



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

SYSTÈMES CONCURRENTS ET COMMUNICANTS

**Projet Hadoop: Implantation et utilisation
du modèle Map-Reduce**

Soumis par les étudiants de 2A SN

HPC :

Wilfried L. Bounsi

Rachid Elmontassir

Charles Meyer-Hilfiger

Obeida Zakzak

Année académique 2019/2020

Table des matières

1	Introduction	2
2	Architecture	3
2.1	Partie HDFS	3
2.1.1	La classe NameNode	3
2.1.2	la classe DataNode	3
2.1.3	La classe HdfsClient	3
2.1.4	la classe Fragmenter	4
2.2	Partie Hidoop	4
2.2.1	La classe Job	4
2.2.2	La classe MapWorker	4
2.2.3	la classe Callback	4
2.3	Partie Commune	4
2.3.1	La configuration	4
2.3.2	Déploiement et Monitoring	5
2.3.3	La persistance	5
3	Mode d'emploi	6
3.1	Installation et Déploiement	6
3.2	Utilisation	6
3.2.1	Lancement & arrêt des démons	6
3.2.2	Interactions Hdfs	6
3.2.3	Exécution MapReduce	6
3.2.4	Monitoring	7
3.2.5	Evaluations des performances	7
4	Démonstration	8
4.1	Monitoring	8
4.1.1	Affichage des configurations courantes	8
4.1.2	Affichage de l'état des workers	9
4.2	Interactions Hdfs	11
4.2.1	Application de HdfsWrite	11
4.2.2	Application de HdfsRead	11
4.2.3	Application de HdfsDelete	11
4.3	Lancement d'un Job MapReduce	12
4.4	Évaluation des performances	13
5	Evaluations des performances	14
5.1	Procédure	14
5.2	Résultats	14
5.3	Interprétation	14
5.3.1	Courbe HdfsWrite	15
5.3.2	Comparatif Worcount Itératif VS MapRed	15
6	Conclusion	17

Chapitre 1

Introduction

Le principe d'une application de calcul intensif est de diviser un fichier volumineux en plusieurs fragments et de les envoyer vers différentes machines pour faire des calculs en parallèles. Dans le but de s'initier à la programmation de ce type d'applications, le projet introduit un modèle simplifié de l'API **Hadoop** (destinée à construire ces applications), appelé **Hidoop** et est composé de deux parties :

- un schéma **Map-Reduce** pour lancer le calcul sur les fragments envoyés et finir ce calcul sur les résultats récupérés
- un système de gestion de fichiers **HDFS** pour envoyer le fichier sous formes de fragments aux différents serveur de calcul et récupérer les résultats du traitement en un seul fichier

Dans ce rapport, nous présentons l'implémentation de l'application demandée par le sujet en rappelant les classes pertinentes mises en jeu. On donnera le mode d'emploi de la plateforme, ensuite nous présenterons une démonstration d'utilisation, puis nous terminerons par une évaluation des performances de l'application.

Chapitre 2

Architecture

2.1 Partie HDFS

2.1.1 La classe NameNode

Le NameNode est un objet partagé qui contient principalement les informations sur les fichiers en cours de traitement et de leurs fragments. Pour assurer le partage du NameNode on utilise un serveur RMI dont le port est spécifié dans le fichier de configurations. Cette classe à cet attribut : `HashMap<String, ArrayList<Pair<Integer, ClusterNode>>> filesIndex` qui contient les informations sur le fragments du fichier et leurs emplacements
Cette classe doit être lancée sur le NameNode.

2.1.2 la classe DataNode

Cette classe permet de relier le NameNode avec les différents Noeuds du réseaux HDFS et contient ces trois methodes :

- `recevoirFichier(String nameFile)` qui permet de recevoir un fichier `nameFile` envoyé depuis le NameNode
- `envoyerFichier(String nameFile)` qui permet d'envoyer un fichier `nameFile` au NameNode
- `deleteFichier(String nameFile)` qui permet de supprimer un fichier `nameFile` se trouvant sur le DataNode

Cette classe doit être lancée sur chaque DataNode.

