INF 727 : Projet de systèmes répartis

Implémentation de Hadoop MapReduce "from scratch" en Python.

1 Introduction

Dans le cadre de l'enseignement INF 727 – Système Répartis Pour le BigData, j'ai implémenté à partir de zéros une version simple du concept MapReduce avec le langage python. Pour cela j'ai utilisé plusieurs ordinateurs d'une salle de TP des locaux de Télécom Paris connectés en réseaux afin de paralléliser le comptage des mots contenu au sein d'un fichier texte.

L'objectif recherché avec l'augmentation du volume de données n'est plus de centralisé les données sur une seule machine mais plutôt de distribuer le stockage et paralléliser les traitements sur plusieurs machines. Le fait de regrouper les ordinateurs entre eux créer un effet de synergie c'est-à-dire que la puissance combinée de plusieurs petits ordinateurs et supérieur à la puissance d'un gros serveur central.

Les avantages du calcul distribué sont les suivants :

- Scalabilité horizontale : Pour augmenter les performances et la capacité de stockage d'un cluster (groupement de plusieurs machines), il suffit de rajouter un ordinateur au cluster cela permet donc d'ajouter très facilement un nouveau nœud au cluster.
- Tolérance aux pannes : Pour être résilient face aux pannes (ex : pannes de nœuds très fréquents = machine qui tombe en panne, problème de connectivité entre les nœuds ou les casiers et le réseau), les données sont dupliquées sur plusieurs machines (3 fois par défaut). Dans Hadoop c'est HDFS qui permet d'être tolérant face aux pannes.
- Les goulots d'étranglement : Dans une architecture client/serveur où l'ensemble des données sont stockées sur un serveur central, les opérations de transfert de données qui se dirigent vers le serveur central engendrent un goulot d'étranglement. Dans une architecture distribuée, la charge de travail est répartie entre les différentes machines du cluster donc on ne retrouve pas ce problème de goulot d'étranglement.

2 Présentation du document

Dans ce document j'expliquerai les différentes étapes itératives de l'implémentation du concept MapReduce ainsi que l'architecture répartie que j'ai mise en place. J'expliquerai surtout les problèmes que j'ai rencontré tout le long de projet et quelles solutions j'ai mis en œuvre pour contourner ces derniers.

3 Présentation du fonctionnement de MapReduce

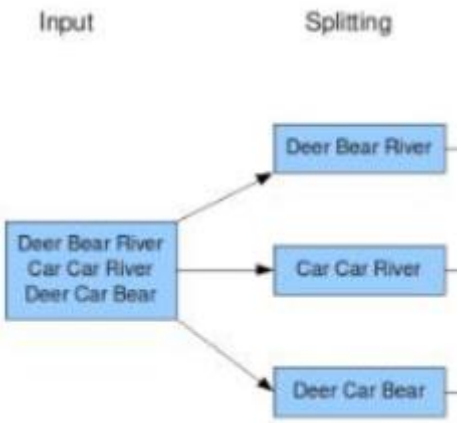
La première motivation de l'introduction de MapReduce concerne le traitement des tâches dit « embarrassingly parallel ». Cela est un abus de langage car contrairement à ce que l'on pourrait croire, « embarrassingly parallel » ne peut pas dire « embarrassant à paralléliser » mais plutôt « parallélisable à l'excès». Ainsi, MapReduce, n'a pas été conçu dans le but de paralléliser des tâches difficiles à paralléliser mais plutôt pour des tâches facilement parallélisable. En effet, MapReduce pourra fonctionner seulement sur une tâche qui peut se découper en plusieurs sous tâches indépendantes. Cela peut être perçu comme une limite de MapReduce mais il s'avère que dans le domaine de l'analyse de données, la majorité des applications de traitement de données, les données ont une structure régulière offrant ainsi l'avantage d'exploiter la puissance du parallèle.

Par ailleurs, MapReduce est un modèle algorithmique qui permet de penser le découpage d'un problème parrallélisable en plusieurs sous tâches indépendantes en suivant 4 phases.

Pour expliquer le fonctionnement de ces 3 phases, nous nous plaçons dans le contexte du comptage des mots d'un fichier texte.

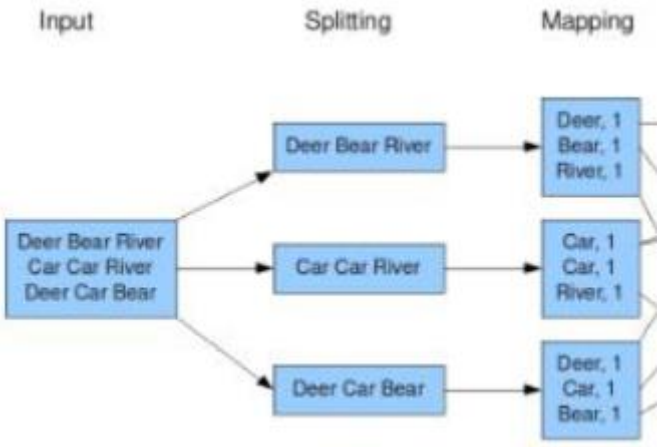
Phase 1 : Splitting

La phase de splitting découpe le texte d'entrée en plusieurs extrait (partition) en fonction du nombre de machine utilisé pour le comptage des mots. Pour chacune des partitions est assigné une machine. Ainsi chaque partition est envoyée à une machine escalve.



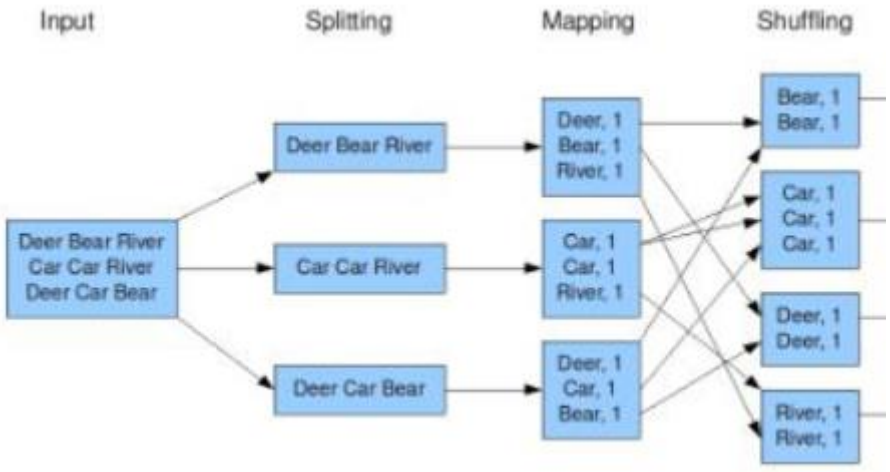
Phase 2 : Map

La phase de Map transforme les données en paires clé/valeur. Dans notre exemple, cela revient à extraire tous les mots du texte en entrée et leur associé la valeur 1.



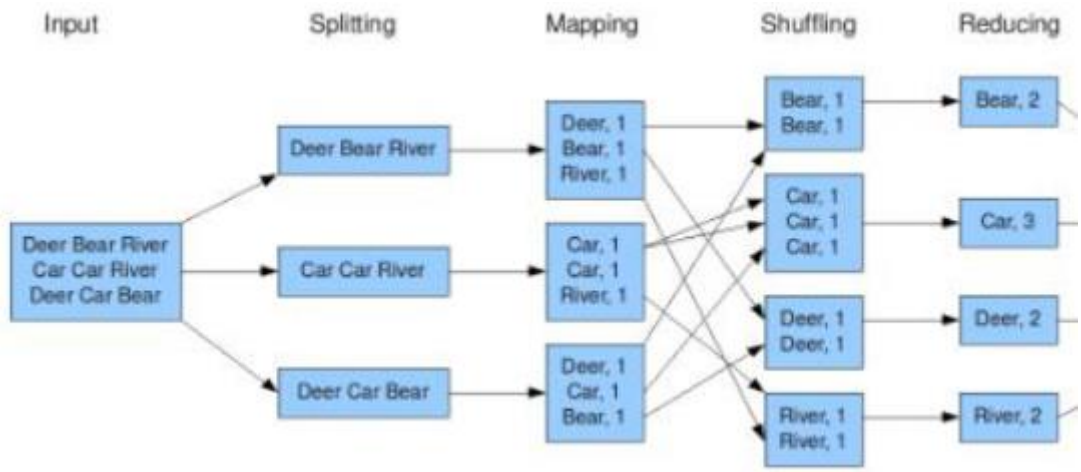
Phase 3 : Shuffle

Une fois la phase ***Map*** achevée sur toute les machines, la phase ***Shuffle*** peut être lancée. Cette dernière est en charge de trier toutes les paires clé/valeur générées par la phase ***Map***. De plus, la phase ***shuffling*** regroupe dans une liste chaque valeur éparpillées entre les différentes machines ayant la même clé. Autrement dit, la phase de ***shuffling*** redistribue les données en fonction des clés de sortie (produite par la fonction ***map***) de sorte que toutes les données appartenant à une clé se trouvent sur le même nœud.



Phase 4 : Reduce

L'algorithme MapReduce se conclu avec la phase « ***reducing*** ». La phase ***reduce*** concatène les différentes listes contenant les clés/valeurs crée par chaque machine durant la phase de ***shuffling*** au sein d'un unique fichier. Durant la phase de « ***reducing*** » il revient à l'utilisateur de choisir la fonction d'agrégat qu'il souhaite utiliser. Dans notre cas pour compter les mots la fonction d'agrégat qui est utilisée est la somme.



