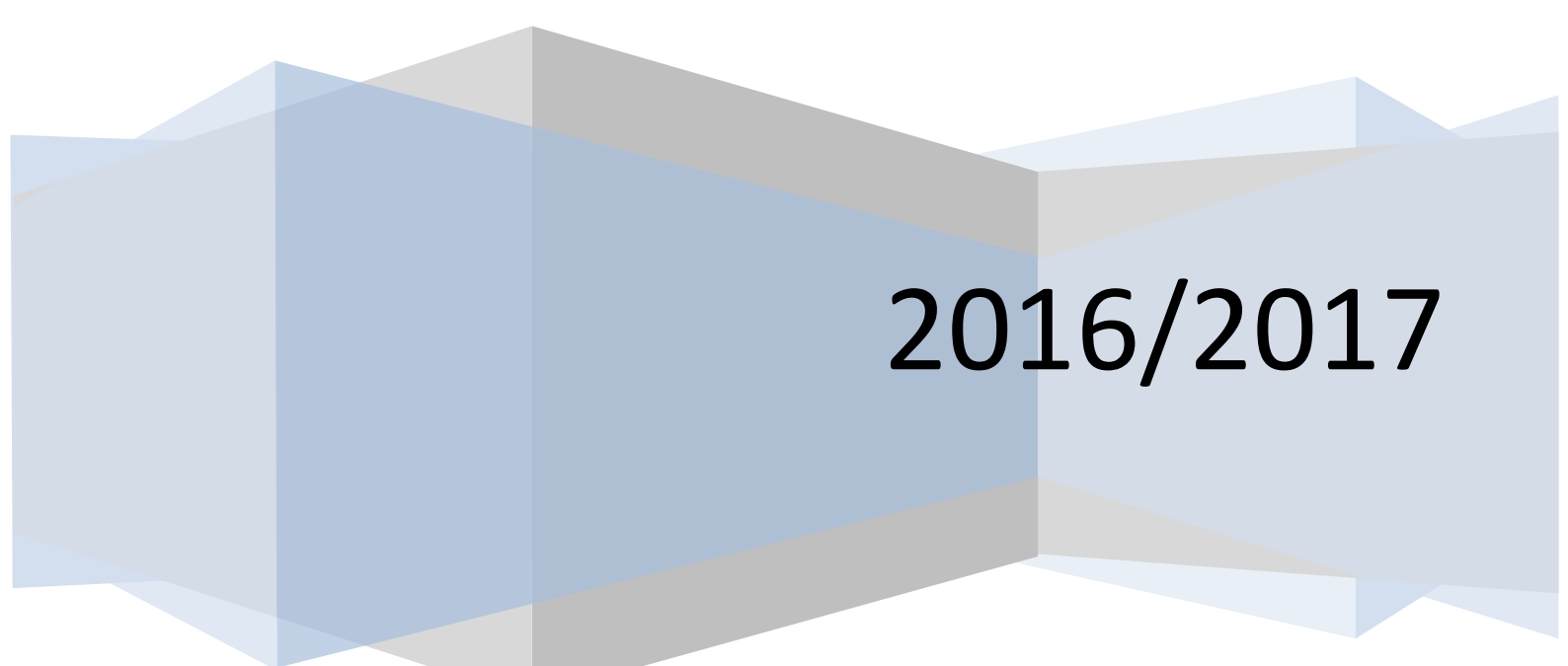


Programmation large échelle

TP3 : Map Reduce

Naji ZAOUI

Yahya BACHIR



2016/2017

Dans cette partie on a découvert le fonctionnement du Map Reduce en écrivant un programme qui nous permettra de calculé le nombre de population totale + le nombre de villes du monde + le nombre de ville dont la population a été renseignée. Tout cela en prenant le fichier « worldcitiespop.txt » en entrée, et comme sortie un fichier « part-r-00000 » qui contient que les villes dont la population a été renseignée.

```
/*
 * The Mapper gets as input the file worldcitiespop.txt that contains all the
 * cities of the world (with their Country, City Name, Accent City, Region, Population,
 * Latitude, Longitude) and sends the result to the reducer which in our case does
 * nothing but write the result it has been given from the mapper.
 *
 *
 * The Mapper does all the work:
 * - Count all the Cities given in the file
 * - removing the lines that doesn't contain a designated number of population
 * (except for the first line that contains the names of the columns)
 * - Count the cities for which we know the number of population
 * - Count the total of all the given population
 */

//the counter used to calculate the number of cities given, cities with an informed
//number of population, the sum of the population respectively nb_cities,nb_pop,
//total_pop
```