2.1.3 La classe HdfsClient

Cette classe contient les trois méthodes :

HdfsWrite (Format.Type fmt, String localFSSourceFname, int repFactor) :

permet de

- fragmenter un fichier en entrée en utilisant la classe `Fragmenter` et d'envoyer ses fragments aux différents `DataNode`.
- Pour chaque fragment faire :
 - Envoyer les taille, le nom , la commande (`CMD_WRITE`) au `DataNode`.
 - Envoyer ligne par ligne le fragment.
 - Ajouter les informations de ce fragment et de ce `DataNode` dans le `NameNode`.
- Supprimer les fragments créés.

HdfsRead(String hdfsFname, String

`localFSDestFname)` Utiliser le tableau du NameNode pour savoir où se trouvent les fichiers,et pour chaque fichier de `hdfsFname` se trouvant sur HDFS faire :

- Créer le fichier `localFSDestFname` pour y mettre les fichiers reçus.
- Envoyer le nom du fichier , la commande (`CMD_READ`) au `DataNode`.
- Recevoir la taille du fichier et se mettre à l'écoute
- Recevoir le fichier et le mettre dans `localFSDestFname`
- Ajouter les informations de ce fragment et de ce `DataNode` dans le `NameNode`.

HdfsDelete(String hdfsFname)

Utiliser le tableau du NameNode pour savoir où se trouvent les fichiers de `hdfsFname`, Et Pour chaque fichier faire

- Envoyer le nom du fichier avec la commande `CMD_DELETE` au DataNode sur lequel se trouve ce fichier.

Cette classe doit être lancée sur le NameNode.

2.1.4 la classe Fragmenter

Cette classe fournit la méthode

`String[] fragmenterFichier(String Fname, int tailleFrag, String emplcmtDesFrag, Format.Type t)` permet de fragmenter un fichier et de renvoyer la liste des noms des fragments créés. Pour se faire on lit le fichier et pour chaque ligne :

- Lire la ligne
- si le type est KV, vérifier que la `clé` et sa `valeur` sont bien dans la même ligne si ce n'est pas le cas mettre la `clé` dans la ligne suivante comme ça en cas d'arrêt dans cette ligne on aura pas de perte d'information
- Si la taille du fragment qu'on est en train de remplir est supérieur à `tailleFrag` on ferme le fragment on l'ajoute à la liste des fragments créés et on crée un autre...
- à la fin du fichier on renvoie la liste des fragments créés.

2.2 Partie Hadoop

2.2.1 La classe Job

La classe Job se charge de lancer les maps sur les différents nœuds sur lesquels se trouvent les fragments. Elle s'occupe également de lire le résultat des maps en un fichier local grâce à un appel à `HdfsWrite` puis de lancer le reduce sur ce fichier une fois que les maps sont terminés. L'attente de la terminaison des maps a été implantée à l'aide d'un sémaphore initialisé à 0 (bloquant).

2.2.2 La classe MapWorker

Cette classe possède une méthode principale `runMap` qui se charge d'exécuter dans un thread, un map sur un fragment de fichier stocké dans hdfs. Au terme de l'exécution du map, elle fait usage du callback passé en argument pour notifier le Job de la terminaison du map.

2.2.3 la classe Callback

Le Callback est un objet dont les MapWorkers se servent pour signaler la terminaison d'un map au Job appelant.

2.3 Partie Commune

2.3.1 La configuration

Les fichiers de configurations

Il s'agit des deux fichiers

- `core-site.xml`

Ce fichier contient les configurations globales du système à l'instar des informations sur les différents nœuds du cluster et le port rmi du rmiregistry utilisé par les différents objets partagés dans le cluster.