4 Implémentation du projet

L'implémentation du projet peut être découpé en 6 étapes :

- Phase d'initialisation : Déploiement de l'environnement sur chaque machine
- Phase de Splitting : Découpage du texte en plusieurs extraits qui seront traité par chaque machine
- Phase de Mapping : Construction de la paire clé/valeur
- Phase Shuffling : Redistribution des paires clé/valeur sur chaque machine
- Phase de Reducing : Regroupement des paires clé/valeur sur chaque machine
- Phase d'Assembling : Regroupement de toutes les paires clé/valeur préalablement regroupées par chaque machine durant la phase reducing.

Pour chaque itération de l'algorithme MapReduce nous allons passer en revu ces 6 phases.

4.1 1ère version de MapReduce avec 3 machines

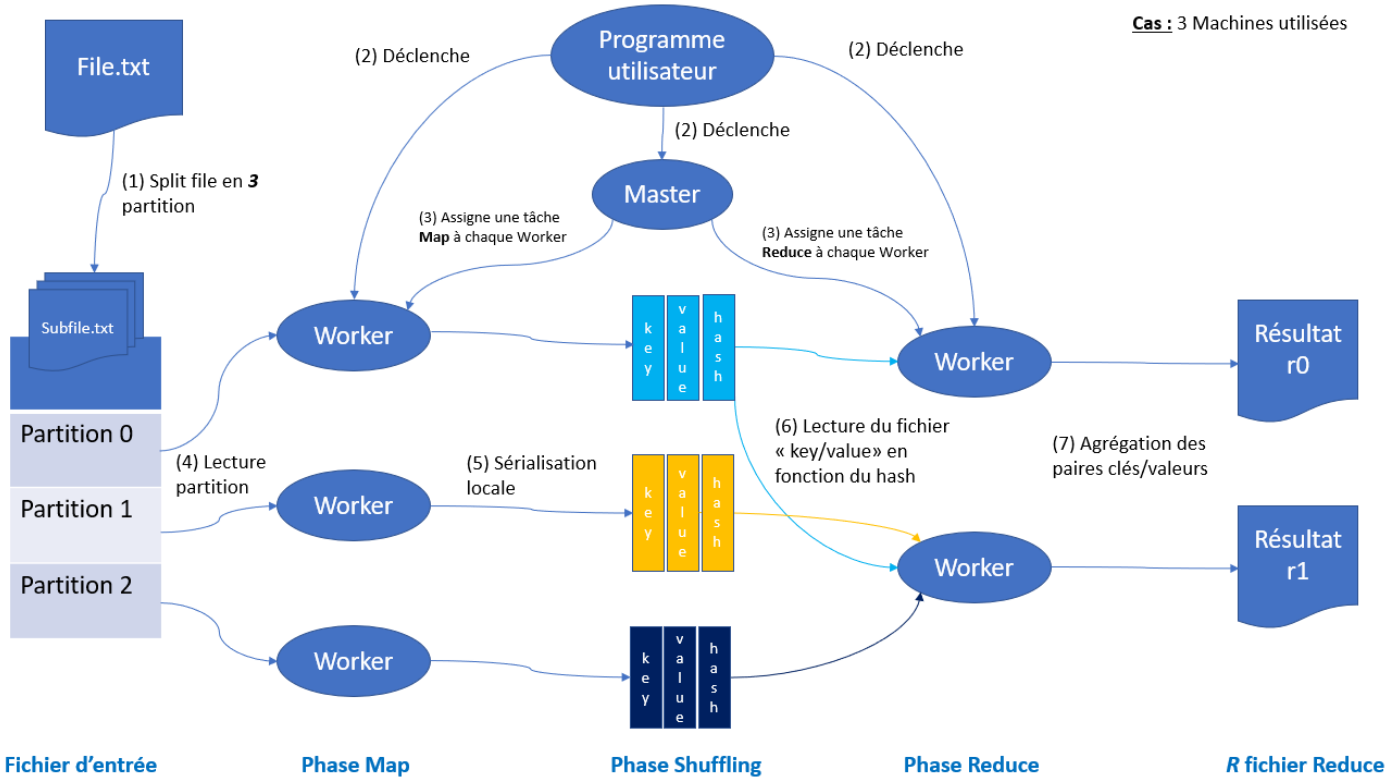
La première version de mon programme MapReduce fonctionne avec 3 machines. Pour expliquer son fonctionnement nous nous plaçons dans un contexte simple qui est le suivant :

Input.txt
Deer Bear River Car Car River Deer Car Bear

Objectif : Compter les mots du fichier texte **input.txt**

Nb_machine : 3

L'architecture répartie mise en place est la suivante :



Les modèles de communication déterminent le ou les responsables lorsqu'une requête est adressé au cluster. Deux machines arrivent à s'échanger des informations grâce à un protocole de communication qui est un ensemble de spécifications qui permettent à deux ordinateurs de communiquer. Le modèle de communication utilisé ici suit le paradigme Maître-Esclave. A travers ce type d'architecture, les requêtes adressées au cluster (groupement des machines) sont réparties entre chaque machine esclave par une machine maître. La gestion entière du cluster est adressée à la machine maître qui connait la répartition exacte des données sur chaque machine. La communication est donc bidirectionnelle et peut être facilement chainée sur le réseau sans créer de goulot d'étranglement. De plus la centralisation rendu possible grâce à la machine maître facilite l'administration du cluster.

De plus, le mode de partage des ressources dans l'architecture distribuée que j'ai mise en place au sein du cluster est celui du « ***shared nothing*** ». En effet, aucune ressource n'est partagée entre chacune des machines du cluster via le réseau LAN. En effet, les machines esclaves ne partagent ni la mémoire ni le disque dur. Chaque machine esclave utilise ses propres ressources.

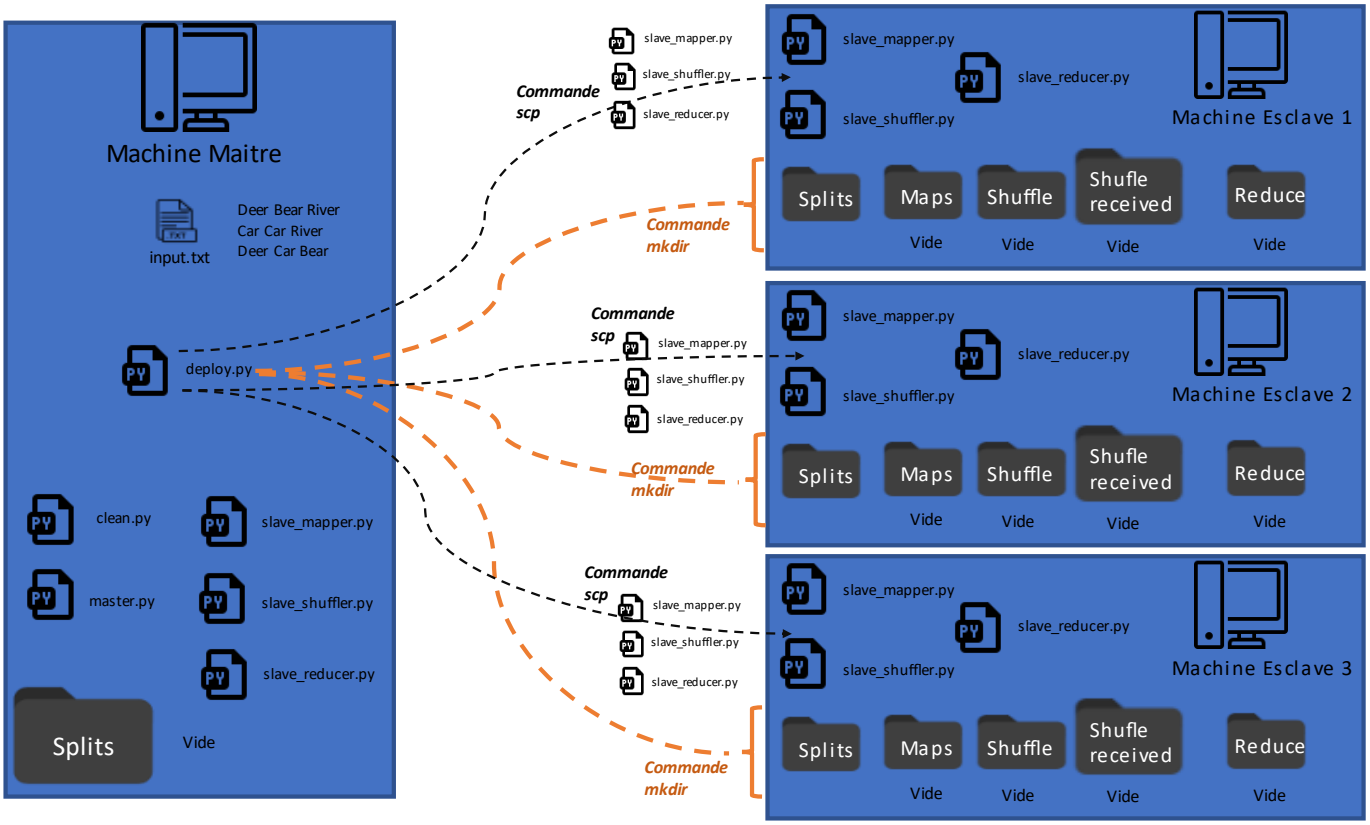
4.1.1 Phase de démarrage

La phase de démarrage est une étape nécessaire afin que chacune des machines disposent d'un environnement identique. Tout ce qui concerne la phase d'initialisation est contenu dans le fichier « ***deploy.py*** ». Ce fichier contient la liste des machines (ici 3 machines pour cette première version de l'algorithme) utilisée pour réaliser le job MapReduce. Sur chacune de ces machines les dossiers nécessaires au stockage des différents fichiers qui seront générées durant l'exécution du job MapReduce sont crées avec la commande bash ***mkdir.*** A noter que pour exécuter des commandes bash depuis un programme python j'ai utilisé la méthode ***subprocess().*** Les dossiers créés par le programme ***deploy.py*** sont les suivants :

- ***/tmp/hmichel-20*** : dossier qui contient l'ensemble des dossiers de chacune des phases ***map***, ***shuffling*** et ***reduce***
- ***/tmp/hmichel-20/splits*** : dossier qui contient une partition du texte qui a été découpé en ***N*** partition
- ***/tmp/hmichel-20/maps*** : dossier qui contient le fichier généré par la phase ***Mapping***.
- ***/tmp/hmichel-20/shuffles*** : dossier qui contient les fichiers générés par la phase ***Shuffling***.
- ***/tmp/hmichel-20/shufflesreceived*** : dossier qui contient les fichiers avec le hash correspondant à chaque clé (mot de la portion de texte).
- ***/tmp/hmichel-20/reduces*** : dossier qui contient le fichier généré par la phase ***Reducing***.

