

UNIVERSITE DE BORDEAUX

Programmation large échelle
Rapport du TP3
Map Reduce

Par :

HOCINI Mohamed Fouad

MAURICE Bastien

Enseignant :

David Auber

15 octobre 2017

Ce TP3, nous a permis de se familiariser avec le framework Map Reduce en écrivant un programme java qui nous permettra de lire le fichier worldcitiespop.txt donné et qui enlève toutes les lignes dont la population n'est pas connue, l'utilisation des Counter pour calculer le nombre de cités valides, le nombre de population, le nombre totale de population ainsi que le nombre de villes et la population par pays et à la fin nous créer un fichier part-r-00000 qui contient le résultat attendu.

1 Exercice 1

Dans cet premier exercice, on devait enlever toutes les lignes dont la population n'est pas connue dans notre fichier worldcitiespop.txt , pour cela on a utilisé un split(",") pour notre entrée value qui est une ligne dans le fichier et comme résultat on avait un tableau de String : Data[], dans lequel la cinquième colonne représente la population.

Pour réaliser ce traitement, il suffisait de vérifier que la cinquième case (Data[4]) qui contient le nombre de population ne soit pas vide et c'est le cas on appelle la méthode write() de context on lui donnant cette ligne.

2 Exercice 2

L'objectif de cet exercice était de manipuler les compteurs avec getCounter(WCD, nameCounter), pour répondre ce qui a été demandé on a créer trois compteurs : nbCities pour le nombre de cité valide, nbPop et totalPop pour le nombre totale de population.

2.1 Calcul du nombre de villes valide

Une ville valide est une ville avec un pays, ville, région, pour répondre à ça il suffisait de vérifier que les trois premières cases (Data[0], Data[1] et Data[2]) ne soient pas vide pour incrémenter le compteur avec la méthode increment(1) de la classe Counter à chaque fois que le Mapper est appelé.

2.2 Calcul du nombre de villes avec une population renseignée

Pour ce calcul, on appelle la méthode increment(1), pour incrémenter notre Counter de 1 pour chaque ligne avec la cinquième case (Data[4]) n'est pas vide prenant en considération que la première ligne ne doit pas être compter parce qu'elle contient que les noms des colonnes.

2.3 Calcul du nombre d'habitants de toutes les villes

Pour ce calcul, on fait la même chose que le calcul précédent sauf que à chaque fois on incrémente par le nombre de population de la ville et non pas par 1, et cela se fait par la conversion de la cinquième case (Data[4]) en entier.

Dans la figure suivante, une capture des résultats des trois compteurs avec les résultats suivantes :

- Nombre de ville valide : 3173959.
- Nombre de ville avec une population renseigné : 47980.
- Nombre totale de population : 2289584999.

```
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0

WCD
nb_cities=3173959
nb_pop=47980
total_pop=2289584999
File Input Format Counters
Bytes Read=151149418
File Output Format Counters
Bytes Written=2382331
mohhocini@noree:~$
```

Figure1 : Résultat des trois compteurs

3 Exercice 3

3.1 Calcul du nombre de ville par pays

Nous allons avoir besoin d'utiliser à la fois le mapper et le reducer pour ces 2 prochaines questions.

Exemple avec un fichier d'entrée comme celui-ci :

France, Bordeaux

France, Paris

Italie, Rome

Étape mapper :

```
1 : string data[] = value.toString.split(",");
```

```
2 : context.write(data[1],1);
```

La ligne 1 va nous permettre de lire chaque ligne de notre fichier d'entrée, de séparer chaque champs, initialement délimité par un char spécial ",", et de les placer dans un tableau.

La ligne 2 va nous permettre d'inscrire en sortie du mapper et d'associer un nom de pays avec une valeur de 1.

Fichier à la sortie du mappeur sous forme de (Key,value) :

France, 1

France, 1

Italie, 1

Après cette étape, le fichier sera automatiquement triée de la sorte (appelée phase de shuffle et sort) :

France[1,1]

Italie[1]

Étape reducer :

```
1 : int compteur = 0;
```

```
2 : foreach v in values :
```

```
3 : compteur = compteur + v;
```

```
4 : context.write(key,compteur);
```

La ligne 1 nous permet d'initialiser un compteur. Il contiendra la valeur du nombre de ville par pays.

La ligne 2 nous permet de boucler pays par pays du fichier qui sort de la phase de shuffle et sort. La ligne 3 ajoute l'ensemble des valeurs de chaque pays. Sachant que la valeur vaut 1 pour représenter une ville. La ligne 4 permet de sortir un fichier avec un couple (Pays, nbVille).