- `hdfs-site.xml`

Ce fichier contient les configurations propres à hdfs, notamment :

- le facteur de réplication

- le répertoire de stockage permanent

Utilisé d'une part par les datanodes pour y stocker les fragments et d'autre part, par le namenode pour y stocker l'index des fichiers (objet serialisé)

- Le port sur lequel répondent les démons datanodes.

La classe Config

Principale classe du package config, la classe Config a pour mission de charger les configurations de la plateforme, d'une part, à partir depuis les fichiers de configuration xml ci-haut mentionnés et d'autre part, depuis les variables d'environnement système.

2.3.2 Déploiement et Monitoring

Le package admin

Le package admin contient des classes qui implémentent des fonctions permettant le lancement, l'arrêt et le monitoring de la plateforme..

2.3.3 La persistance

Grâce à la sauvegarde de son index (sous forme d'un objet serialisé) dans un fichier du système, le NameNode assure la persistance des données stockés dans Hdfs en cas de redémarrage ou d'interruption brusque de son démon.

Chapitre 3

Mode d'emploi

3.1 Installation et Déploiement

Sur chaque machine du cluster :

1. Copiez le dossier du projet dans l'emplacement de votre choix.
2. Editez à votre convenance les fichiers de configurations qui se trouvent dans le dossier `config`.
3. Ajoutez les lignes suivantes dans votre profil bash puis recharger le.

```
1 # Adding Hadoop
2 HADOOP_HOME=/path/to/hadoop
3 export HADOOP_HOME
4 PATH=$PATH:$HADOOP_HOME/bin
5 export PATH
```

Listing 3.1 – Variables d'environnement

4. Ouvrez un terminal.
5. Tapez la commande `hadoop start`.

3.2 Utilisation

Les interactions avec la plateforme se font par l'intermédiaire de la commande `hadoop` qui peut être lancée depuis n'importe quel répertoire quand l'étape 3 de l'installation s'est bien déroulé.

3.2.1 Lancement & arrêt des démons

- `hadoop start`
Lance les bons démons sur le noeud.
- `hadoop stop`
Stop les démons sur le noeud.
- `hadoop restart`
Arrête et relance les démons sur le noeud.

3.2.2 Interactions Hdfs

- `hadoop write line|kv <input>`
- `hadoop read <hdfs_file> <output_file>`
- `hadoop delete <hdfs_file>`
- `hadoop ls [<hdfs_file>]`

3.2.3 Exécution MapReduce

- `hadoop run <app_name> <hdfs_file>`

3.2.4 Monitoring

- `hidoop report configs`
Affiche juste les configurations en cours
- `hidoop report master`
Affiche juste les infos sur le master
- `hidoop report workers`
Affiche juste les infos sur les workers
- `hidoop report worker <hostname>`
Affiche les infos sur un worker
- `hidoop report`
Affiche un rapport global

3.2.5 Evaluations des performances

- `hidoop bench mb|gb <data_size>`
Evalue les performances du cluster sur un fichier de taille `datasize` méga octets (opt. mb) ou giga octets (opt. gb). Ce fichier est généré automatiquement à l'aide d'un script `data-gen` disponible dans le dossier bin du projet.