Ensuite les 3 fichiers python associés aux phases de ***Mapping***, ***Shuffling*** et ***Reducing*** sont copiés sur chacune des machines. Ces fichiers permettront à la machine maître de de donner à distance l'ordre d'exécution de la phase en question à chacune des machines. Ces fichiers sont les suivants :

- ***slave_mapper.py***
- ***slave_shuffler.py***
- ***slave_reducer.py***



Phase de déploiement avec le fichier « deploy.py »

4.1.2 Phase de File Splitting

La phase de ***File Splitting*** consiste à découper le texte d'entrée en ***P*** partitions. Par ailleurs le nombre de partitions ***P*** correspond tout simplement au nombre de machine ***N*** participant au job MapReduce. Ici cette phase est prise en charge entièrement par la machine maître car elle seule contient le fichier de texte d'entrée. Afin que les partitions soient toutes équilibrées entre les machines la phase de splitting réalise les opérations suivantes :

1. Nettoyage du fichier : Suppression des lignes vides du fichier texte avec la fonction python ***strip()***

2. Calcul du nombre de répartition des lignes pour chaque partition : Le nombre total des lignes du fichier d'entrée est divisé (division entière) par le nombre de machine. De cette façon on obtient un nombre de ligne identique pour chaque partition. Mais il se peut qu'il reste encore des lignes du texte qui ne sont pas pris en compte car nous effectuons une division entière.

   split_line = nb_line // nb_machine

3. Le nombre de ligne restant est calculé avec le modulo du nombre de machine

   remaining_line = nb_line % nb_machine

4. Création des intervalles pour sélectionner les ***N*** lignes qui seront intégrées dans chaque partition

   int1 = split_line

   int2 = 2 * split_line

   int3 = 3 * split_line

   …

   int30 = 30*(split_line) + remaining_line

   En généralisant ça donne : int(i) = i * split_line et int_last(i) = i*(split_line) + remaining_line

5. Enfin il convient à l'aide des différents intervalles calculés à l'étape 3 de créer les dataframes dont chacune des lignes du dataframe contiendra une ligne de la partition de texte.

   df1 = df[0:int1]

   df2 = df[int1:int2]

   df3 = df[int2:int3]

   …

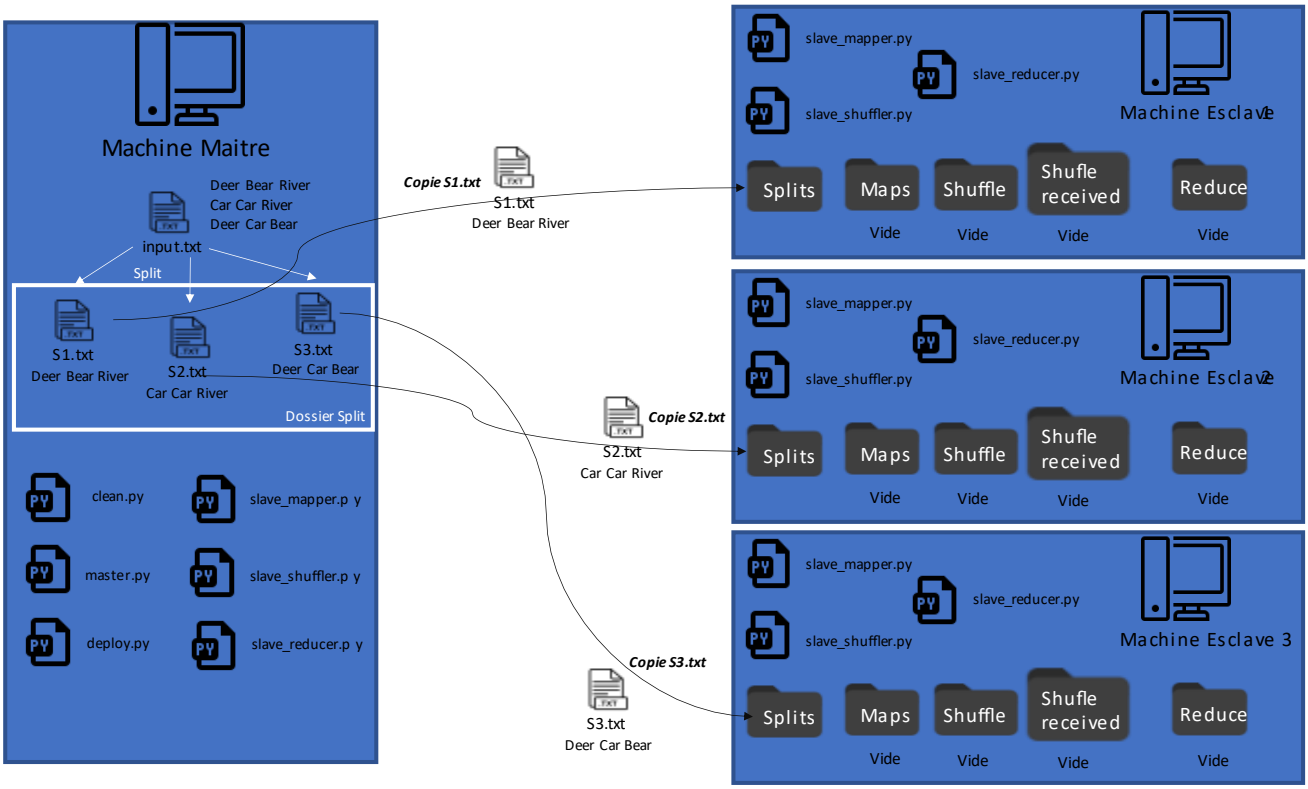
   df30 = df[int29:int30]

   En généralisant cela donne df(i) = df[int(i-1) :int(i)]

6. On sauvegarde le dataframe au format ***.txt*** sur le dossier ***splits*** de la machine maître.

La machine maître contient alors dans le dossier ***splits*** les ***P*** partitions du texte d'entrée : ***{s1.txt, s2.txt, …, sN.txt}***

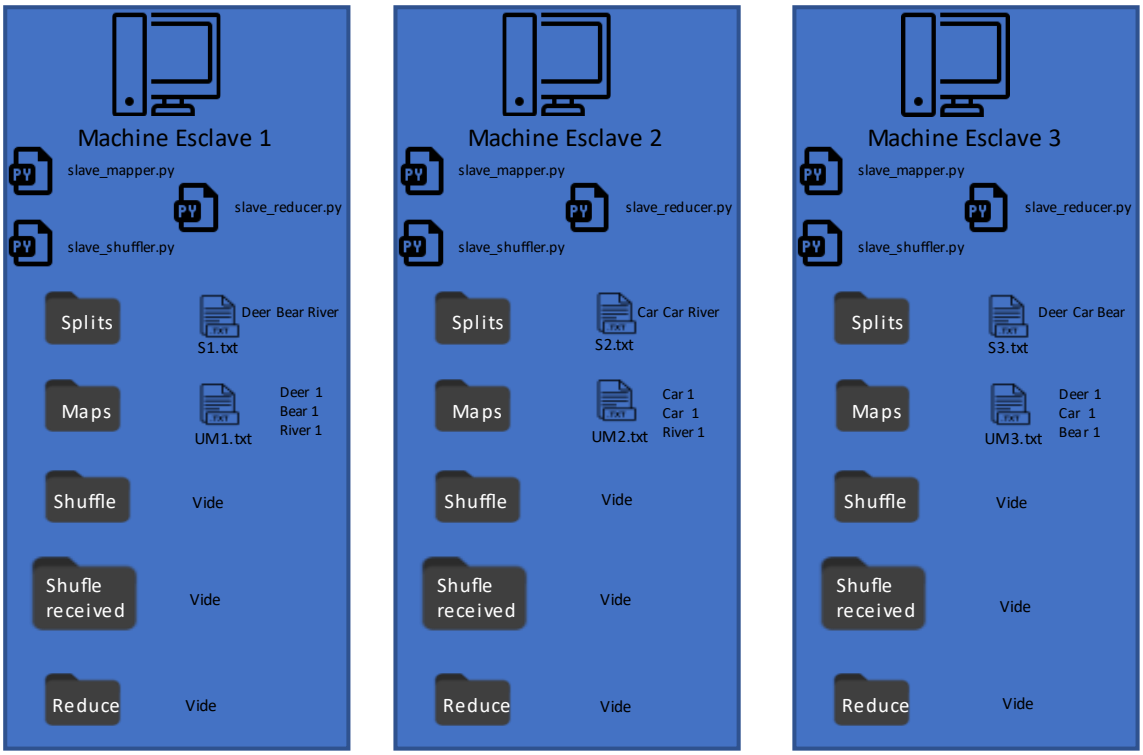
Enfin la dernière étape consiste en copier via la commande ***ssh scp*** l'ensemble de ces fichiers sur chacune des machines distantes depuis la machine maître.



