# SLR207 - Théo KARST

Théo KARST

June 2023

## 1 Introduction

L'objectif de ce projet était d'implémenter l'algorithme de MapReduce afin d'utiliser du calcul en parallèle sur plusieurs ordinateurs pour compter le nombre d'occurences de chaque mot dans un fichier volumineux.

Dans la partie préliminaire, nous nous intéresserons à l'implémentation d'un algorithme séquentiel, sur un seul ordinateur, en vue de réaliser le dénombrement des occurences des mots d'un fichier. Nous détaillerons ensuite l'utilisation de diverses commandes Linux utiles dans le cadre de ce projet, pour échanger des fichiers et exécuter des commandes sur des ordinateurs distants.

Dans une seconde partie, nous verrons le fonctionnement de l'algorithme MapReduce, ainsi que la façon dont il a été implémenté dans ce projet.

Enfin, dans une dernière partie, je présenterai mes résultats sur cette implémentation de l'algorithme MapReduce, en essayant notemment de déterminer l'influence de du nombre de machines sur la vitesse d'exécution de l'algorithme.

## 2 Préliminaires

### 2.1 Etape 1

#### 2.1.1 Premier comptage en séquentiel pur

La structure de données HashMap de Java est pertinente pour stocker les données, car elle permet d'associer à chaque mot (utilisé comme une clé) son nombre d'occurences dans le fichier. De plus, HashMap permet d'ajouter et de trouver des élements avec une complexité en $O(1)$ en théorie, ce qui est efficace.

Pour trier les mots par nombre d'occurences, puis par ordre alphabetique, on peut utiliser l'interface Comparable de Java et la méthode `sort()` de `java.util.Collections`.

#### 2.1.2 Test du programme séquentiel sur le code forestier de Mayotte

En testant le programme sur le fichier `forestier_mayotte.txt`, on obtient le résultat suivant (seulement les 5 premiers mots sont affichés):

```
de:  12
biens:  8
ou:  8
forestier:  6
des:  5
```

#### 2.1.3 Test du programme séquentiel sur le code de la déontologie de la police nationale

En testant le programme sur le fichier `deontologie_police_nationale.txt`, on obtient le résultat suivant (seulement les 5 premiers mots sont affichés):

```
de:  86
```

```
la:  40
police:  29
et:  27
à:  25
```

#### 2.1.4 Test du programme séquentiel sur le code du domaine public fluvial

En testant le programme sur le fichier `domaine_public_fluvial.txt`, on obtient le résultat suivant (seulement les 5 premiers mots sont affichés):

```
de:  621
le:  373
du:  347
la:  330
et:  266
```

#### 2.1.5 Test du programme séquentiel sur le code de la santé publique

En testant le programme sur le fichier `sante_publique.txt`, on obtient le résultat suivant (seulement les 5 premiers mots sont affichés):

```
de:  189699
la:  74433
des:  66705
à:  65462
et:  60940
```

Sur le code de la santé publique, on obtient les chronométrages suivants:

- Compter le nombre d'occurences: 454 ms
- Tri (par nombre d'occurences et alphabétique): 63 ms

On peut comparer ces chronométrages avec une version du programme écrite en C++, utilisant une map<string, int> et compilée avec le flag d'optimisation -O3 (optimisation maximale). On obtient une durée totale de 576 ms (les map de C++ trient automatiquement les données en fonction de la clé).

#### 2.1.6 Travailler sur des plus gros fichiers

En utilisant le fichier `CC-MAIN-20220116093137-20220116123137-00001.warc.wet`, on obtient le résultat suivant (seulement les 5 premiers mots sont affichés):

```
the:  415696
to:  329324
and:  324287
de:  308976
-:  295589
```

On obtient un chronométrage de 8631 ms pour le décompte des occurences de chaque mot, et une durée de 4949 ms pour le tri par nombre d'occurences et par ordre alphabétique.

### 2.2 Etape 2

#### 2.2.1 Nom court, nom long

Le nom court de mon ordinateur de TP est `tp-1d22-10`, et s'obtient avec la commande `hostname -s`. Son nom long est `tp-1d22-10.enst.fr` et s'obtient avec la commande `hostname -f`. Sous Linux, il est également possible de consulter son `hostname` en regardant le fichier `/etc/hostname`.