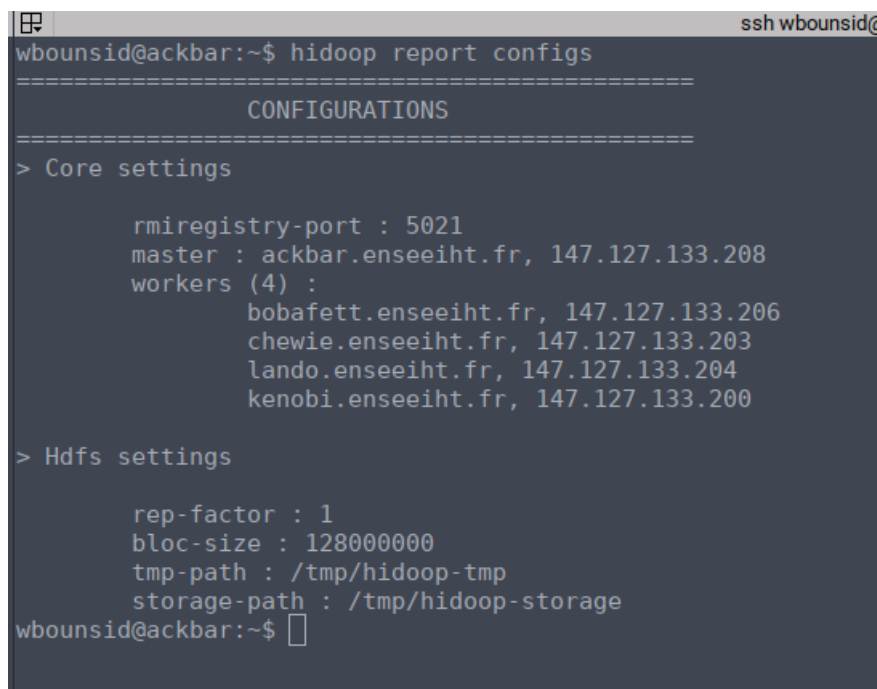
Chapitre 4

Démonstration

4.1 Monitoring

Nous ne présentons ici que quelques captures d'écran illustrant le fonctionnement des commandes shell disponibles.

4.1.1 Affichage des configurations courantes



```
ssh wbounsid@ackbar:~$ hadoop report configs
=====
CONFIGURATIONS
=====
> Core settings

rmiregistry-port : 5021
master : ackbar.enseeiht.fr, 147.127.133.208
workers (4) :
    bobafett.enseeiht.fr, 147.127.133.206
    chewie.enseeiht.fr, 147.127.133.203
    lando.enseeiht.fr, 147.127.133.204
    kenobi.enseeiht.fr, 147.127.133.200

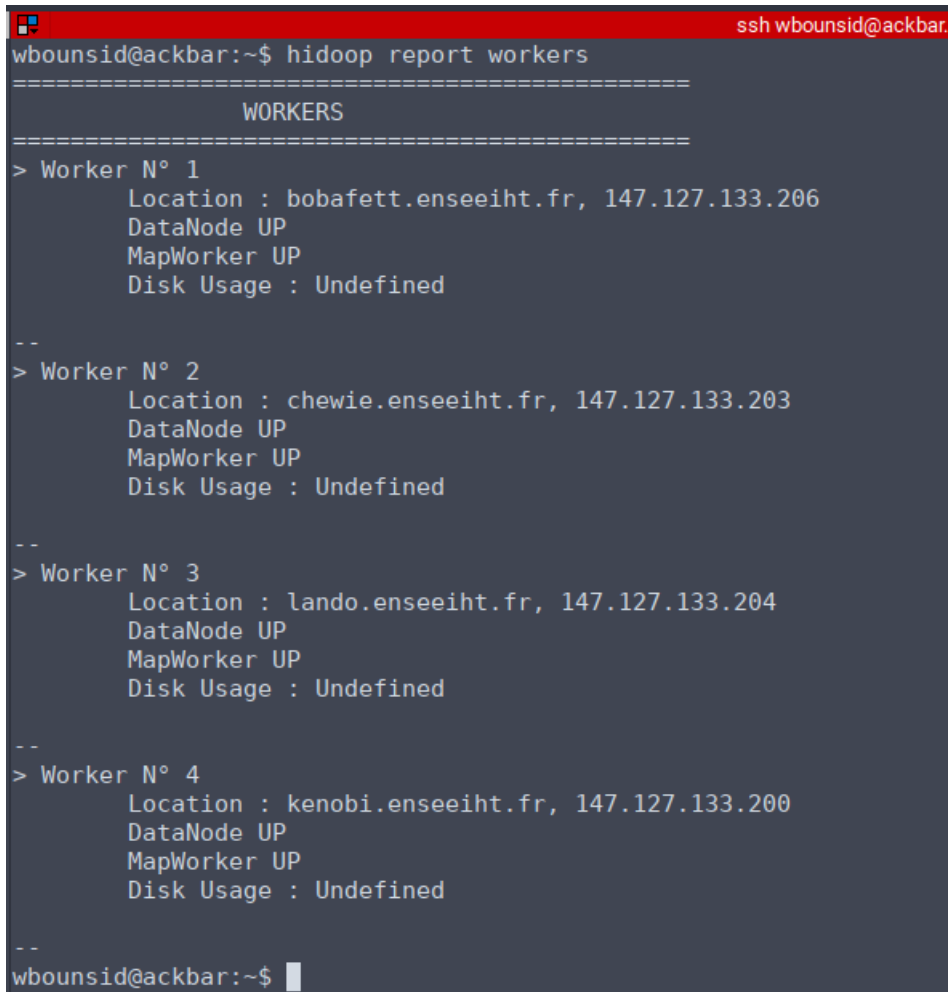
> Hdfs settings

rep-factor : 1
bloc-size : 128000000
tmp-path : /tmp/hadoop-tmp
storage-path : /tmp/hadoop-storage
wbounsid@ackbar:~$
```

