MS Big Data

INF727

Rapport du projet SHAVADOOP

Encadrés par:
Rémi Sharrock

Réalisé par :
Sami Bargaoui
Mohammed Benseddik

Contents

1 Contexte du projet

Le but du projet est d'implémenter une fonction de Wordcount en MapReduce, d'une manière analogue à la fonction implémentée sur Hadoop. La fonction Wordcount comptera le nombre d'occurences de chaque mot présent dans un fichier texte donné en entrée, l'approche MapReduce pour le Wordcount répartira les opérations (Splitting, Mapping, Reducing, Shuffling, ...) de manière distribuée sur un ensemble de machines connectées.

L'approche Wordcount - MapReduce consiste à cet ensemble de traitements :

- **Une phase de Mapping :** Chaque machine appelle la fonction Map() sur le jeu de données qu'elle a reçu. La fonction Map() calculera le nombre d'occurrences de chaque mot présent dans les données.
- **Une phase de Shuffling :** Chaque machine redistribue les données à partir du résultat de l'étape de Mapping. On peut par exemple rassembler un ensemble de mots ou de clés sur une machine en particulier, de façon à répartir l'ensemble des clés sur toutes les machines.
- **Une phase de Reducing :** Chaque machine ayant reçu un ensemble de clés chacune ayant une certaine valeur, appliquer la fonction Reduce() consiste à rassembler toutes les clés et à faire la somme de toutes les valeurs associées, on aura donc un ensemble de clés (les mots présents dans le fichier en entrée), auxquelles on associe des valeurs (le nombre d'occurrences de chaque mot dans le fichier), d'où le principe du MapReduce.

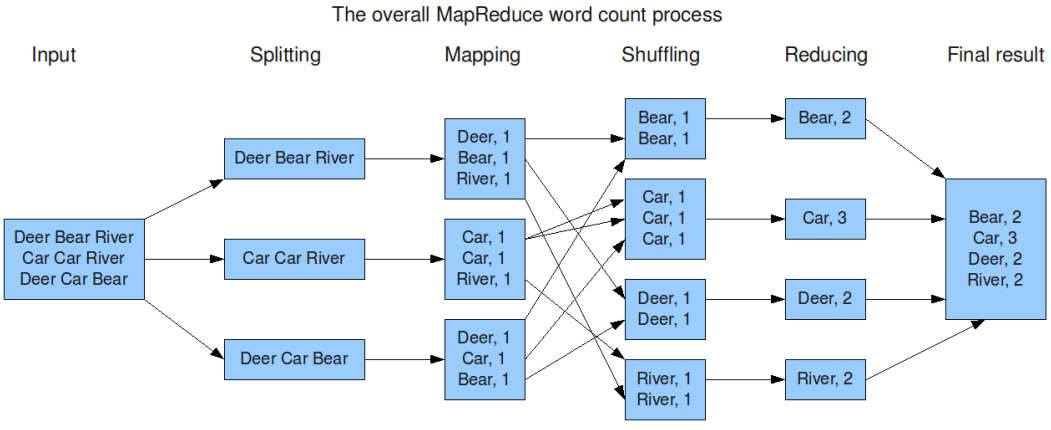

Figure 1: Schema MapReduce

Cette opération fait intervenir un master et plusieurs slaves ou workers. Le rôle du master est de surveiller toutes les opérations et lancer sur les workers les opérations requises, tandis que le rôle des workers est d'exécuter les opérations demandées par le master, toute la communication entre les différentes parties se fera en SSH.