### 2.2.2 Adresse IP

La commande `ip addr` permet de connaitre son (ses) adresse(s) IP(s). Mais de nombreux sites webs proposent d'afficher son adresse IP, comme par exemple le site: https://nordvpn.com/fr/what-is-my-ip/.

### 2.2.3 Du nom vers l'IP

Il est possible de retrouver l'adresse IP d'un ordinateur à partir de son nom complet avec l'outil `nslookup`:

```
$ nslookup tp-1d22-10.enst.fr
Server:  127.0.0.53
Address:  127.0.0.53#53

Non-authoritative answer:
Name:  tp-1d22-10.enst.fr
Address:  137.194.141.139
```

### 2.2.4 De l'IP vers le nom

De même, `nslookup` permet de retrouver le nom long d'un ordinateur à partir de son adresse IP:

```
$ nslookup 137.194.141.139
139.141.194.137.in-addr.arpa name = tp-1d22-10.enst.fr.
```

### 2.2.5 Ping pong à l'intérieur!

On remarque que la commande `ping` fonctionne avec l'adresse IP de l'ordinateur distant, et avec son nom long, mais pas avec son nom court:

```
$ ping tp-1d22-10.enst.fr
PING tp-1d22-10.enst.fr (137.194.141.139) 56(84) bytes of data.
64 bytes from tp-1d22-10.enst.fr (137.194.141.139):  icmp_seq=1 ttl=58 time=21.7
ms
^C
--- tp-1d22-10.enst.fr ping statistics ---
1 packets transmitted, 1 received, 0rtt min/avg/max/mdev = 21.745/21.745/21.745/0.000
ms

$ ping tp-1d22-10
ping:  tp-1d22-10:  Échec temporaire dans la résolution du nom
```

### 2.2.6 Calculer en ligne de commande

Il est possible de calculer en ligne de commande avec Python par exemple:

```
$ python3 -c "print('2+3 =', 2+3)"
2+3 = 5
```

Pour effectuer le même calcul sur un ordinateur distant sans mot de passe, on peut utiliser `ssh`. On commence par ajouter sa clé publique dans les `authorized_keys` de l'ordinateur distant, et on peut ensuite executer sur son ordinateur local:

```
$ ssh karst-21@tp-1d22-10.enst.fr "python3 -c \"print('2+3 =', 2+3)\""
2+3 = 5
```

## 2.3 Etape 3

Sur les ordinateurs de l'école, les fichiers ne sont pas enregistrés localement, mais à distance avec le système NFS, ce qui permet de retrouver les fichiers sur sa session, peu importe sur quel ordinateur de l'école on se connecte. La commande `stat` permet de savoir où est enregistré physiquement chaque fichier. On peut ainsi remarquer que les fichiers dans le dossier `/tmp` sont stockés physiquement sur l'ordinateur, mais pas les fichiers dans notre dossier personnel:

```
$ stat -f ~/ftemp.txt
Fichier :  " /cal/exterieurs/karst-21/ftemp.txt "
Identif.  :  0 Longueur du nom :  255 Type :  nfs
Taille de bloc :  1048576 Taille de bloc fondamentale :  1048576
Blocs :  total :  2097152 libre :  594921 disponible :  594921
Inœuds :  total :  1252211833 libre :  1218396585
```

La commande `scp` permet de copier un fichier local sur un ordinateur distant ou inversement:

```
$ scp local_file karst-21@tp-1d22-10.enst.fr:/path/to/remote/folder
```