Phase de splitting avec le fichier « master.py »

4.1.3 Phase de Mapping

L'étape de Mapping consiste pour une machine esclave à découper en mot la partition qui vient d'être envoyé par la machine maître. Cette phase va donc créer sur chaque machine esclave un fichier ***UMX.txt*** avec ***X*** qui correspond au numéro du fichier split envoyé par la machine maître. Pour découper le texte en mot, le programme ***slave_mapping.py*** lit toutes les lignes de la partition les unes après les autres afin de découper chaque ligne en mots. Ensuite à chaque mot est attribué la valeur 1. De cette façon, pour chaque mot du fichier split, nous obtenons une paire clé/valeur avec la clé qui correspond à un mot accompagné de sa valeur qui est 1. Ainsi pour chacune des machines le fichier Mapping est généré de la manière suivante :



Phase de mapping avec le fichier « slave_mapper.py »

4.1.4 Phase de Shuffling

La phase de shuffling consiste à regrouper les paires clés/valeurs dont les clés sont identiques.

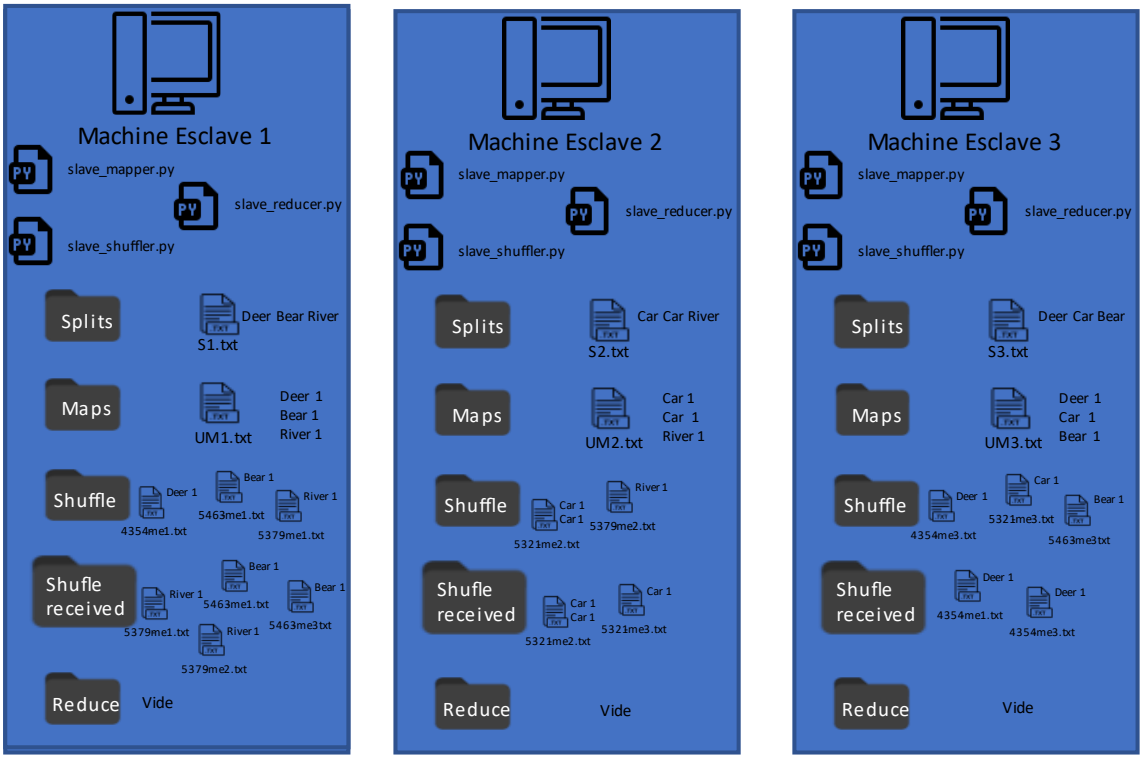
Tout d'abord afin de regrouper les clés identiques au sein d'une machine esclave, nous parcourons la liste de paires clé/valeur du fichier ***UMX.txt*** généré par la phase de Mapping. Ensuite pour chaque clé nous calculons un ***hash***. Le code hash peut être considéré comme un ***Id*** unique associé à chaque clé (i.e mot). De cette façon les clés identiques se retrouveront au sein du même fichier de chacune des machines esclaves. La convention de nommage du fichier est le suivant : ***hash-hostname.txt*** (ex : 54636-tp1a201.txt). A l'intérieur de ce fichier, chaque paire clés/valeurs similaires sont enregistrées. Ce fichier contient alors la liste des mots qui sont apparu plusieurs fois dans la partition de texte. Enfin, l'ensemble de ces fichiers sont alors stockés dans le dossier ***shuffle.***

Cependant il se peut que des mots se retrouvent également dans une partition d'une autre machine esclave. Cela signifie donc qu'il convient également regrouper les paires clé/valeur identifié par chacune des machines esclaves entre elles. Pour ce faire, grâce au code hash contenu dans le nom de chacun des fichiers du dossier ***shuffle*** nous calculons le modulo de façon à identifier à quelle machine le fichier en question doit être envoyé. Le pseudo-code est le suivant :

Calcul du hash pour connaître le numéro de la machine

```
    nb_machines = 3
    hashcode = hashcode
    num_machine = hashcode % nb_machines
    if num_machine == 0:
        id_machine = 10
    elif num_machine == 1:
        id_machine = 11
    elif num_machine == 2:
        id_machine = 13
```

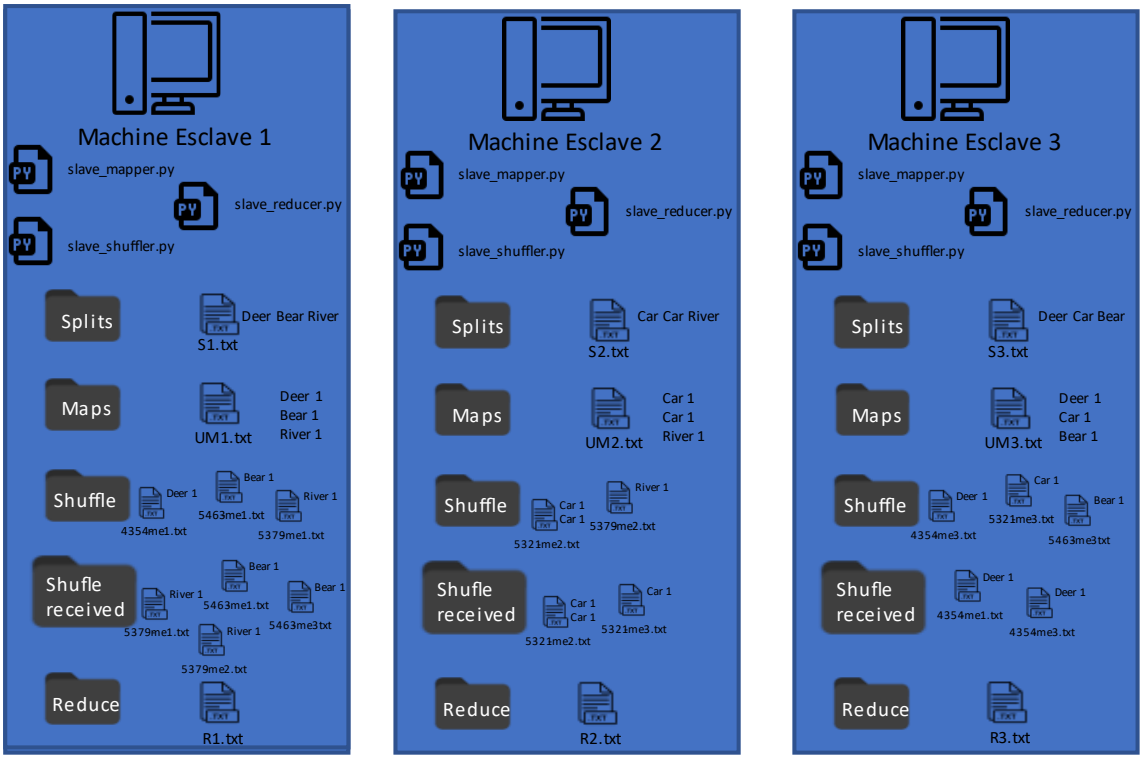
Chacun de ces fichiers générés par la phase de ***shuffling*** qui sont stockés dans le dossier ***shuffle*** sont envoyé aux autres machines esclaves via une bash ***scp*** et ce en fonction du hash qui vient d'être calculé précédemment. Ainsi, les noms des fichiers comportant le même code hash sont envoyés et stockés dans le dossier ***shufflereceived*** de la machine esclave en question. Finalement le dossier ***shufflereceived*** de chacune de ces dernières contient un ensemble de fichiers dont les noms des fichiers possèdent un code hash identique.