FIGURE 4.1 – Affichage des configurations

4.1.2 Affichage de l'état des workers

Tous les workers sont actifs



```
wbounsid@ackbar:~$ hadoop report workers
=====
                        WORKERS
=====
> Worker N° 1
  Location : bobafett.enseeiht.fr, 147.127.133.206
  DataNode UP
  MapWorker UP
  Disk Usage : Undefined

--
> Worker N° 2
  Location : chewie.enseeiht.fr, 147.127.133.203
  DataNode UP
  MapWorker UP
  Disk Usage : Undefined

--
> Worker N° 3
  Location : lando.enseeiht.fr, 147.127.133.204
  DataNode UP
  MapWorker UP
  Disk Usage : Undefined

--
> Worker N° 4
  Location : kenobi.enseeiht.fr, 147.127.133.200
  DataNode UP
  MapWorker UP
  Disk Usage : Undefined

--
wbounsid@ackbar:~$
```

FIGURE 4.2 – Workers tous actifs

Un Worker est inactif

```
ssh wbounsid@bobafett.enseeiht.fr
wbounsid@bobafett:~$ hidoop stop
Stopping DataNode and MapWorker...
wbounsid@bobafett:~$
```

FIGURE 4.3 – Arret d'un worker

```
ssh wbounsid@ackbar.enseeiht.fr
wbounsid@ackbar:~$ hidoop report workers
=====
                        WORKERS
=====
> Worker N° 1
    Location : bobafett.enseeiht.fr, 147.127.133.206
    DataNode DOWN
    MapWorker DOWN
    Disk Usage : Undefined
--
> Worker N° 2
    Location : chewie.enseeiht.fr, 147.127.133.203
    DataNode UP
    MapWorker UP
    Disk Usage : Undefined
--
> Worker N° 3
    Location : lando.enseeiht.fr, 147.127.133.204
    DataNode UP
    MapWorker UP
    Disk Usage : Undefined
--
> Worker N° 4
    Location : kenobi.enseeiht.fr, 147.127.133.200
    DataNode UP
    MapWorker UP
    Disk Usage : Undefined
--
```

FIGURE 4.4 – Workers tous actifs sauf un

4.2 Interactions Hdfs

4.2.1 Application de HdfsWrite

```
wbounsid@ackbar:/tmp$ hidoop write line 1mb.txt
1mb.txt
Fragmentation du fichier: 1mb.txt ... OK
  1 fragments créés
Suppression du fragment /tmp/hidoop-tmp/1mb.txt.frag.0 ...OK
wbounsid@ackbar:/tmp$ hidoop ls
1mb.txt => [
            (0, bobafett.enseeiht.fr);
          ]
```