# 3 Implémentation

## 3.1 Présentation de l'algorithme MapReduce

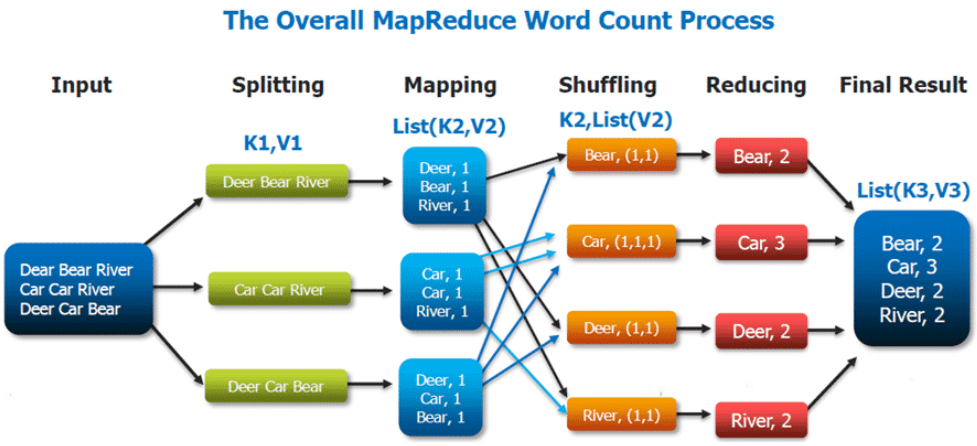


Figure 1: Schema des différentes parties de l'algorithme de MapReduce

MapReduce est un modèle de programmation permettant de traiter un grand volume de données grace à du calcul en parallèle sur de nombreux ordinateurs. Il a l'inconvénient d'être plus lent que d'autres modèles existants, mais possède l'avantage d'être résistant aux pannes, comme les données sont distribuées sur un grand nombre d'ordinateurs. Il est composé de 5 étapes principales:

- Splitting: Le fichier de départ est divisé en différentes parties, qui sont chacunes envoyées sur un ordinateur différent
- Mapping: Chaque ordinateur distant traite le fichier qui lui a été envoyé durant la phase de Splitting, et forme un ensemble de paires clé-valeur. Dans le cadre de ce projet, comme l'objectif était de compter le nombre d'occurences de chaque mot, la phase de Mapping produit sur chaque ordinateur distant un fichier qui a chaque mot (clé), associe la valeur 1. Dans cette partie, les occurences de chaque mot ne sont donc pas additionnées (cette étape se fera dans la phase reduce), et les fichiers de la phase de Mapping peuvent donc comporter plusieurs fois la même clé
- Shuffling: Dans cette étape, les mots (clés) identiques sont regroupés dans un même fichier, puis ces fichiers sont redistribués sur les différents ordinateurs, de sorte à regrouper sur chaque ordinateur l'ensemble des clés ayant la même valeur
- Reducing: Cette étape permet sur chaque ordinateur de fusionner les différents fichiers obtenus lors de la phase de Shuffling, de sorte à obtenir un fichier par mot, contenant le nombre d'occurences de ce mot dans le fichier de départ
- Collect: La dernière étape permet de finalement récupérer sur l'ordinateur qui a lancé la requête l'ensemble des fichiers reduce produits par les ordinateurs distants

## 3.2 Présentation générale de l'implémentation

Dans la première version de mon implémentation de l'algorithme de MapReduce, j'ai utilisé des connections ssh entre les différents ordinateurs du réseau, afin de permettre l'exécution de commandes et le partage de fichiers à distance et ainsi lancer les différentes étapes de l'algorithme MapReduce. J'ai utilisé pour cela la classe `ProcessBuilder` de Java notamment. Cependant, le nombre de connections ssh étant limité sur les ordinateurs utilisés, j'ai finalement choisi d'utiliser des connections TCP (classe `Socket` de Java) entre les différents ordinateurs pour leur permettre de communiquer.

Les différents échanges entre les ordinateurs du réseau devaient permettre l'envoi de fichiers, la création de dossiers et l'exécution de commandes à distance, pour le bon fonctionnement de MapReduce. Comme il s'agit d'opérations sensibles, j'ai cherché dans ce projet à assurer une connection sécurisée et chiffrée entre les ordinateurs du réseau. Pour cela, j'ai créé une classe `CommunicationManager`, permettant à deux ordinateurs distants de s'échanger des fichiers et de communiquer via une connection chiffrée avec l'algorithme AES, à l'aide des classes `CipherInputStream` et `CipherOutputStream` de Java. Pour que la connection soit sécurisée, la phase d'initialisation de l'algorithme génère des clés aléatoires et les partage sur les ordinateurs du réseau avec ssh. De cette manière, la génération et le partage des clés sont sécurisés, et ces clés permettent ensuite à chaque ordinateur du réseau de communiquer avec les autres de manière chiffrée.