Phase de reducing avec le fichier « slave_shuffler.py »

4.1.5 Phase de reducing

Le phase de reducing consiste à regrouper sur chacune des machines les paires clés/valeurs contenu au sein de chacun des noms de fichier possédant le même code hash. Finalement l'objectif de cette phase est de fusionner entre eux les fichiers qui possèdent des paires paires clés/valeurs identiques en y appliquant une fonction d'agrégation. Dans notre cas d'étude il s'agit de la somme.



Phase de reducing avec le fichier « slave_reducer.py »

4.1.5 Phase d'assembling

A travers cette dernière étape, l'objectif de la machine maître est de copier les fichiers ***reduce {R1.txt, R2.txt, R3.txt}*** de chaque machine générée lors de la phase de ***reducing*** dans son répertoire ***reduces*** de manière à concaténer tous ces fichiers au sein d'un unique fichier qui contiendra alors chaque paire clé/valeur. Ainsi ce fichier généré contiendra une liste des mots suivi de son nombre d'occurrence au sien du fichier ***input.txt***

WC_result.txt
Beer 2 Car 3 Deer 2 River 2

4.1.6 Processus de lancement de la première version du programme

Depuis la machine maitre :

1- Lancer le programme clean.py
2- Lancer le programme deploy.py
3- Lancer le programme master.py

Phase de Mapping

```
------- STARTING MAPPING PHASE ... -------

Start Mapping Phase for machine 0
0 out: 'b'''
0 err: 'b'''
0 exit: 0
Mapping Phase finished for machine 0
Start Mapping Phase for machine 1
1 out: 'b'''
1 err: 'b'''
1 exit: 0
Mapping Phase finished for machine 1
Start Mapping Phase for machine 2
2 out: 'b'''
2 err: 'b'''
2 exit: 0
Mapping Phase finished for machine 2

------- MAPPING PHASE FINISHED-------

MAPPING --- 0.33518028259277344 seconds ---
```

Phase de shuffling

```
-------STARTING SHUFFLING PHASE ... -------

Start Shuffling Phase for machine 0
0 out: 'b"tp-1a201-11 out: 'b'''\ntp-1a201-11
xit: 0\n"'
0 err: 'b'''
0 exit: 0
Shuffling Phase finished for machine 0
Start Shuffling Phase for machine 1
1 out: 'b"tp-1a201-11 out: 'b'''\ntp-1a201-11
1 err: 'b'''
1 exit: 0
Shuffling Phase finished for machine 1
Start Shuffling Phase for machine 2
2 out: 'b"tp-1a201-13 out: 'b'''\ntp-1a201-13
xit: 0\n"'
2 err: 'b'''
2 exit: 0
Shuffling Phase finished for machine 2

------- SHUFFLING PHASE FINISHED -------

SHUFFLING --- 1.1905457973480225 seconds ---
```

Phase de Reducing

```
------- STARTING REDUCING PHASE ... -------

Start Reducing Phase for machine 0
0 out: 'b'''
0 err: 'b'''
0 exit: 0
Reducing Phase finished for machine 0
Start Reducing Phase for machine 1
1 out: 'b'''
1 err: 'b'''
1 exit: 0
Reducing Phase finished for machine 1
Start Reducing Phase for machine 2
2 out: 'b'''
2 err: 'b'''
2 exit: 0
Reducing Phase finished for machine 2

------- REDUCING PHASE FINISHED -------

REDUCING --- 0.31209492683410645 seconds ---

Total execution time --- 1.83793306350708 seconds ---
```

Résultat

```
-----------------------------------

RESULT ==>  b'Beer 2Car 3River 2Deer 2'

-----------------------------------
```

Résultats :

Temps Mapping	Temps Shuffling	Temps Reducing	Temps Total
0.33 s	**1.19 s**	**0.31 s**	**1.84 s**

Avec cette première itération du programme MapReduce nous remarquons que l'étape de shuffling est la phase la plus chronophage. Ceci s'explique simplement par le fait que durant cette étape beaucoup d'opérations d'écritures, et de lectures s'enchaine en plus des échanges multiples de fichiers entre les machines.

4.1.5 Application de la 1ère version MapReduce sur différents fichiers

Fichier : deontologie_police_nationale.txt (8 ko)

Condition du test : 3 machines et time = 5 s

Temps Mapping	Temps Shuffling	Temps Reducing	Temps Total
0.35 s	**Machine 0 : timeout** **Machine 1 : timeout** **Machine 3 : timeout**	**-**	**-**

	`-------STARTING SHUFFLING PHASE ... -------`
	
	`Start Shuffling Phase for machine 0`
	`0 timeout`
	`Start Shuffling Phase for machine 1`
	`1 timeout`
	`Start Shuffling Phase for machine 2`
	`2 timeout`		

Sur un fichier un peu plus conséquent en termes de taille (8 ko contre 1ko pour le fichier intput.txt) la phase de shuffling ne fournit aucun résultat. En effet la commande bash lançant l'exécution de fichier ***slave_shuffler.py*** retourne un timeout. Le timeout a été configuré par défaut à 5 sec.

En configurant le timeout à 120 s nous obtenons.

Condition du test : 3 machines et time = 120 s

Temps Mapping	Temps Shuffling	Temps Reducing	Temps Total
0.36 s	**68.35 s**	**0.34**	**69.05 s**

```
------- STARTING MAPPING PHASE ... -------

Start Mapping Phase for machine 0
0 out: 'b'''
0 err: 'b'''
0 exit: 0
Mapping Phase finished for machine 0
Start Mapping Phase for machine 1
1 out: 'b'''
1 err: 'b'''
1 exit: 0
Mapping Phase finished for machine 1
Start Mapping Phase for machine 2
2 out: 'b'''
2 err: 'b'''
2 exit: 0
Mapping Phase finished for machine 2

------- MAPPING PHASE FINISHED-------

MAPPING --- 0.364063739776611 33 seconds ---
```

```
------- SHUFFLING PHASE FINISHED -------

SHUFFLING --- 68.34784579277039 seconds ---
```

```
------- STARTING REDUCING PHASE ... -------

Start Reducing Phase for machine 0
0 out: 'b'''
0 err: 'b'''
0 exit: 0
Reducing Phase finished for machine 0
Start Reducing Phase for machine 1
1 out: 'b'''
1 err: 'b'''
1 exit: 0
Reducing Phase finished for machine 1
Start Reducing Phase for machine 2
2 out: 'b'''
2 err: 'b'''
2 exit: 0
Reducing Phase finished for machine 2

------- REDUCING PHASE FINISHED -------

REDUCING --- 0.337761640548706 05 seconds ---

Total execution time --- 69.04981923103333 seconds ---
```

Un extrait du fichier généré par le programme pour le fichier **deontologie_police_nationale.txt** est disponible en ANNEXE 4.1

Fichier : domaine_public_fluvial.txt (71 ko)

Condition du test : 3 machines et time = 120 s

Temps Mapping	Temps Shuffling	Temps Reducing	Temps Total
0.40 s	**Machine 0 : timeout Machine 1 : timeout Machine 2 : 240s**	**0.40s**	**240.87 s**

```
------- STARTING MAPPING PHASE ... -------

Start Mapping Phase for machine 0
0 out: 'b'''
0 err: 'b'''
0 exit: 0
Mapping Phase finished for machine 0
Start Mapping Phase for machine 1
1 out: 'b'''
1 err: 'b'''
1 exit: 0
Mapping Phase finished for machine 1
Start Mapping Phase for machine 2
2 out: 'b'''
2 err: 'b'''
2 exit: 0
Mapping Phase finished for machine 2

------- MAPPING PHASE FINISHED-------

MAPPING --- 0.406530141830444 34 seconds ---
```

```
-------STARTING SHUFFLING PHASE ... -------

Start Shuffling Phase for machine 0
0 timeout
Start Shuffling Phase for machine 1
1 timeout
Start Shuffling Phase for machine 2
2 out: 'b"tp-1a201-13 out: 'b'''\ntp-1a201-13
xit: 0\ntp-1a201-13 out: 'b'''\ntp-1a201-13 e
------- SHUFFLING PHASE FINISHED -------

SHUFFLING --- 240.05611538887024 seconds ---
```

```
------- STARTING REDUCING PHASE ... -------

Start Reducing Phase for machine 0
0 out: 'b'''
0 err: 'b'''
0 exit: 0
Reducing Phase finished for machine 0
Start Reducing Phase for machine 1
1 out: 'b'''
1 err: 'b'''
1 exit: 0
Reducing Phase finished for machine 1
Start Reducing Phase for machine 2
2 out: 'b'''
2 err: 'b'''
2 exit: 0
Reducing Phase finished for machine 2

------- REDUCING PHASE FINISHED -------

REDUCING --- 0.4052693843841553 seconds ---

Total execution time --- 240.86805129051208 seconds ---
```