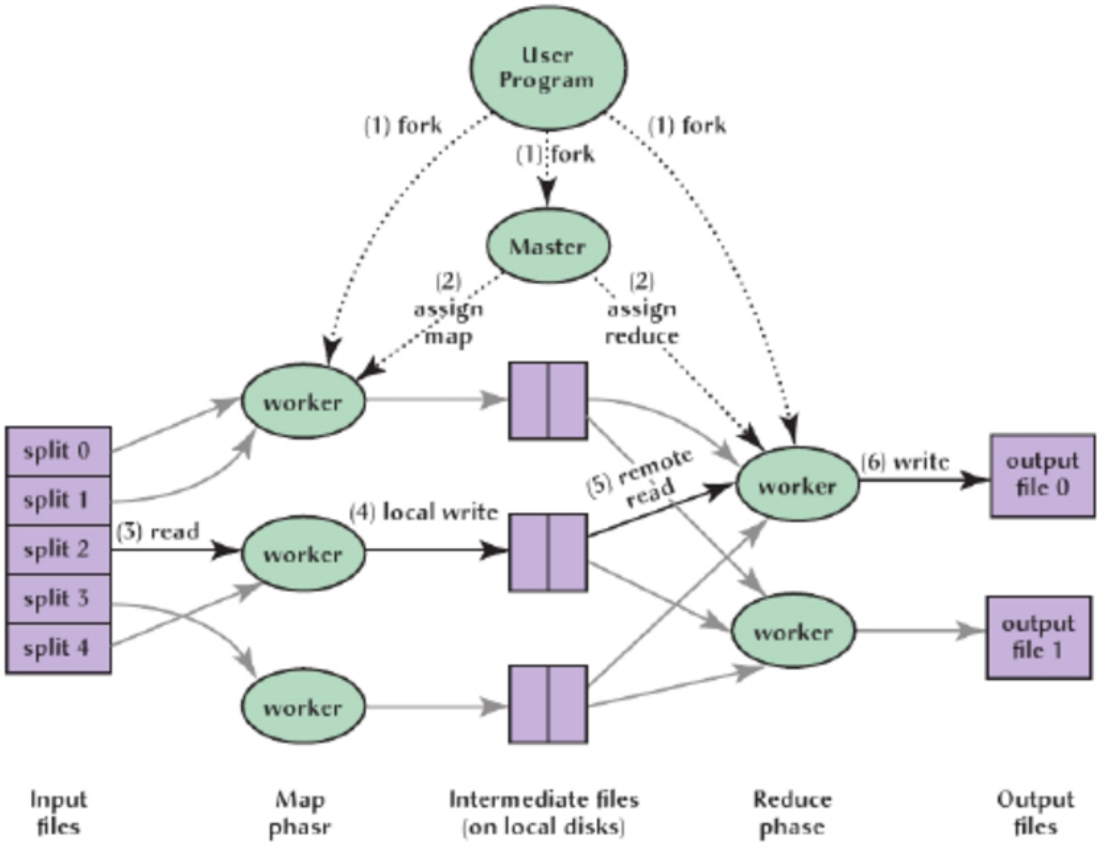

Figure 2: Schema MapReduce

2 Organisation du code

Le programme est organisé en deux principales parties :

2.1 Les classes

2.1.1 Le Master

- Les traitements du Master :
  - CheckPcsThread : Contient les fonctions liées à la connexion du master aux différentes machines reliées au réseau.
  - Master : Contient les procédures et fonctions liées au traitement du master (Création des dictionnaires, Splitting, Merging, ...)
  - ShavadoopThread : gère l'exécution des threads sur les différents slaves, fonctionne en deux modes : SxUMx (Mapping) et UmxSmx (Reducing).
  - ShavadoopUtils : Contient des sous-fonctions utilisées par les méthodes invoquées dans le master.

2.1.2 Le Slave

- Les traitements du Slave :
  - Slave : exécute les fonctions du mapping et reducing sur les différentes machines.

2.2 Les interfaces

Les deux interfaces implémentées (ConstantsSlave et ConstantsMaster) permettent d'enregistrer les paramètres et les variables globales du programme.

3 Fonctionnement du code

3.1 Fonctionnement du master

Le master fait appel principalement à trois grandes fonctions, qui font appel à des sous fonctions. D'abord le master opère le split sur le fichier donné en entré et crée les fichiers Sx à partir de la procédure splitSxBloc, qui reçoit en entrée le fichier à splitter et un nombre de lignes choisi par l'utilisateur : si par exemple le fichier contient 2000 lignes et que l'utilisateur donne en paramètre une division chaque les

100 lignes, on aura 20 fichiers Sx qui correspondent à 20 partitions du fichier, le résultat renvoyé est un dictionnaire des fichiers Sx.

Puis on opère le MapReduce sur les slaves à travers la fonction runOnSlaves : on recoit le dictionnaire de fichiers de la phase précédente, on établit une liste de machines sur lesquelles lancer notre traitement, on opère la phase de Mapping à travers la méthode splitOnUmx qui retourne un dictionnaire, on passe à la phase de shuffling et on termine par lancer le Reducing sur le dictionnaire obtenu en faisant appel à la méthode splitOnSmXAndRmx. On sauvegarde après le fichier obtenu où on retrouvera les mots du fichier et leur occurence.

La classe Master fait appel à des méthodes intermédiaires qu'on retrouve sur dans ShavadoopUtils. Cette classe de méthodes regroupe les traitements utiles au lancement du programme comme la suppression des fichiers crées lors de lancements du code antérieurs, des procédures d'affichage, les fonctions qui interviennent dans le split du fichier en plusieurs sous fichiers, etc...

La classe ShavadoopThread est la classe qui lance et gère les traitements sur les différents slaves et fonctionne en deux modes distincts : mapping ou reducing. Elle contient aussi des méthodes qui gèrent l'exécution de tâches sur des machines distantes.

3.2 Fonctionnement du slave

La classe Slave contient des méthodes d'exécution des tâches lancées par le main. Parmi les méthodes implémentées dans le Slave, on retrouve des traitements sur les fichiers : éliminations de caractères spéciaux, éliminations des mots de liaisons et des conjonctions, etc.