## 3.3 Déploiement de l'algorithme

Pour déployer l'algorithme de MapReduce et permettre la communication entre les machines du réseau, j'ai choisi d'utiliser un script bash, qui fonctionne de la manière suivante:

- Le script prend en entée la liste des adresses des ordinateurs distants sur lesquels on veut déployer MapReduce, ainsi que le nom du fichier qui y sera exécuté (slave.jar), pour écouter les requêtes de l'ordinateur maître (qui lance le protocole)
- Un dossier temporaire est créé sur les ordinateurs distants, en vue de stocker l'ensemble des fichiers relatifs à l'exécution de MapReduce
- Un fichier contenant une clé aléatoire de 32 bytes est générée sur l'ordinateur maître, puis ce fichier est envoyé sur les ordinateurs du réseau. C'est ce fichier qui permettra de chiffrer les communications
- Enfin, le fichier exécutable slave.jar est envoyé et exécuté sur les ordinateurs distants, en vue d'écouter et exécuter les requêtes du maître

## 3.4 Slave

Le fichier slave.jar exécuté sur les ordinateurs réseau permet de lancer un serveur par ordinateur, afin d'écouter et exécuter les requêtes du maître (qui initie la procédure de MapReduce). Chaque serveur utilise les `Thread` de Java pour pouvoir traiter des commandes de plusieurs clients en parallèle. Cependant, pour chaque connection avec un client, les commandes exécutées se font de manière séquentielle, ce qui permet de créer un dossier à distance, puis d'y copier un fichier par exemple, en ayant la garantie que le dossier a bien été créé avant la copie du fichier. Ainsi, dès qu'un ordinateur distant reçoit une connection, il exécute dans un Thread séparé l'ensemble des commandes envoyées par le Client de manière séquentielle, jusqu'à reçevoir un message EOF (End Of File). Si tout s'est passé correctement, le serveur renvoie un acquittement au client, et la connection se ferme.

## 3.5 Master

Une fois les fichiers slave.jar déployés et les serveurs distants démarrés, le maître peut commencer l'exécution de l'algorithme de MapReduce. Pour cela, il peut prendre en paramètres un dossier "split" contenant la liste des fichiers à déployer sur les ordinateurs distants (cette version permet donc de tester l'algorithme de MapReduce à partir de splits déjà créés, et n'est à utiliser que pour le débuggage). La version complète de l'algorithme prend en entrée le fichier dont on veut compter les occurences des mots, et créé automatiquement les fichiers splits en découpant le fichier donné en autant de parties qu'il y a d'ordinateurs sur le réseau. J'ai choisi de découper le fichier de sorte à obtenir des splits de taille similaire, tout en garantissant qu'un même mot du fichier de base ne puisse pas être découpé sur deux splits différents. Ainsi, chaque split contient des mots entiers. Dans la version débug, on donne en paramètre de l'algorithme le dossier contenant les fichiers splits, mais dans la version complète, le dossier "splits" est automatiquement créé dans le répertoire courant, afin de contenir les différentes sous-parties du fichier de base.

Le master va ensuite charger la clé de chiffrement pour chiffrer les communications, puis envoyer les splits sur les ordinateurs distants (un fichier split par ordinateur).

Une fois cette phase terminée pour tous les ordinateurs du réseau, le master passe à l'étape de "Mapping" et envoie une commande à chaque ordinateur distant pour exécuter la procédure "Mapping" sur le fichier qu'il a reçu. Chaque ordinateur distant va donc créer un dossier maps, qui contiendra le fichier split correspondant sous la forme d'une liste de paires (mot, nombre), en conservant les doublons.

Une fois la phase de "Mapping" terminée sur l'ensemble des ordinateurs distants, le master envoie aux ordinateurs distants la liste des ordinateurs présents sur le réseau, en vue de démarrer la phase "Shuffling". Dans cette étape, chaque ordinateur du réseau va créer un dossier "shuffles", contenant un fichier par clé à partir du fichier map qu'il a créé lors de la phase "Mapping". Chaque fichier dans le dossier "shuffles" est nommé à partir du hash du mot qu'il représente, puis est envoyé sur un ordinateur distant. Pour faire en sorte que les fichiers se répartissent de manière uniforme sur l'ensemble des machines du réseau, on calcule pour chaque fichier $i = hash(clé)\ \%\ nombre\_machines$, et on envoie le fichier à la machine n°i sur le réseau, dans le dossier "shufflereceived". De cette manière, on garantit que les clés ayant le même hash sont envoyées sur un même ordinateur.