Exercice 1 :

Le but de cet exercice est d'écrire un code qui nous permettra de repérer les villes dont on ne connaît pas le nombre de populations, pour cela on a utilisé la méthode split (",") sur pour notre entrée « value », qui est une ligne de notre fichier worldcitiespop.txt, et comme résultat on avait un tableau de String : Data []. La cinquième colonne de notre tableau représentera la population.

Finalement, il suffisait qu'on vérifie si la 5^{ème} case (qui contient le nombre de population) est vide ou non, si c'est le cas on ignore cette ligne sans rien faire, sinon on appelle la méthode write () de « context » on lui donnant la toute la ligne comme argument.

La méthode context.write () nous sert pour écrire sur le OutPut du Mapper ou bien du Reducer, et vu que dans notre cas tout le calcul est fait par le Mapper, donc lors de l'appelle de la methode write() par le mapper cela va écrire sur l'output du Mapper = l'input du Reducer , et le Reducer ne fera que transférer le résultat du Mapper sur le fichier de sortie.

Exercice 2 :

L'objectif de cette partie et de pouvoir calculer :

- Le nombre de villes dans notre fichier d'entrée (nb_cities).
- Le nombre de villes dont la population a été renseignée (nb_pop).
- Le nombre d'habitants de toutes les villes (total_pop).

Pour cela on récupère des conteurs à partir de « context » grâce a la méthode `getCounter()`, dans laquelle on donne deux arguments : Le groupe de compteurs (WCD), et le nom du compteur (`nb_cities / nb_pop / total_pop`), et on met le retour de la méthode `getCounter` dans des objets Counter différent : `Valid_cities` pour 'nb_cities' / `Cities_With_Pop` pour 'nb_pop' / `Total_pop` pour 'total pop'.

- Pour le calcul de nombre de villes dans notre fichier d'entrée : (`Valid_cities`)
Il suffit d'appeler la méthode `increment(1)` de la classe Counter à chaque fois que le Mapper est appelé, autrement dit pour chaque ligne du fichier « `worldcitiespop.txt` » on incrémente de 1.
- Pour le calcul de villes dont la population a été renseignée : (`Cities_With_Pop`)
Pour cela on appelle la méthode `increment(1)`, qui incrémentera de 1, pour chaque ligne pour laquelle on a renseignée la 5^{ème} case de `Data []` plus le fait que ça ne doit pas être la première ligne du fichier vu qu'elle contient que les noms des colonnes.
- Pour le calcul du nombre d'habitants de toutes les villes: (`Total_Pop`)
De même pour le point précédant sauf qu'ici au lieu d'incrémenter de 1 on va incrémenter par le nombre de population de cette ville, cela dit la 5^{ème} case de `Data []` qu'on convertie en entier.

```
String Data[]=value.toString().split(",");
if (!Data[4].isEmpty())
{

    //The goal of this test is to make the mapper ignore the first line in the
    //file since it only contains the name of the columns

    if(!Data[4].matches("Population"))
    {

        //The entries that reach this point are all the lines, starting
        //from the second one, for which we know the number of population.
        //For every city with an informed number of population => increment
        //the counter nb_pop by one, and the total_pop by the number of
        //population of that city.

        Cities_With_Pop.increment(1);
        Total_Pop.increment(Integer.parseInt(Data[4]));
    }
    context.write(value, new IntWritable(1));
}
```

N.B : On est conscient que ce qu'on a écrit au rapport est redondant vu le code a été bien expliqué par des commentaires et il y avait pas beaucoup de choses à y ajouter sur le rapport.