On augmente le délai du timeout à 400 s

Condition du test : 3 machines et time = 400 s			
Temps Mapping	**Temps Shuffling**	**Temps Reducing**	**Temps Total**
0.38 s	**271.6 s**	**0.41s**	**272.4 s**

```
------- STARTING MAPPING PHASE ... -------

Start Mapping Phase for machine 0
0 out: 'b'''
0 err: 'b'''
0 exit: 0
Mapping Phase finished for machine 0
Start Mapping Phase for machine 1
1 out: 'b'''
1 err: 'b'''
1 exit: 0
Mapping Phase finished for machine 1
Start Mapping Phase for machine 2
2 out: 'b'''
2 err: 'b'''
2 exit: 0
Mapping Phase finished for machine 2

------- MAPPING PHASE FINISHED-------

MAPPING --- 0.3827955722808838 seconds ---
```

```
Shuffling Phase finished for machine 2

-------- SHUFFLING PHASE FINISHED --------

SHUFFLING --- 271.6040790081024 seconds ---
```

```
------- STARTING REDUCING PHASE ... -------

Start Reducing Phase for machine 0
0 out: 'b'''
0 err: 'b'''
0 exit: 0
Reducing Phase finished for machine 0
Start Reducing Phase for machine 1
1 out: 'b'''
1 err: 'b'''
1 exit: 0
Reducing Phase finished for machine 1
Start Reducing Phase for machine 2
2 out: 'b'''
2 err: 'b'''
2 exit: 0
Reducing Phase finished for machine 2

------- REDUCING PHASE FINISHED -------

REDUCING --- 0.4131288528442383 seconds ---

Total execution time --- 272.4001626968384 seconds ---
```

Un extrait du fichier WC_result.txt pour le fichier **domaine_public_fluvial.txt** est disponible en ANNEXE 4.2

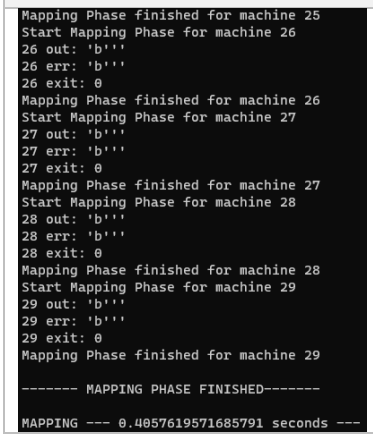
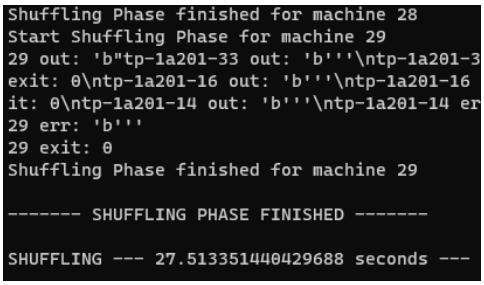
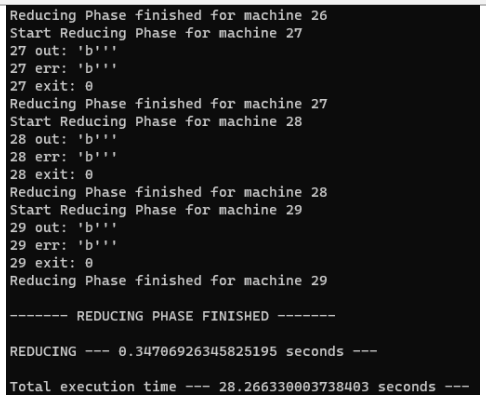
Conclusion

Avec cette première itération de mon programme nous remarquons que la phase de shuffling est chronophage. De manière globale la première version du MapReduce que j'ai implémentée exécute énormément d'opération de lecture, d'écriture et de copie de fichiers entre les machines. La prochaine étape sera d'observer l'impact du nombre de machine sur le temps d'exécution de la 1 ère version de MapReduce.

4.1.6 Application de la 1ère version MapReduce sur différents fichiers avec 30 machines

Le passage à 30 machines, ne change rien au fonctionnement des 6 phases que j'ai expliqué précédemment. La seule différence est que maintenant 30 machines esclaves sont utilisées pour réaliser le job MapReduce.

Fichier : deontologie_police_nationale.txt (8 ko)

Condition du test : 30 machines et time = 400 s			
Temps Mapping	**Temps Shuffling**	**Temps Reducing**	**Temps Total**
0.36 s (Avec 3 machines)	**68.35 s** (Avec 3 machines)	**0.34** (Avec 3 machines)	**69.05 s** (Avec 3 machines)
0.40 s	**27.5 s**	**0.41s**	**28,26 s**
			

Fichier : domaine_public_fluvial.txt (71 ko)

Condition du test : 30 machines et time = 400 s

Temps Mapping	Temps Shuffling	Temps Reducing	Temps Total
0.38 s (Avec 3 machines)	**271.6 s** (Avec 3 machines)	**0.41 s** (Avec 3 machines)	**272.4 s** (Avec 3 machines)
0.40 s	**81.14 s**	**0.50 s**	**81.05 s**

```
Mapping Phase finished for machine 26
Start Mapping Phase for machine 27
27 out: 'b'''
27 err: 'b'''
27 exit: 0
Mapping Phase finished for machine 27
Start Mapping Phase for machine 28
28 out: 'b'''
28 err: 'b'''
28 exit: 0
Mapping Phase finished for machine 28
Start Mapping Phase for machine 29
29 out: 'b'''
29 err: 'b'''
29 exit: 0
Mapping Phase finished for machine 29

------- MAPPING PHASE FINISHED-------

MAPPING --- 0.39961981773376465 seconds ---
```

```
Shuffling Phase finished for machine 29

------- SHUFFLING PHASE FINISHED -------

SHUFFLING --- 81.14480566978455 seconds ---
```

```
Reducing Phase finished for machine 27
Start Reducing Phase for machine 28
28 out: 'b'''
28 err: 'b'''
28 exit: 0
Reducing Phase finished for machine 28
Start Reducing Phase for machine 29
29 out: 'b'''
29 err: 'b'''
29 exit: 0
Reducing Phase finished for machine 29

------- REDUCING PHASE FINISHED -------

REDUCING --- 0.5054361820220947 seconds ---

Total execution time --- 82.05000615119934 seconds ---
```

Conclusion

Avec le passage à 30 machines, j'ai réussi à faire baisser le temps d'exécution la phase de ***shuffling***. Par exemple sur le fichier **deontologie_police_nationale.txt** le temps d'exécution global du programme a été diminué de plus de moitié (69.05 s avec 3 machines contre 28.26 s avec 30 machines). On s'aperçoit toutefois que le temps d'exécution des phases de mapping et de ***reducing*** reste inchangé. Cela s'explique par le fait que les machines ont moins d'opérations de lecture et d'écriture, de copie et de calcul de hash. En effet, la charge de travail pour les opérations engagés durant la phase de ***shuffling*** a été lissé sur 30 machines plutôt que de centraliser toutes ces opérations sur 3 machines comme cela été le cas avant. En décentralisant les opérations de la phase de ***shuffling*** sur les 30 machines permet de mieux faire face au problème du « goulot d'étranglement ».

Cependant on remarque que la phase de ***shuffling*** reste encore la phase la plus chronophage. Pour optimiser le temps d'exécution durant cette phase, je dois diminuer la répétition des opérations de lecture, d'écriture et de copie de fichiers.

4.1.7 Implémentation de la 2ème itération de MapReduce

Pour limiter le nombre de répétition des opérations de lecture, écriture et copie de fichiers durant la phase de shuffling j'ai décidé de revoir entièrement le fonctionnement algorithmique de mon programme. Originellement durant la phase de ***shuffling*** de multiples fichiers sont générées en fonction du hash associé à chaque paire clé/valeur. L'idée est donc de transformer les fichiers textes générés au format ***.txt*** en fichier au format ***.csv.*** Cela permettra alors de diminuer le nombre de fichiers crée durant la phase de ***shuffling***. En procédant de cette manière il sera alors nécessaire d'envoyer un seul fichier ***csv*** contenant toutes les différentes paires de clé/valeur à la

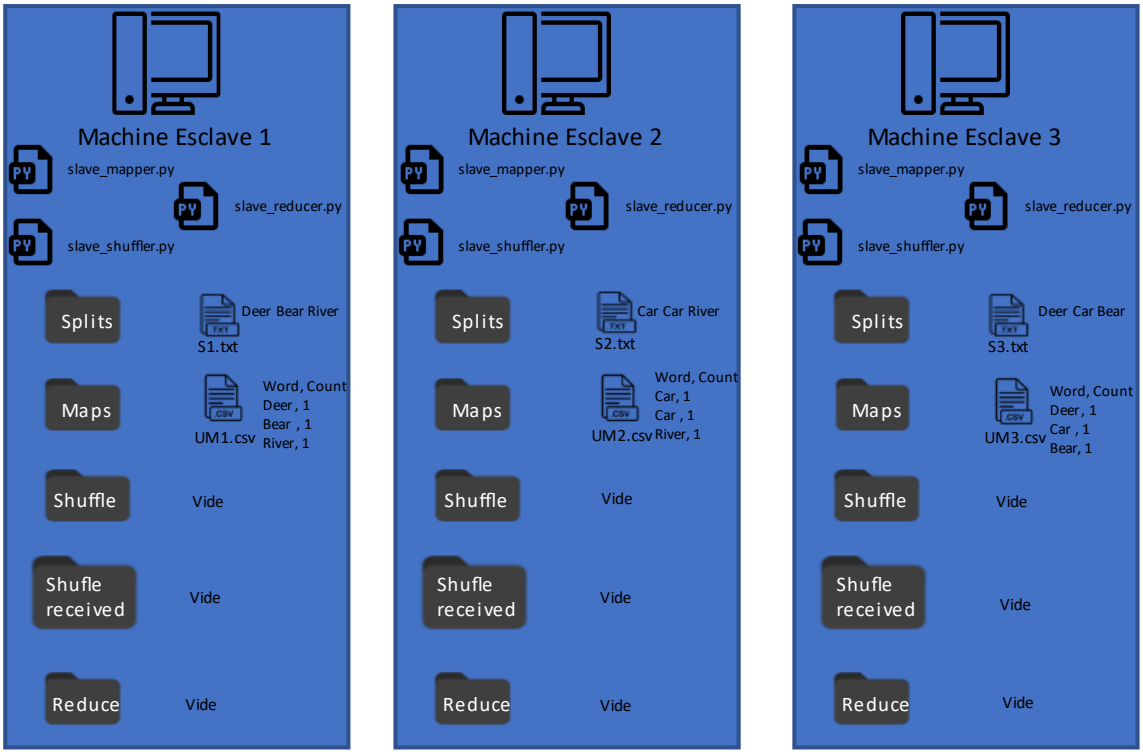
machine en question. Cependant le changement de format de fichier impose également la modification du code pour les phases de ***mapping***, ***shuffling*** et ***reducing***.