Enfin, la phase de "Reducing" est démarrée par le master sur les ordinateurs du réseau, afin de fusionner les fichiers ayant le même hash dans le dossier "shufflereceived". Ainsi, après cette étape, les ordinateurs distants possèdent un dossier "reduces", contenant un fichier par mot, ainsi que son nombre d'occurences dans le fichier initial.

Dans la dernière phase, le master peut donc collecter l'ensemble des fichiers des les dossiers "reduces" des ordinateurs distants afin de récupérer les résultats de l'algorithme MapReduce. On obtient ainsi sur l'ordinateur master, qui a lancé la procédure de MapReduce sur le fichier de base, un dossier "reduces" contenant un fichier par mot du fichier de base, nommé à partir du hash du mot. Chaque fichier contient alors un mot du fichier de base, ainsi que son nombre d'occurences.

# 4 Résultats

Pour tester le bon fonctionnement de l'algorithme de MapReduce, on peut commencer par utiliser les fichiers splits suivants:

```
$ cat S0.txt
Deer Beer River
$ cat S1.txt
Car Car River
$ cat S2.txt
Deer Car Beer
```

Il suffit pour cela de se placer dans le dossier "Testing" et de lancer la commande "`make test`. Cette commande va lister 3 ordinateurs disponibles pour lancer l'algorithme de MapReduce, et déployer sur ces ordinateurs l'exécutable `slave.jar`, en vue d'initialiser l'algorithme. La commande va ensuite utiliser les fichiers S0.txt, S1.txt et S2.txt présents dans le dossier Testing/test_splits pour démarrer l'algorithme de MapReduce, en exécutant `master.jar`.

Une fois l'exécution terminée, on obtient les chronométrages suivants pour les différentes phases de l'algorithme:

- Split: 162 ms
- Map: 100 ms
- Shuffle: 127 ms
- Reduce: 92 ms
- Collect: 94 ms

On observe que l'algorithme s'est exécuté de manière attendue, car un dossier reduces a été créé sur l'ordinateur maître (qui a lancé `master.jar`). Ce dossier contient bien un fichier par mot, chaque fichier étant nommé à partir du hash du mot. Chaque fichier contient également le nombre attendu d'occurences pour chaque mot:

```
$ cat reduces/2066512.txt
Beer 2
$ cat reduces/2126094.txt
Deer 2
$ cat reduces/67508.txt
Car 3
$ cat reduces/78973420.txt
River 2
```

Les chronométrages n'ont que peu de sens pour un fichier aussi petit que celui utilisé dans ce premier exemple (le fichier correspondant aux splits S0.txt, S1.txt et S2.txt ne contiendrait que 9 mots). Pour avoir des résultats plus réalistes, on peut tester le fichier `domaine_public_fluvial.txt` (de taille 72K), en déployant l'algorithme de MapReduce sur 3 ordinateurs distants avec la commande:
`$ ./run.sh test_files/domaine_public_fluvial.txt 3`

On alors obtient les chronométrages suivants:

- Split: 208 ms
- Map: 121 ms
- Shuffle: 3924 ms
- Reduce: 121 ms
- Collect: 133 ms

On observe que la phase "Shuffle" est clairement la plus longue, car elle doit traiter un grand nombre de fichiers (un fichier par mot), et envoyer cet ensemble de fichiers sur d'autres ordinateurs distants à partir du hash de chaque mot. Le grand nombre de mots dans chaque fichier fait qu'un grand nombre de connections TCP est créé dans cette étape, ce qui en fait l'étape la plus longue. Les autres étapes au contraire transmettent un nombre moins important de fichiers sur le réseau, ou effectuent des calculs en local, ce qui explique la rapidité de leur exécution.

Pour mesurer l'influence du nombre de machines dans l'algorithme de MapReduce, on peut utiliser le fichier `domaine_public_fluvial.txt`, et exécuter l'algorithme de MapReduce plusieurs fois en faisant varier le nombre de machines utilisées.