On retrouve aussi les méthodes liées à la phase de Shuffling, Reducing et gestion des traitements sur les fichiers intermédiaires et dictionnaires (Um, Sm, Rm). La méthode main quant à elle est la méthode où on éxécute les méthodes citées audessus sur les différentes machines de la liste pré-établie.

4 Lancer le code

Le code ne fait pas appel à des librairies extérieures. Pour lancer le code, il importer les fichiers sources dans Eclipse. Il faut ensuite changer le répertoire qui contient les fichiers en entrée dans le Slave (SlaveShavadoop) et paramétrer le bon chemin pour les fichiers nécessaires. Le lancement se fait à partir du Master (MasterShavadoop).

En sortie, on verra l'exécution des différentes étapes, ainsi que les temps impartis pour chaque étape du traitement.

Le lancement se fait comme suit :

- Créer des clés SSH liées au compte Telecom utilisé.
- Importer les deux projets MasterShavadoop et SlaveShavadoop sous Eclipse.
- Aller aux interfaces constantSlave et constantMaster :
  - Changer le directory et le remplacer par le chemin qui mène aux fichiers qu'on va traiter et les fichiers utiles (liste des pcs.txt, mots ignorés.txt). Le dossier s'appelle shavadoopFiles dans notre cas. On peut aussi modifier les pramètres d'exécution (Timeouts, Pas du split des fichiers), mais les paramètres présents par défauts on été testés et fournissent des résultats concluants.
  - Changer le userName par le login au compte associé (Ex : @user)
- Exporter le slave en Runnable JAR et le renommer "SLAVESHAVADOOP.jar". Déplacer le jar dans le même directory paramétré à l'étape précédente.
- Mettre en argument le fichier à traiter du Master.java (Sous Eclipse, aller à Run Configurations - Arguments - Program Arguments). Par exemple, fournir Input.txt.
- Attendre la fin du traitement et récupérer le fichier ReducedFile.txt créé dans le directory.

5 Problèmes confrontés et Optimisations

Lors de la réalisation du projet, on a été confrontés à quelques problèmes. Un exemple a été les délais de connexions SSH, les timeouts et l'éventualité de threads qui ne répondaient plus causait qu'on perdait des données en les envoyant à des slaves qui ne répondaient plus, donc on aboutissait à un fichier après traitement où il nous manquait plusieurs clés/valeurs. Pour remédier à ce problème, on a pensé à relancer les threads morts après le premier lancement de threads, et en relançant à chaque fois, on s'assure de ne pas avoir perdu de données en cours de traitement, et que tous les fichiers soient répartis et traités. Aussi, une recherche de machines sur le réseau est effectuée au début du processus puis stockée sous forme de liste. Cette recherche est unique, il n'y a donc pas de vérification par la suite, ce qui

pose le problème des threads perdus et des longs délais de timeout qui ralentissent l'ensemble de l'exécution.

Parmi les autres problèmes, on a été confrontés à la réplication des clés Umx lors du Mapping des fichiers. On a utilisé des HashMap et HashSet qui gèrent automatiquement la suppression des duplicata de clés. L'un des autres enjeux principaux du projet a été d'écrire un code générique qui peut s'adapter à nos besoins et fonctionner sous plusieurs modes, ce qui a été le cas des deux modes (SxUMx - Mapping et UmxSmx - Reducing).

Lors de l'exécution du Slave, on a recours à un fichier donné en entrée (MotsIgnores.txt) qui fait guise de fichiers de délimiteurs qui contient les caractères à ignorer.

Enfin, l'un des enjeux majeurs a été de réduire les temps d'exécution. L'une des méthodes a été de sous-diviser notre fichier en entrée en plusieurs sous-fichiers traités que l'on attribue aux machines à travers la méthode splitSxBloc. Une division par taille mémoire a été aussi envisagée mais fournissait des résultats similaires, on a donc retenu la sous-division en lignes. L'autre solution a été aussi de réduire au minimum les connexions SSH et de gérer les threads qui ne répondaient pas : c'est dans cette optique qu'on a crée au lancement des threads une liste des threads qui s'actualise avec les machines qui ne répondent pas pour renvoyer les fichiers Sx dessus, et ainsi on s'assure de ne pas perdre des données quand certaines machines ne répondent plus, et ainsi de suite, jusqu'à ce que chaque Sx soit traité sur des machines qui renvoient des résultats.

6 Exemples d'exécution

Voici les temps d'exécution obtenus en moyenne sur différents fichiers fournis en entrée, avec la phase de recherche de machines actives :

Fichier	Reducing(Secondes)	Temps total(Secondes)
Mayotte	11	27
Fluvial	77	93

On fournit en annexe quelques Screenshots des phases d'exécution.