Il convient alors de passer en revue ces phases afin d'expliquer en détail les modifications que j'ai mises en œuvre.

A noter, que les phases de déploiement/initialisation de l'environnement, de ***splitting***, et ***d'assembling*** reste inchangé.

4.1.7.1 Phase de Mapping

La phase de Mapping change très peu par rapport à la première itération de mon programme MapReduce. La différence situe au niveau du format du fichier généré par la phase de Mapping. Ici, la phase de mapping génère un fichier ***.csv*** contenant dont le header est ***{Word, Count}.*** De cette façon, la phase ***shuffling*** opèrera directement sur ce fichier ***.csv***.



<u>Phase de reducing avec le fichier « slave_mapper_enhanced.py »</u>

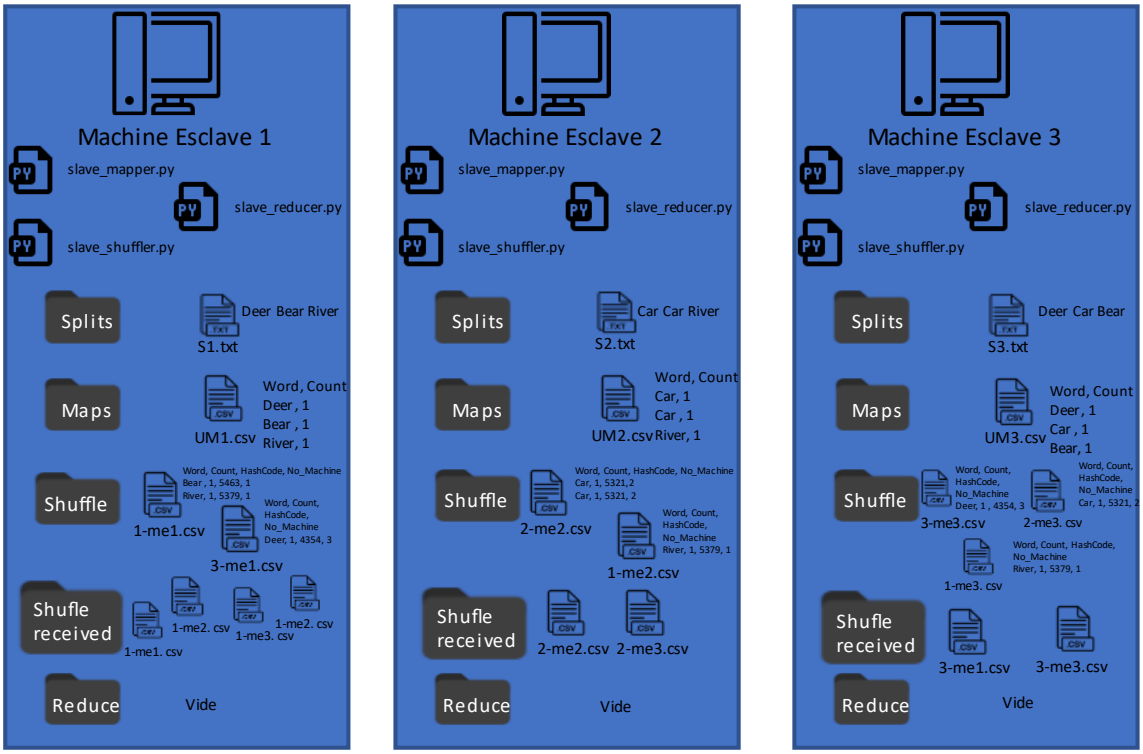
4.1.7.2 Phase de Shuffling

Comme évoqué avant j'ai revu complètement la phase de shuffling de manière à limiter le nombre d'opération durant cette phase. Pour ce faire le programme récupère le fichier ***csv*** généré par la phase de ***mapping***. Ensuite comme pour la première itération du programme MapReduce, deux nouvelles colonnes sont ajoutées à ce fichier.

- Une colonne qui contient le hash pour chaque paire clé/valeur
- Une colonne qui contient le numéro de la machine en fonction du hash qui a été calculé pour chaque paire clé/valeur.

Pour réaliser ces traitements et notamment calculé le code hash j'ai utilisé la fonction ***apply()*** qui permet d'optimiser les traitements sur une colonne entière d'un Dataframe.

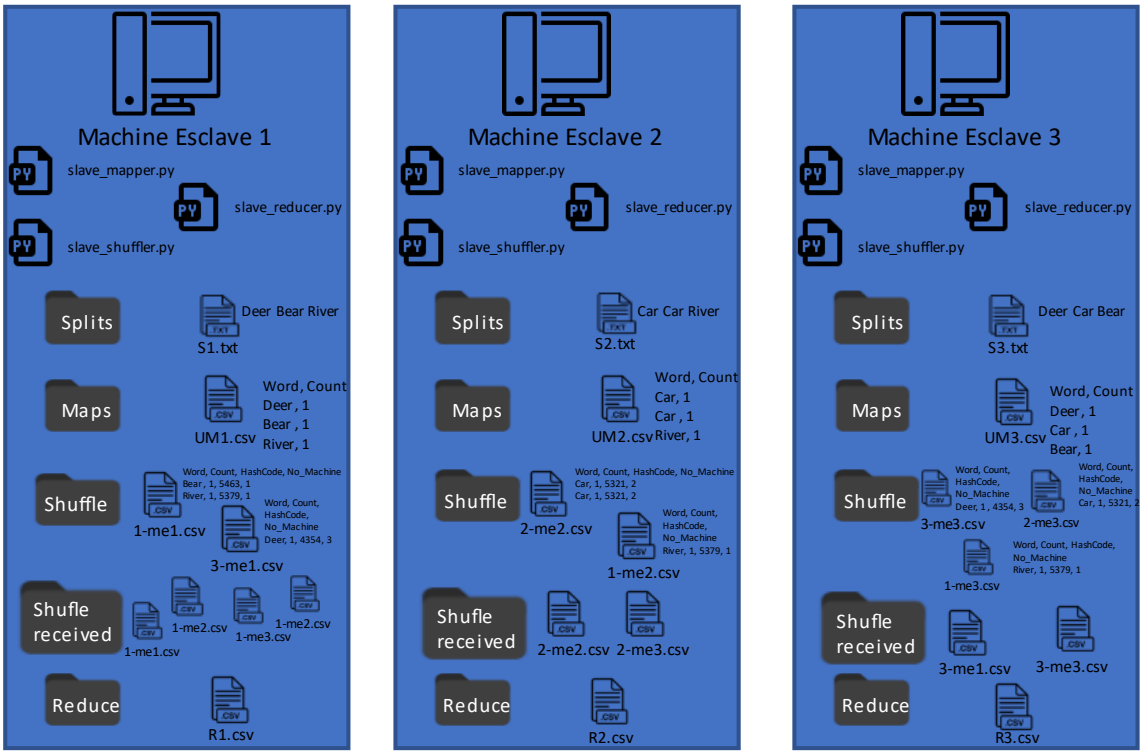
Ensuite le programme sélectionne les lignes du Dataframe avec le même numéro de machine. La fonction optimisée ***groupby()*** de python permet de réaliser efficacement cette opération. Ainsi, un Dataframe est crée pour chacune des machines esclaves. Cela signifie donc qu'un seul fichier sera envoyé à la machine esclave en question. Ceci n'était pas le cas dans la 1ère itération du programme car pour chaque paire clé/valeur unique était créé un fichier spécifique dans le dossier ***shuffle***. De cette façon lors de la phase de ***reducing***, le programme aura moins de fichier à lire.



Phase de reducing avec le fichier « slave_shuffler_enhanced.py »

4.1.7.3 Phase de Reducing

A travers la phase de ***reducing***, le programme concatène l'ensemble des fichiers .csv contenu dans le dossier ***shuffledreceived*** au sein d'un unique Dataframe. Ensuite le programme, applique sur ce dernier une fonction d'agrégation sur la colonne « ***Word*** » de manière à sommer les valeurs de chacun de mot identique contenu dans la colonne « ***Count*** ». Cette étape est réalisée sur chacune des machines exclaves.



