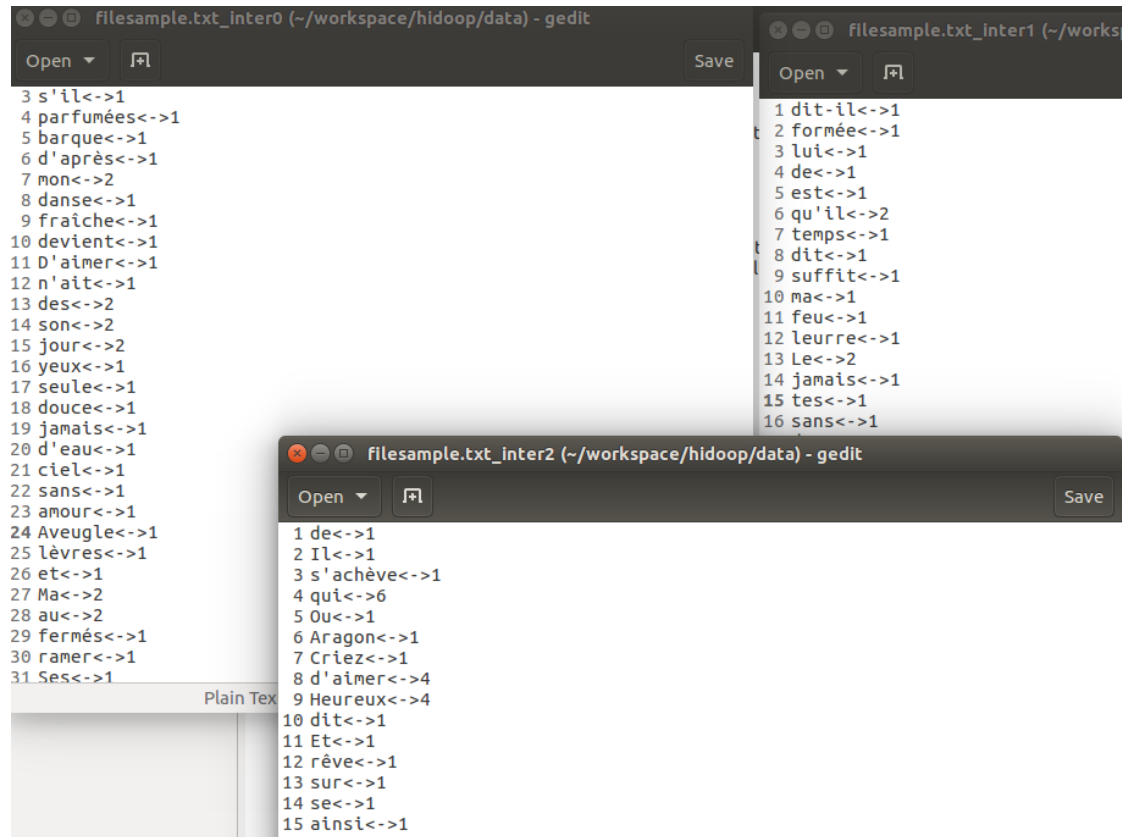
L'exécution de la fonction main dans le fichier "MyMapReduce" donné, comme on est en local, tous les résultats seront mis dans un même dossier.



```
freenemya@freenemya: ~/workspace/hadoop/bin
freenemya@freenemya:~/workspace/hadoop/bin$ java application.M
yMapReduce ../data/filesample.txt
startJob lancé
  On récupère les Daemons
On se connecte ♦ : //localhost:/4080Daemon_localhost:2010
On se connecte ♦ : //localhost:/4080Daemon_localhost:2011
On se connecte ♦ : //localhost:/4080Daemon_localhost:2012
Succes
Initialisation du Callback
Lancement des Maps
##### file: ../data/filesample.txt
Appel des Maps
##### file: ../data/filesample.txt
Appel des Maps
##### file: ../data/filesample.txt
Appel des Maps
Succes
Attente du Callback
1 maps se sont finis.
2 maps se sont finis.
3 maps se sont finis.
Succes
Récupération du fichier résultat final
Succes
```

