École Nationale Supérieure
d'Électrotechnique, d'Électronique, d'Informatique,
d'Hydraulique et des Télécommunications

Institut National Polytechnique

Rapport Intermédiaire
Hidoop V.1

Aymen Chla
Mohamed Maghfour
Iliass Moumen
Salah Eddine Chailou

SOMMAIRE

Table des figures

Introduction

Dans cette partie du projet, il nous est demandé d'améliorer le fonctionnement du map/reduce sur des fichiers stockés en réparti sur des serveurs distants HDFS, gérer les pannes, augmenter le nombre de reducers ainsi que le test de notre programme sur plusieurs applications.

Ce rapport présentera donc l'avancement de notre groupe en expliquant les tâches réalisées, et celles en cours de développement.

1 Rectifications par rapport à la version 0

On a commencé par rectifier notre programme de la version 0 afin d'automatiser le lancement de notre projet hidoop.

1.1 Utilisation d'un script

pour l'automatisation de notre programme, on a implanté deux scripts :

- **start.sh** : ce script sert à établir une connexion avec les serveurs distants et d'exécuter notre programme en utilisant la commande **ssh**.
- **stop.sh** : ce script permet de libérer les connexions TCP établies lors de l'utilisation du script précèdent en utilisant la commande **fuser -k port/tcp**.

Dans les deux scripts, les informations sur le serveur, à savoir son adresse ip et son port, sont extraites à partir des fichiers

- **namenode.properties** : pour le serveur qui va jouer le rôle du NameNode.
- **datanodes.properties** : pour les serveurs DataNodes.
- **daemons.properties** : pour les serveurs qui appliquent le traitement.

Ces fichiers *properties* sont organisés de la manière suivantes :
chaque ligne contient une information sur un serveur par exemple :

host1 = dragon.enseeiht.fr
port1 = 4000
name1 = m1 pour les daemons

1.2 Partition Local des fichiers

Dans la version 0, on a stocké les fichiers intermédiaires et les répartitions des fichiers dans le même endroit qui est /data . l'inconvénient c'est que ce répertoire est partagé entre les différentes machines. pour cela on a modifié les chemins de stockage des fichiers en utilisant la partition locale sans NFS (Network File System) qui est */tmp*.

2 Étude des Applications

2.1 Étude de scalabilité sur WordCount

Afin de bien tester notre code et voir des résultats concrets, On a essayé plusieurs configurations et des tailles des fichiers qui varient entre 1Ko et 1Go. L'intérêt est de comparer les temps d'exécutions et mettre les points sur les qualités et les défauts de notre code.

Pour bien visualiser les résultats obtenus, le diagramme ci-dessous présente les temps d'exécution des fichiers de différentes tailles en utilisant 3 DataNodes dans le 1er test et 5 dans le 2ème.

File Size (M)	Parallel Time (ms)	Sequential Time (ms)
1.3	234	4
511	7741	14961
767	11442	18689
1022	15213	31574

Table 1 – scalabilité sur WordCount avec 3 DataNodes

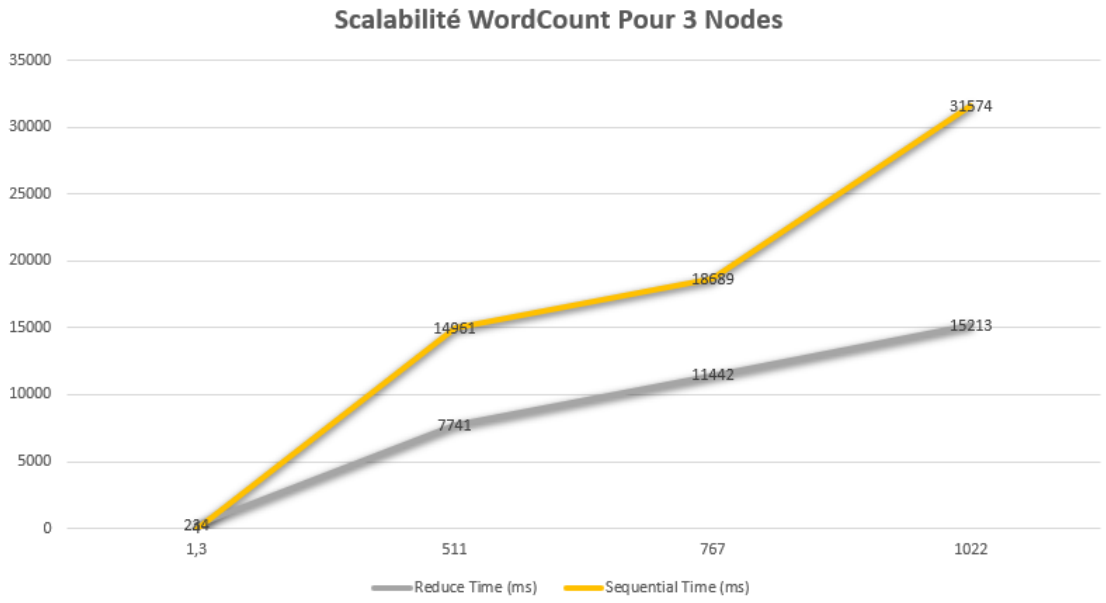

Figure 1 – Diagramme de test sur 3 Nodes

File Size (M)	Parallel Time (ms)	Sequential Time (ms)
1.3	105	4
511	10014	14961
767	26970	18689
1022	37875	31574