4.2.2 Application de HdfsRead

```
wbounsid@ackbar:/tmp$ hidoop ls
1mb.txt => [
            (0, bobafett.enseeiht.fr);
          ]

wbounsid@ackbar:/tmp$ hidoop read 1mb.txt out.txt
--Reception du fichier 0 ...OK
--Concatenation des fichiers reçu ...
--Fichier out.txt crée
wbounsid@ackbar:/tmp$ ls -lh out.txt
-rw-rw-r-- 1 wbounsid wbounsid 1023K janv. 15 19:32 out.txt
wbounsid@ackbar:/tmp$ diff 1mb.txt out.txt
wbounsid@ackbar:/tmp$
```

4.2.3 Application de HdfsDelete

```
wbounsid@ackbar:/tmp$ hidoop ls
1mb.txt => [
            (0, bobafett.enseeiht.fr);
          ]

wbounsid@ackbar:/tmp$ hidoop delete 1mb.txt
wbounsid@ackbar:/tmp$ hidoop ls

wbounsid@ackbar:/tmp$
```

4.3 Lancement d'un Job MapReduce

```
ssh wboundsid@ackbar.enseeiht.fr
wboundsid@ackbar:~/nosave/hadoop/bin$ hadoop ls
512mb.txt => [
  (0, bobafett.enseeiht.fr);
  (1, chewie.enseeiht.fr);
  (2, lando.enseeiht.fr);
  (3, kenobi.enseeiht.fr);
  (4, bobafett.enseeiht.fr);
]

wboundsid@ackbar:~/nosave/hadoop/bin$ hadoop run application.MyMapReduce 512m
Job : Lancement des maps...
Job : Lancement d'un map sur le noeud bobafett.enseeiht.fr
Job : Map lancé sur le noeud bobafett.enseeiht.fr
Job : Lancement d'un map sur le noeud chewie.enseeiht.fr
Job : Map lancé sur le noeud chewie.enseeiht.fr
Job : Lancement d'un map sur le noeud lando.enseeiht.fr
Job : Map lancé sur le noeud lando.enseeiht.fr
Job : Lancement d'un map sur le noeud kenobi.enseeiht.fr
Job : Map lancé sur le noeud kenobi.enseeiht.fr
Job : Lancement d'un map sur le noeud bobafett.enseeiht.fr
Job : Map lancé sur le noeud bobafett.enseeiht.fr
Job : Tous les maps sont lancés
Job : un map terminé 4 restant(s)...
Job : un map terminé 3 restant(s)...
Job : un map terminé 2 restant(s)...
Job : un map terminé 1 restant(s)...
Job : un map terminé 0 restant(s)...
Job : Tous les maps sont terminés
Job : Fusion des resultats des maps...
--Reception du fichier 0 ...0K
--Reception du fichier 1 ...0K
--Reception du fichier 2 ...0K
--Reception du fichier 3 ...0K
--Reception du fichier 4 ...0K
--Concatenation des fichiers reçu ...
--Fichier 512mb.txt-map créé
Job : Succes
Job : Lancement du reduce...
Job : Succes
Job : Terminé, fichier output -> 512mb.txt-reduce
time in ms =4225
wboundsid@ackbar:~/nosave/hadoop/bin$ head -n 10 512mb.txt-reduce
Conflict.--As<->1024
Reserve<->512
happiness."<->1024
confirmed--Napoleon<->1024
costing<->512
```

FIGURE 4.5 – Exécution job mapreduce

4.4 Évaluation des performances

```
wbounsid@ackbar:~/nosave/hadoop/bin$  
wbounsid@ackbar:~/nosave/hadoop/bin$ hadoop bench mb 512  
Taille fichier de test généré => 511,482MiB  
Temps d'exécution hdfs write : 14427ms  
Temps d'exécution wordcount itératif : 9128ms  
Temps d'exécution wordcount mapred : 4034ms  
Différences entre les sorties : 0  
wbounsid@ackbar:~/nosave/hadoop/bin$
```