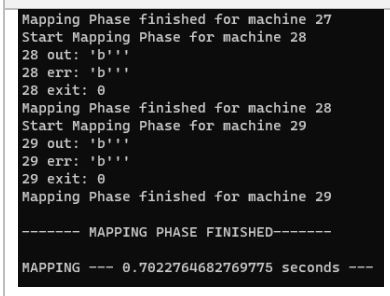
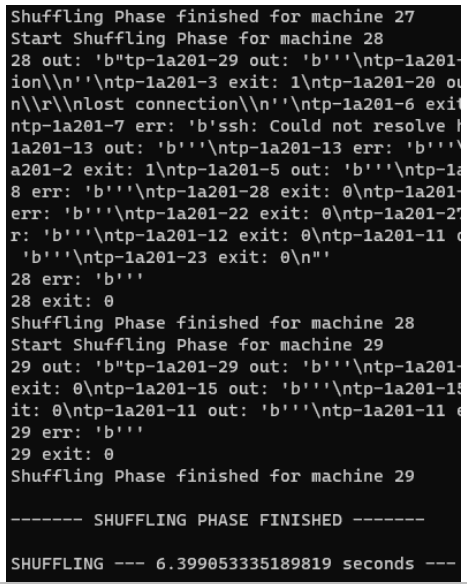
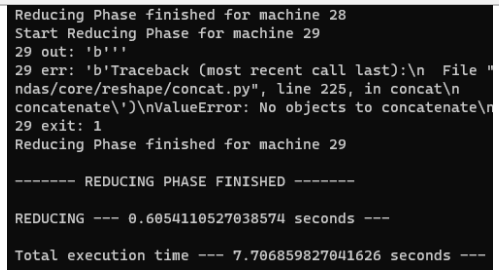
Phase de reducing avec le fichier « slave_reducer_enhanced.py »

4.1.7.3 Phase d'assembling

En ce qui concerne la phase, ***d'assembling***, le fichier généré lors de la phase de ***reducing*** par chacune des machines esclaves est rapatrié sur la machine maitre. Pour générer le fichier final, la machine maitre concatène simplement l'ensemble de ces fichiers.

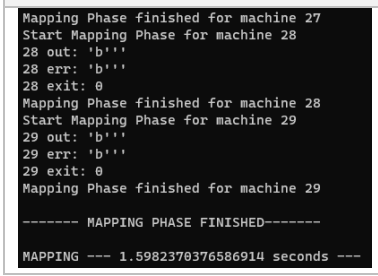
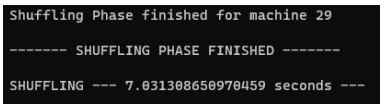
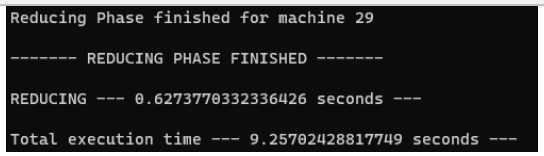
4.1.8 Application de la 2ère version MapReduce sur différents fichiers avec 30 machines

Fichier : deontologie_police_nationale.txt (8 ko)

Condition du test : 30 machines et time = 400 s			
Temps Mapping	**Temps Shuffling**	**Temps Reducing**	**Temps Total**
0.40 s (**1ère version** avec 30 machines)	**27.5 s** (**1ère version** avec 30 machines)	**0.41 s** (**1ère version** Avec 30 machines)	**28.26 s** (Avec 30 machines)
0.70 s	**6.40 s**	**0.61s**	**7,71 s**
			

Un extrait du résultat généré pour le fichier **deontologie_police_nationale.txt** est disponible en ANNEXE 4.3

Fichier : domaine_public_fluvial.txt (71 ko)

Condition du test : 30 machines et time = 400 s			
Temps Mapping	**Temps Shuffling**	**Temps Reducing**	**Temps Total**
0.40 s (**1ère version** avec 30 machines)	**81.14 s** (**1ère version** avec 30 machines)	**0.63 s** (**1ère version** avec 30 machines)	**81.05 s** (Avec 30 machines)
1.60 s	**7.03 s**	**0.50 s**	**9.26 s**
			

Un extrait du résultat généré pour le fichier **domaine_public_fluvial.txt** est disponible en ANNEXE 4.4

Conclusion

On remarque globalement que le temps de la phase de Mapping de cette 2ème version de mon programme prend un peu plus de temps par rapport à la 1ère version même si cela reste toutefois négligeable. Cela s'explique par le fait qu'à chaque mot extrait du texte on crée une nouvelle ligne dans le DataFrame. A l'inverse grâce au passage en DataFrame, le temps d'exécution de la phase ***shuffling*** à bien diminué. A titre de comparaison, sur le fichier **domaine_public_fluvial.txt** nous avons gagné environ 74 secondes entre la 1ère version du programme MapReduce et la 2nd version.

En opérant directement sur des tableaux et en regroupant les lignes du tableau en fonction du numéro de machine, j'ai réussi à limiter le nombre de création de fichiers ainsi que le nombre de fichiers qui doivent être copiés sur les machines esclaves. En effet, chaque machine esclave reçoit un unique fichier ***csv*** d'une autre machines esclave. Par conséquent, le dossier ***shuffledreceived*** de chaque machine esclave comportera au maximum ***30*** fichiers. Enfin le langage python à l'avantage d'être optimisé pour effectuer des transformations de données sur des tableaux. Cela permet ainsi de diminuer le temps d'exécution des opération usuelles comme, la lecture, l'écriture, la concaténation de plusieurs fichiers et l'application de fonctions d'agrégation sur un tableau de données. Par exemple la fonction python ***read_csv()*** permet de lire rapidement un grand nombre de lignes contenu dans un fichier ***csv***.

5 Conclusion

Pour le projet de l'enseignement INF 727, j'ai pu implémenter une 1ère version simple du concept MapReduce en tirant profit de la puissance de calcul de plusieurs machines. J'ai pu m'apercevoir que le nombre de machines utilisées à une influence sur le temps d'exécution global du programme. Plus le nombre de machines utilisé est important plus le temps d'exécution du programme MapReduce diminue. Par ailleurs pour ce projet j'ai également

pu proposer une version améliorée de la 1ère version de mon programme MapReduce. Cette version améliorée diminue le nombre des opérations de lectures, d'écriture et de copie de fichiers rendu possible grâce au passage au format .csv.

Finalement, à travers cet enseignement, j'ai pu me familiariser avec le fonctionnement de MapReduce et plus généralement avec le concept des systèmes répartis pour le calcul distribué. Enfin, j'ai pu apercevoir m'apercevoir des limites de MapReduce dont l'une concerne le fait que MapReduce est dit (I/O intensive) car ce dernier à recourt à beaucoup d'opérations d'écriture et de lecture.

4. ANNEXE

4.1 Un extrait du fichier WC_result.txt pour le fichier *deontologie_police_nationale.txt (1ère version du programme)*

```
inhumain 1
fonctions. 1
aucun 2
pour 5
but 1
lui 2
sa 5
cas 5
informés 1
II 3
si, 1
fonctionnaire 9
a 5
disciplinaire, 1
manifestement 1
cette 2
résultant 1
conformer 3
Outre 1
médical 1
sont 5
police 29
malgré 1
donné 2
l'individu 1
autorités 1
9 1
librement 1
voie 1
Placé 1
l'ordre 5
peut 2
d'exécuter 1
atteindre. 1
armes, 1
chambre 1
l'exercice 1
place, 1
n'entreprend 1
lorsqu'ils 1
DE 12
eux 1
s'exprimer 1
```

4.2 Un extrait du fichier WC_result.txt pour le fichier*domaine_public_fluvial.txt (1er version du programme)*

```
construction. 4
fluviale, 3
principaux 2
greffiers 2
frais, 2
général 2
5° 7
nullité 2
lettre 8
seing 6
l'égard 4
navigables. 4
l'adjudication 8
requérir 8
l'Etat 2
autorisation 2
reçu 6
pour 69
poursuites 6
Paris. 2
apparente 2
but 2
prix.L'acte 2
noms, 17
sa 10
régime 1
entraîne, 2
cas 26
délivre 4
Livre 15
privilégiés. 2
;L'élection 2
différents.L'acte 2
appel, 2
intéressées, 2
l'expiration 8
354 2
demande 6
L'immatriculation 2
d'enregistrement. 2
II 17
locale. 6
l'enquête, 2
si, 2
domaine_public_fluviald 1
réquisitions 1
```

4.3 Un extrait du fichier WC_result.txt pour le fichier *domaine_public_fluvial.txt (2ème version du programme)*

```
8,1
article,1
ce,3
donné,2
définis,1
envers,2
fonctionnelle,1
fonctions,2
ils,2
loyal,1
nécessaires,1
victimes,1
:,9
DROITS,3
Ier,3
Il,4
POLICE,6
attache,1
compétente.,1
l'ordre,5
le,19
librement,1
posées,1
refus,1
strictement,1
subordonnés,1
"tenu,",1
"voie,",1
CONTROLE,3
assistance,1
"danger,",1
dans,6
expose,1
intermédiaires,
intègre,1
malgré,1
nationalité,1
personnes,3
précis,1
public,2
qu'un,1
sans,2
échelons,1
11,1
13,1
GENERAUX,3
```

4.4 Un extrait du fichier WC_result.txt pour le fichier *deontologie_police_nationale.txt (2ème version du programme)*

```
137.,2
"160,",2
171,2
"81,",2
Est,4
Ils,1
V,7
article,4
assurée,1
autorisées,6
avisés,2
bis,2
ce,18
"certificat,",2
citations,4
commerce.,6
cour.,2
criées,2
"d'hypothèque,",2
d'office,2
donné,8
dressé,6
exigible,2
facultés,2
fonctions,2
ils,4
inséré,2
l'exécution.Le,2
lesquelles,2
logement,3
"mécanique,",2
notification,2
on,2
pourront,4
poursuit,4
proche,2
procédé,10
rapport,2
répertoire,4
spéciaux.,2
traité,2
vigueur;,1
122,2
:,50
Ier,14
Il,33
```

6 Bibliographie

- Juvénal CHOKOGOUE, « Devenez opérationnel dans le monde du Big Data ». Editions ENI, 2017