En choisissant un nombre de machines égal à 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20 puis 25, on obtient le graphique suivant:

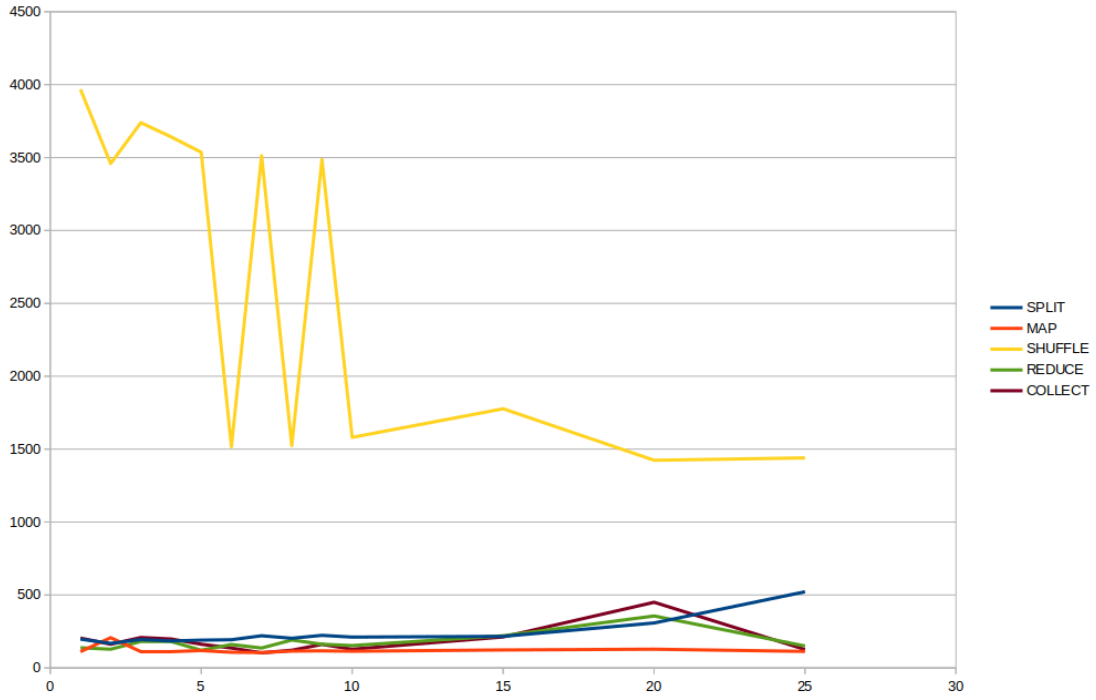


Figure 2: Durée d'exécution en millisecondes des différentes phases de l'algorithme de MapReduce, en fonction du nombre de machines déployées

On observe que la phase la plus longue reste dans tous les cas la phase "Shuffle", ce qui est cohérent avec nos observations pour les fichiers S0.txt, S1.txt et S2.txt. On observe également que la durée d'exécution de la phase "Shuffle" est décroissante en fonction du nombre de machines, ce qui est cohérent avec la loi d'Amdahl. Pour obtenir des mesures plus précises, il aurait fallu exécuter l'algorithme sur de plus gros fichiers, et faire des moyennes avec le même nombre d'ordinateurs, pour avoir des courbes moins bruitées. Cependant avec mon implémentation, j'ai fait face à des erreurs de timeout, qui font que le serveur distant ne répond pas aux requêtes du maître, sans provoquer de message d'erreur. Ces timeouts n'apparaissant que de manière aléatoire, j'ai eu des difficultés à effectuer des mesures plus précises de durées pour chaque phase.

# 5 Conclusion

Pour conclure, l'algorithme de MapReduce peut avoir un intérêt pour compter le nombre d'occurences de mots dans un fichier volumineux, si on a à notre disposition un grand nombre d'ordinateurs. Cet algorithme, qui possède l'avantage d'être résistant aux pannes, et qui exploite le calcul en parallèle pour réaliser des taches importantes en un temps réduit, nécessite cependant de nombreuses communications à distance entre les ordinateurs du réseau, ce qui le rend par exemple beaucoup moins rapide qu'un algorithme séquentiel, fonctionnant sur une machine unique, dans le cas de petits fichiers.