FIGURE 2 – Exécution de la fonction main :MapReduce

Ces trois fichiers représentent les résultats de la fonction map sur chaque fragment.



```
filesample.txt_inter0 (~/.workspace/hadoop/data) - gedit
3 s'il<->1
4 parfumées<->1
5 barque<->1
6 d'après<->1
7 mon<->2
8 danse<->1
9 fraîche<->1
10 devient<->1
11 D'aimer<->1
12 n'ait<->1
13 des<->2
14 son<->2
15 jour<->2
16 yeux<->1
17 seule<->1
18 douce<->1
19 jamais<->1
20 d'eau<->1
21 ciel<->1
22 sans<->1
23 amour<->1
24 Aveugle<->1
25 lèvres<->1
26 et<->1
27 Ma<->2
28 au<->2
29 fermés<->1
30 ramer<->1
31 Ses<->1

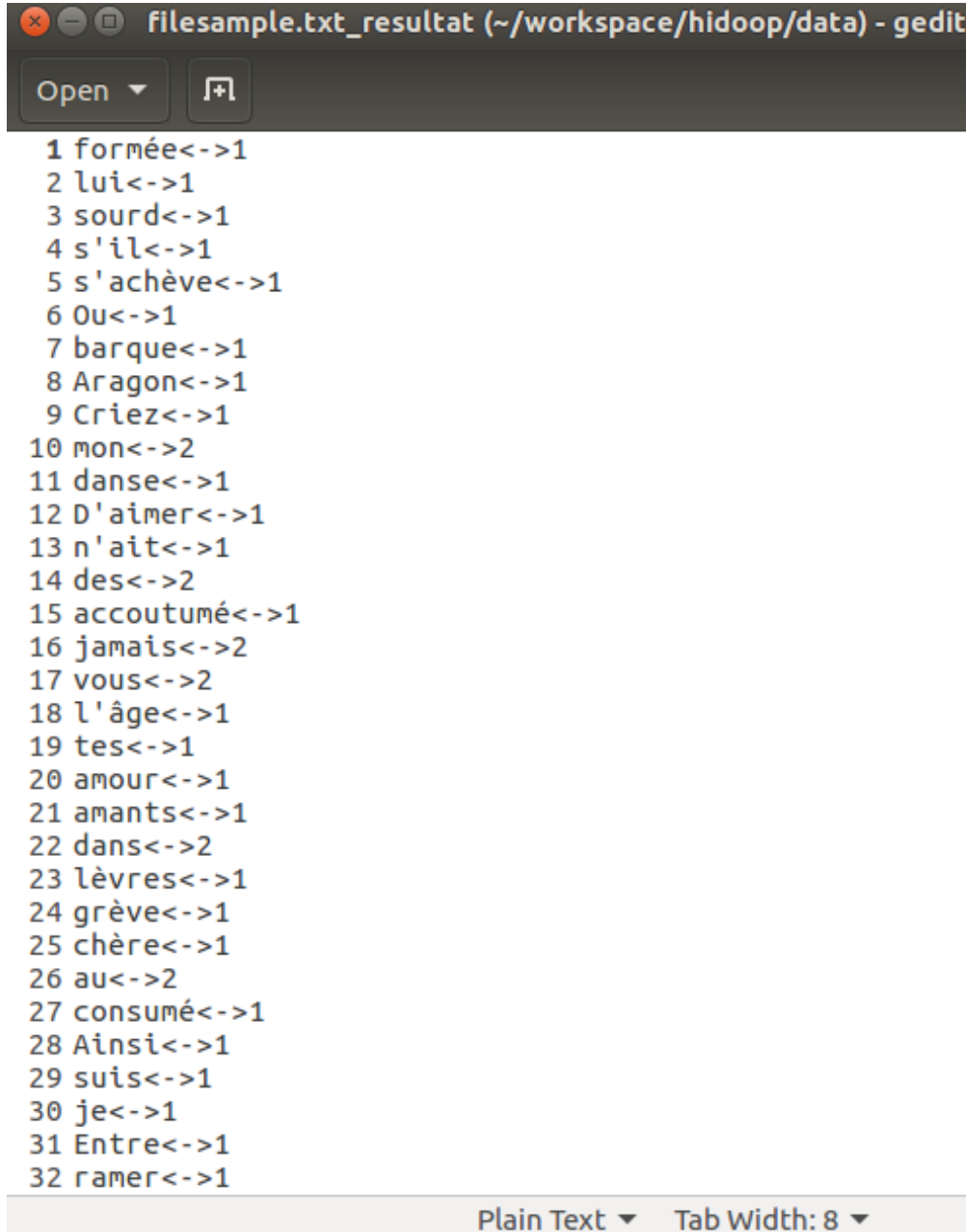
filesample.txt_inter1 (~/.workspace/hadoop/data) - gedit
1 dit-il<->1
2 formée<->1
3 lui<->1
4 de<->1
5 est<->1
6 qu'il<->2
7 temps<->1
8 dit<->1
9 suffit<->1
10 ma<->1
11 feu<->1
12 leurre<->1
13 Le<->2
14 jamais<->1
15 tes<->1
16 sans<->1

filesample.txt_inter2 (~/.workspace/hadoop/data) - gedit
1 de<->1
2 Il<->1
3 s'achève<->1
4 qui<->6
5 Ou<->1
6 Aragon<->1
7 Criez<->1
8 d'aimer<->4
9 Heureux<->4
10 dit<->1
11 Et<->1
12 rêve<->1
13 sur<->1
14 se<->1
15 ainsi<->1
```

FIGURE 3 – Résultats des map

4.3 Exécution de la fonction Reduce

Le fichier résultat final "filesample.txt-résultat" est identique au fichier donné "-res", ce qui montre que le programme fonctionne correctement.



```
filesample.txt_résultat (~/.workspace/hadoop/data) - gedit
Open ▾ [New File Icon]
1 formée<->1
2 lui<->1
3 sourd<->1
4 s'il<->1
5 s'achève<->1
6 Ou<->1
7 barque<->1
8 Aragon<->1
9 Criez<->1
10 mon<->2
11 danse<->1
12 D'aimer<->1
13 n'ait<->1
14 des<->2
15 accoutumé<->1
16 jamais<->2
17 vous<->2
18 l'âge<->1
19 tes<->1
20 amour<->1
21 amants<->1
22 dans<->2
23 lèvres<->1
24 grève<->1
25 chère<->1
26 au<->2
27 consommé<->1
28 Ainsi<->1
29 suis<->1
30 je<->1
31 Entre<->1
32 ramer<->1
Plain Text ▾ Tab Width: 8 ▾
```

FIGURE 4 – Le fichier résultat final

5 Conclusion

En guise de conclusion, ce projet nous a permis en général de mettre en pratique nos connaissances en programmation concurrente et en intergiciel. En effet, ce projet nous a donné l'opportunité de mettre en place un programme qui se base sur les appels distants via RMI et les sockets, ainsi que l'utilisation des Threads pour garantir le parallélisme dans notre programme.

De plus, ce projet nous a montré l'efficacité du travail en collaboration, surtout que l'autre équipe a montré un haut niveau de professionnalisme, et cela nous a permis de travailler ensemble en toute cohésion et efficacité.