FIGURE 4.6 – Démo evaluation des performances

Chapitre 5

Evaluations des performances

Pour les configurations suivantes :

- Taille bloc : 128Mo
- Facteur de réplication : 1
- 4 Workers¹

NB La dernière ligne affichée ici signifie qu'il n'existe pas de différences entre les lignes des deux fichiers résultats, l'un issue de l'exécution du wordcount itératif et l'autre de l'exécution du wordcount MapReduce

5.1 Procédure

Nous écrivons le script suivant

```
1 touch evals
2 hidoop bench mb 64 &>> evals
3 hidoop bench mb 128 &>> evals
4 hidoop bench mb 256 &>> evals
5 hidoop bench mb 512 &>> evals
6 hidoop bench gb 1 &>> evals
7 hidoop bench gb 2 &>> evals
8 hidoop bench gb 5 &>> evals
9 hidoop bench gb 10 &>> evals
```

Listing 5.1 – Script d'évaluation

5.2 Résultats

L'exécution du script précédent nous génère un fichier `eval_perfs`² contenant les résultats que nous synthétisons dans le tableau suivant.

Taille Fichier	Temps HdfsWrite (ms)	Temps WordCount Itératif (ms)	Temps WordCount MapRed (ms)
64M	2 309	1 395	2 073
128M	4 564	2 909	3 254
256M	8 432	5 323	3 511
512M	16 679	9 920	3 758
1G	31 954	18 130	4 753
2G	63 674	37 661	6 050
5G	162 346	93 194	13 947
10G	281 866	185 142	36 871

5.3 Interprétation

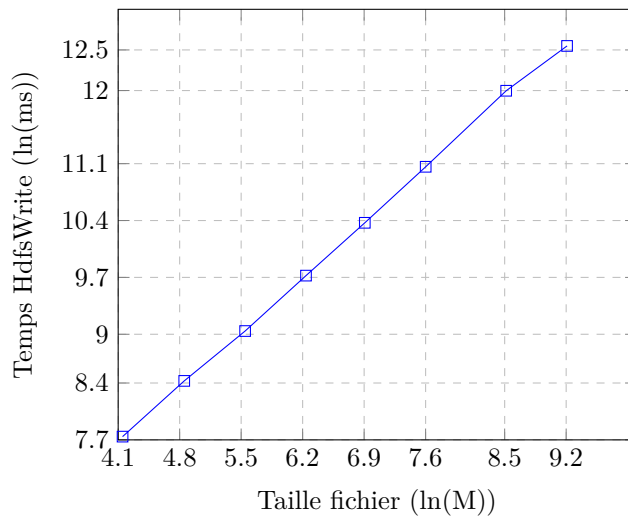
On peut étudier ces résultats sous forme graphique avec une double échelle logarithmique pour faciliter leurs représentation.

1. Caractéristiques : 16Gb RAM, Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz ×4, Il en va de même pour le noeud Master

2. Ce fichier se trouve dans le dossier doc du projet

5.3.1 Courbe HdfsWrite

Comparaison du temps HdfsWrite en fonction de la taille des fichiers



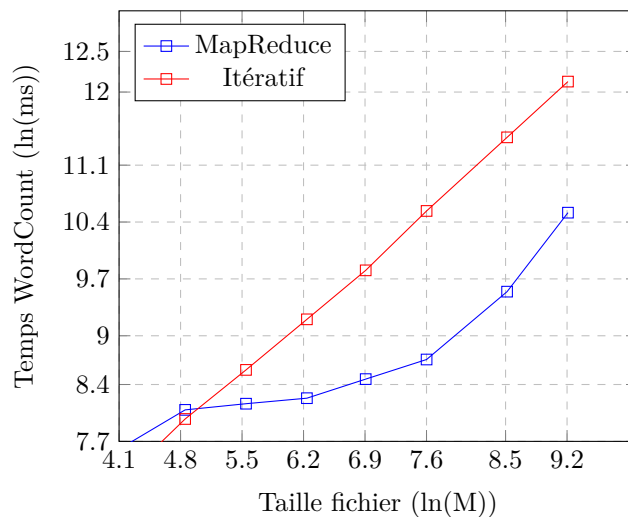
Comme on le voit, le temps du HdfsWrite évolue linéairement par rapport à la taille du fichier. On se retrouve donc, dans les cas où les fichiers sont volumineux (supérieur à 100 Méga et beaucoup plus), avec HdfsWrite qui prends le plus de temps. Le WordCount ne représentant alors qu'une petite partie du temps de calcul total. Il est en revanche bien attendu que le temps du HdfsWrite évolue linéairement par rapport à la taille du fichier car les envoient des fragments sont successif (même avec des threads).

