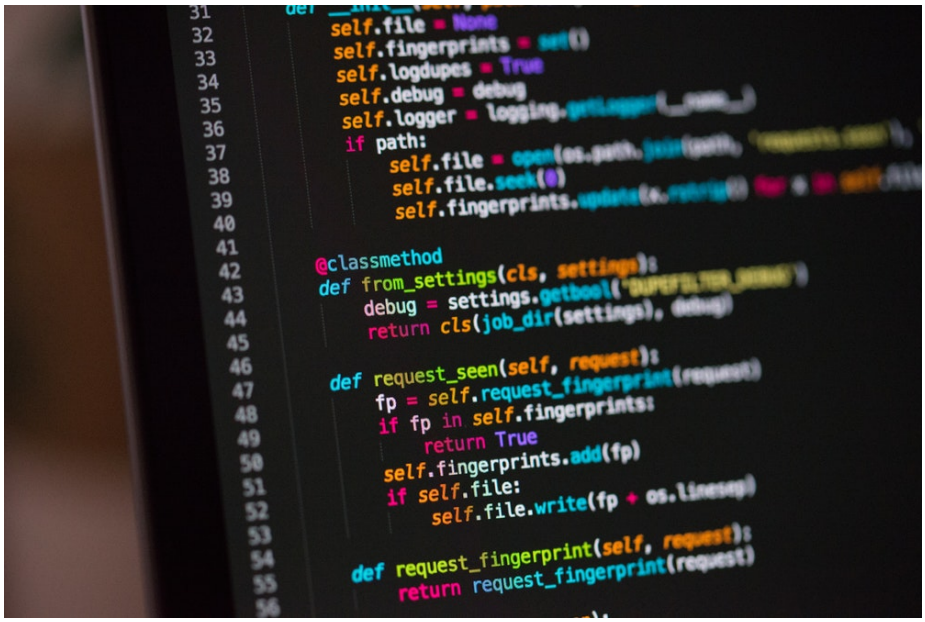
Pont d'avancement 1



Hamza Mouddene
Manal Hajji

21 février 2021

Table des matières

1 Introduction

L'introduction de ce rapport représentera une synthèse de ce qui a été fait par le passé, je tiens à préciser que notre version répartie est opérationnelle à 100% sans le moindre problème, en plus de ça nous avons réalisé des tests unitaires sur toute ce qui a été fait afin de couvrir un maximum l'intégralité du code ainsi que des tests de performance sur des jeux de données tailles différentes qui vont de 10Ko jusqu'à 6.8Go.

2 Tests

Plusieurs tests ont été effectués pour vérifier la correction de passage à l'échelle, nous avons choisit d'enrichir notre programme avec un jeu de tests unitaires afin de s'assurer de la pertinence du produit final, puis viennent les tests de performances pour s'assurer que le programme est fonctionnel pour un jeu de données d'une taille quelconque afin de s'assurer que ceci fonctionne tout le temps quelques soit les circonstances.

3 Déploiement et le test de la version répartie

Dans la racine du projet, il y a un script nommé *run.sh* qui permet de déployer la plateforme avec une simple ligne de commande.

```
./run.sh
```

4 Étude de *scalabilité* sur *WordCount*

Nous avons choisi de tester *Hidoop* et *HDFS* avec un nombre variable de serveurs et *Workers* sur un fichier de 8.2 Go. Le protocole expérimental consiste à :

- Écrire le fichier à tester sur *HDFS* avec le bon nombre de chunks.HDFS
- Effectuer le mapReduce sur les chunks écrits.
- Nettoyer les ordinateurs et réitérer.

Voici le résultat final : On peut observer que le nombre de serveurs permet bien de diminuer le temps d'exécution de Hidoop. Cependant, plus le nombre de serveurs augmente, moins le temps gagné est conséquent : il y a un plafond au temps que nous permet de gagner ce système, ce qui est normal.

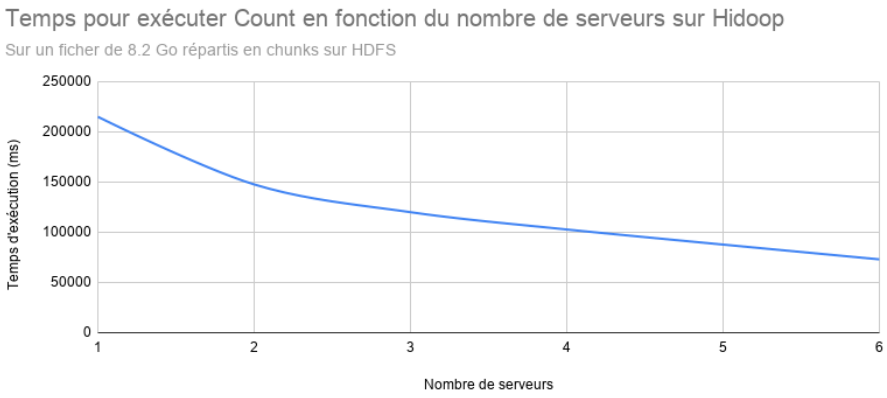


FIGURE 1 – Résultat du test de performance

5 les améliorations envisagées à la plateforme produite, afin d'en améliorer les performances

On compte apporter une améliorations du coté Hidoop, qui consiste en réalité de paralléliser les runMap de chaque Worker. En théorie, si on a un Worker qui exécute en séquentiel un runMap dans un laps de temps t, ceci peut être diviser en 2 avec un Worker qui exécute runMap avec deux processus ou bien $\frac{t}{4}$ avec 4 processus.

6 L'application choisie

Nous avons choisit l'option **Tolérance aux pannes** qui consiste :

— Coté **HDFS** : Gérer les pannes des serveurs HDFS en testant continuement si le serveur est encore vivant. Si un serveur tombe en panne pendant une lecture, il faut pouvoir continuer la lecture au même point avec un autre serveur disponible. Si un serveur tombe en panne pendant une écriture, il faut pouvoir réécriture le chunk en en cours sur un autre serveur disponible. Si un serveur tombe en panne durant une suppression de fichier, alors, dans le cas où ce serveur revient en ligne, qu'il supprime le fichier.

— Coté **Hidoop** : Gérer la disponibilité des *Workers*, relancer les *Maps* impactés, prendre l'exécution sans ré-exécution de tous les *Maps*.