7 Github, Javadoc

Le projet est disponible sous Github : https://github.com/BenseddikM/Shavadoop ou https://github.com/Sbargaoui/Shavadoop . Dans le projet aussi, une Javadoc a été créée pour toutes les classes et les méthodes implémentées.

La Javadoc détaillée se trouve sur ce siteweb créée pour le projet sous Github : https://sbargaoui.github.io/Shavadoop/docs/.

A Screenshots du programme

Voici quelques screenshots de l'exécution :

Figure 3: Debut de l'exécution

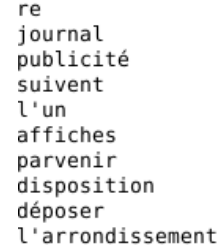

Figure 4: Mapping

```
On lance Slave sur le pc : c129-10 pour avoir le reducedFile : /cal/homes/mbenseddik/shavadoopFiles/Sm889.txt
On lance Slave sur le pc : c129-11 pour avoir le reducedFile : /cal/homes/mbenseddik/shavadoopFiles/Sm890.txt
On lance Slave sur le pc : c129-12 pour avoir le reducedFile : /cal/homes/mbenseddik/shavadoopFiles/Sm891.txt
On lance Slave sur le pc : c129-14 pour avoir le reducedFile : /cal/homes/mbenseddik/shavadoopFiles/Sm892.txt
```

Figure 5: Reducing

```
même 8

intéressée 2

seulement 7



We re-run for the word : on the pc :c129-06
We re-run for the word :profession on the pc :c129-14
We re-run for the word :intéressant on the pc :c129-15
We re-run for the word :d'immatriculation on the pc :c129-18
We re-run for the word :prêter on the pc :c129-19
We re-run for the word :marchandises on the pc :c129-20
We re-run for the word :délivrer on the pc :c129-26
We re-run for the word :chapitre on the pc :c129-28
We re-run for the word :vingt-quatre on the pc :c129-29
We re-run for the word :patentés on the pc :c129-32
We re-run for the word :supérieur on the pc :c129-35
We re-run for the word :doivent on the pc :c128-07
We re-run for the word :seulement on the pc :c128-08
We re-run for the word :9 on the pc :c128-19
We re-run for the word :rapportent on the pc :c128-24
We re-run for the word :demander on the pc :c128-27
We re-run for the word :intéressée on the pc :c129-22
We re-run for the word :partielle on the pc :c128-13
We re-run for the word :l'officier on the pc :c129-30
We re-run for the word :l'acte on the pc :c128-35
We re-run for the word :;2° on the pc :c129-23
We re-run for the word :lorsqu'ils on the pc :c129-21
We re-run for the word :l'objet on the pc :c128-16
chapitre 4


délivrer 1

prêter 2

supérieur 1
```

Figure 6: Relance des threads morts sur les slaves

```
Smx and Rmx parts Done!
Duration of SPLIT ON SMX AND RMX : 77seconds !

                        ********************************************************************************
                        ********************************************************************************


All Done!
Total Duration : 93seconds !
```

Figure 7: Sortie finale du programme

B Diagrammes UML

On expose dans cette section les diagrammes UML générés par les classes implémentées.

B.1 Projet Slave

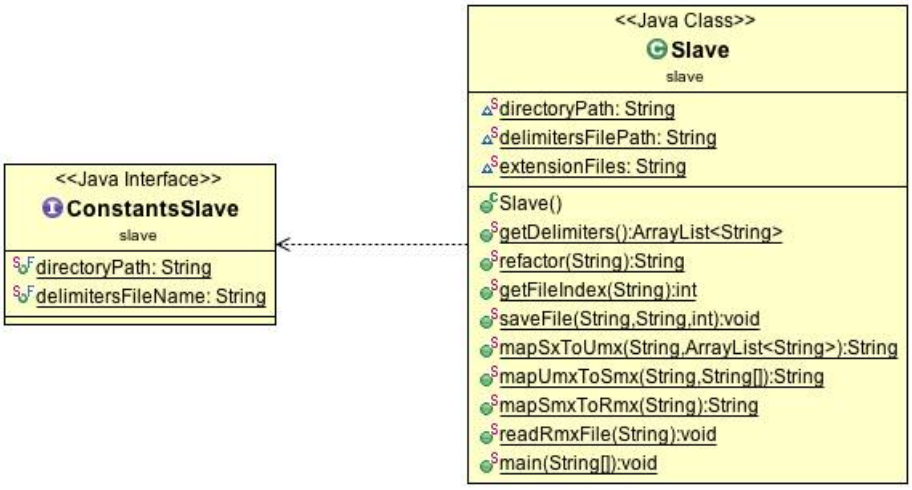

Figure 8: Diagramme du Slave

B.2 Projet Master

Figure 9: Diagramme du Master