5.3.2 Comparatif Wordcount Itératif VS MapRed

On compare maintenant le temps du WordCount en itératif et avec MapReduce. Le temps WordCount MapReduce prend en compte seulement le temps d'exécution des procédures Maps (le plus long), de la récupération des données avec HdfsRead et de la procédure Reduce.

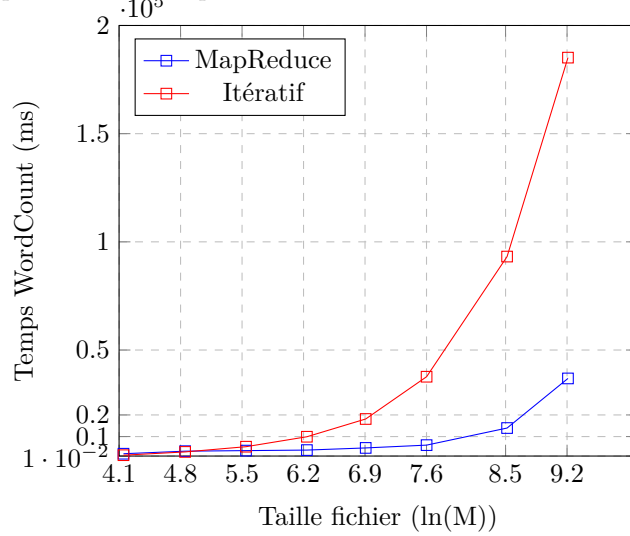
Double échelle logarithmique

Comparaison des temps de WordCount en fonction de la taille des fichiers



Echelle logarithmique sur la taille des fichiers

Comparaison des temps de WordCount en fonction de la taille des fichiers



On remarque que le WordCount en MapReduce est bien plus efficace que celui en Itératif. Cela est du en parti au fait que le WordCount est une procédure qui est facilement partageable et que la charge du Reducer n'est pas trop importante. Par une procédure d'optimisation classique (Gradient Conjugué Tronqué) de fonction est bien plus difficilement au MapReduce de part sont coté "itératif obligatoire". Il faut donc être critique et ne pas tout transformer en procédure MapReduce..

Chapitre 6

Conclusion

Nous pensons que le projet est une réussite, tant au niveau du modèle qu'à sa réalisation. Nous avons également largement tiré avantage de la mise en place des retours croisés, notamment pour les tests. Nous devons néanmoins noter que si la procédure de WordCount en MapReduce s'effectue bien plus rapidement que celle en itératif, le scénario globale est ralenti par HdfsWrite qui occupe la majorité du temps dans le cas de fichier volumineux. Nous pensons que notre modèle est bon mais que ceci est dû à des détails d'implantations. Toujours en rapport avec cette problématique, nous souhaitons étudier de manière plus précise l'importance du rapport de la taille du fichier sur la taille de ces fragments quant à la vitesse d'exécution du HdfsWrite et du MapReduce. Il pourrait donc être intéressant de mettre en place une taille de fragment variable.

Une amélioration possible (et prochaine) de notre plateforme serait la mise en place de connections ssh sans mot de passe, qui permettrait le déploiement des démons sur tous les noeuds, depuis n'importe quel machine du cluster.