On obtient alors un fichier comme celui-ci après le job de map reduce :

France 2

Italie 1

```

mohhocini@noree:~$ hdfs dfs -tail hdfs://10.0.105.13:9000/urez/part-r-000000
jm      2661
jo      2031
jp      17849
ke      2910
kg      1580
kh      14447
ki      165
km      564
kn      187
kp      29358
kr      44601
kw      283
ky      73
kz      14745
la      16477
lb      6782
lc      211
li      42
lk      18214
lr      8481
ls      439
lt      17341
lu      966
lv      9233
ly      1998
ma      28356
mc      7
md      5978

```

Figure2 : Nombre de villes par pays

3.2 Calcul de la population de chaque pays

Exemple avec un fichier d'entrée comme celui-ci :

France, Bordeaux, 100

France, Paris, 100

Italie, Rome, 100

Étape mapper :

1 : `string data[] = value.toString.split(",");`

2 : `context.write(data[1],data[population]);`

La ligne 1 va nous permettre de lire chaque ligne de notre fichier d'entrée, de séparer chaque champs, initialement délimité par un char spécial ",", et de les placer dans un tableau.

La ligne 2 va nous permettre d'inscrire en sortie du mapper et d'associer un nom de pays avec la population de la ville analysée.

Fichier à la sortie du mappeur sous forme(key,value) :

France[100]

France[100]

Italie[100]

Étape du shuffle et sort :

France[100,100]

Italie[100]

```
Étape reducer ;  
1 : int pop = 0 ;  
2 : foreach v in values :  
3 : pop = pop + v ;  
4 : write(key,pop) ;
```

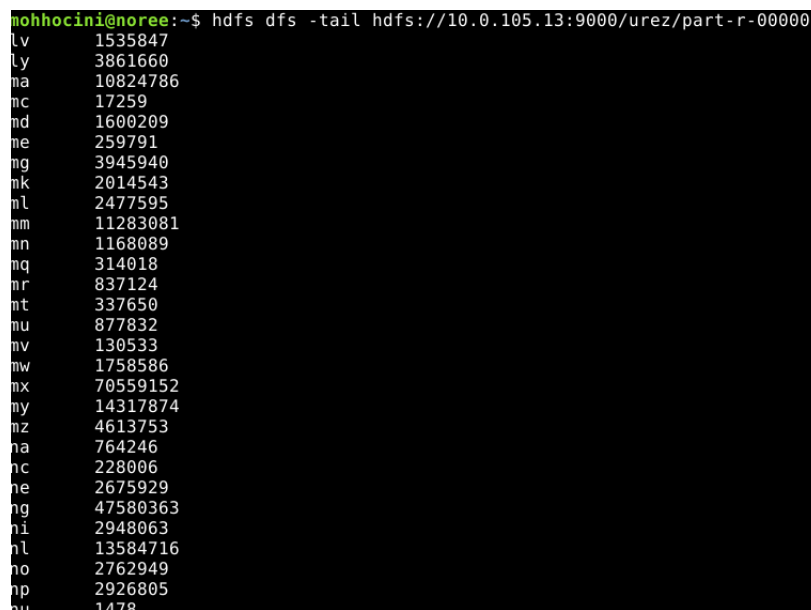
La ligne 1 nous permet d'initialiser un compteur. Il contiendra la valeur du nombre de ville par pays.

La ligne 2 nous permet de boucler pays par pays du fichier qui sort de la phase de shuffle et sort. La ligne 3 ajoute la population de toute les villes d'un même pays. La ligne 4 permet de sortir un fichier avec un couple (Pays, popPays).

On obtient alors un fichier comme celui-ci après le job de map reduce :

France 200

Italie 100



```
mohhocini@noree:~$ hdfs dfs -tail hdfs://10.0.105.13:9000/urez/part-r-00000  
lv      1535847  
ly      3861660  
na      10824786  
nc      17259  
nd      1600209  
ne      259791  
ng      3945940  
nk      2014543  
nl      2477595  
nm      11283081  
nn      1168089  
nq      314018  
nr      837124  
nt      337650  
nu      877832  
nv      130533  
nw      1758586  
nx      70559152  
ny      14317874  
nz      4613753  
oa      764246  
oc      228006  
oe      2675929  
of      47580363  
oi      2948063  
ol      13584716  
oo      2762949  
op      2926805  
ou      1478
```

Figure3 : Population par pays

Toutes les parties ont été bien implémentées et testées sur notre cluster et les résultats sont montrés dans les captures d'écran qu'on a fait pour tout les cas demandés, dans notre code java, à chaque fois qu'on réalise un traitement on met les précédents en commentaire pour ne pas mélanger le fichier de sortie et pour que notre jar se lance bien avec yarn sans erreurs, ce que vous pouvez le remarquer dans notre code qui est bien commenté afin de distinguer et tester les différentes parties. Dans le code fournis, vous trouvez le jar généré pour les compteurs, et pour générer un autre jar qui répond à l'exercice trois avec ses deux parties, il suffit de commenté les lignes de code (26 à 59) et aussi commenter la ligne 89 du Reducer de enlever les commentaires pour la partie que vous voulez tester.