Table 2 – scalabilité sur WordCount avec 5 DataNodes

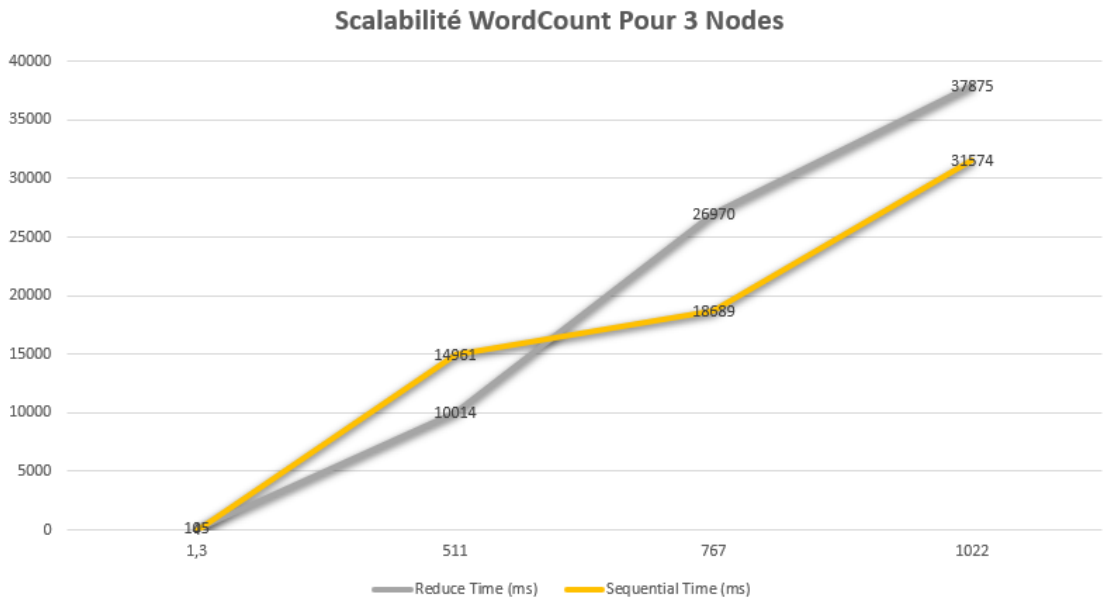

Figure 2 – Diagramme de test sur 5 Nodes

2.2 Choix des Applications

Pour le choix d'applications, on s'est mis d'accord d'ajouter :

— L'application **Quasi-Monte-Carlo** qui permet de résoudre des problèmes numériques, nous trouvons donc par exemple une approximation du nombre π par la méthode de Monte-Carlo géométrique.
— Les différentes variantes/extensions autour du comptage des mots d'un fichier de texte à savoir **WordMedian WordMean ...**

On a commencé l'implantation de l'application Quasi-Monte-Carlo en se basant sur le fichier fourni sur moodle, mais il reste encore des bugs non encore résolus.

3 Augmentation de la fiabilité

3.1 Ajout d'un nombre de réctifications

Dans cette partie, nous souhaitons assurer plus de fiabilité à notre programme dans le cas où un de nos serveurs tombe en panne. Dans ce cas là, on a décidé d'augmenter la fiabilité au niveau HDFS en gérant la réplication des blocs.
En fixant un nombre de réplication $repfact$ (on a pris par exemple 3), chaque chunk sera stocké dans $repfact$ différents serveurs pour assurer la récupération du fichier en cas de panne.
la manière avec laquelle on stocke les chunks sur les serveurs assure l'équilibrage de charge(au niveau de la taille de stockage sur chaque serveur) vu qu'on parcourt les serveurs dans un ordre bien connu.

3.2 Fiabilité lors de la Récupération

Pour bénéficier de la réplication, on a modifié notre programme pour qu'il puisse récupérer un chunck à partir d'un serveur même si un (le serveur qui devrait être normalement fonctionnel) de nos serveurs tombe en panne.
Le NameNode stocke les serveurs qui possèdent chaque chunk dans un fichier, et

pour récupérer le chunk, on essaie d'établir une connexion avec un de ces serveurs, si la connexion est bien établie on commence à transmettre le fichier, si une erreur est survenue (utilisation try catch finally) on essaie d'établir la connexion avec le prochain serveur qui possède le chunk.

3.3 Gestion des ressources

Dans une première étape nous souhaitons faire du monitoring des ressources locales (mémoire, processeur....) de toutes les machines de hidoop. Pour ce faire nous allons commencer par exécuter le NodeManager sur toutes les machines, ces dernières seront enregistré auprès du NameNode et vont envoyer les informations sur les ressources au RessourceManager.

4 Conclusion

Pour nous, la gestion des Daemons était déjà traitée dans la version 0, mais on est en train d'ajouter un daemon $RessourceManager$ pour la gestion des pannes des DataNodes. Aussi, L'idée pour les semaines qui viennent est d'implanter le $Shuffle$ qui consiste à regrouper et trier par clés les paires (Clé,Valeur) du résultat de la fonction Map pour faire des $reducesmultiples$.