

UNIVERSITÉ DE BORDEAUX I

PROGRAMMATION LARGE
ECHELLE

Compte rendu du TP 3
Hadoop MapReduce

Réalisé par :

GAMELIN Antoine
LASSOUANI Sofiane

TABLE DES MATIÈRES

1	Introduction	1
2	Partie 1	2
2.1	Utilisation d'une démo hadoop de MapReduce	2
3	Projet	4
3.1	Exercice 1 : Filtrage	4
3.1.1	Structure du fichier	4
3.1.2	Premier programme	4
3.2	Exercice 2 : Compteurs	5
4	Conclusion	7

TABLE DES FIGURES

2.1	exécution pi 32 32	3
3.1	Compteurs	6

INTRODUCTION

À travers ce TP, nous allons mettre en place une application basique de mapreduce afin de mettre en pratique les connaissances étudiées en cours.

Nous avons réalisés les tâches suivantes :

- Partie 1 : Utilisation d'une démo hadoop de MapReduce (Monte-Carlo)
- Partie 2 : Réalisation d'une application MapReduce
- Exercice 1 : Filtrage
- Exercice 2 : Compteurs

PARTIE 1

2.1 Utilisation d'une démo hadoop de MapReduce

Pour tester le programme nous avons utiliser la commande suivante :

```
yarn jar /espace/alperrot-cluster1/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.3.jar pi 32 32
```

Ce programme permet de calculer une valeur approximative de Pi.

Le premier argument 32 permet de définir sur combien de machines(processeurs) nous allons effectué ce calcul.

Dans notre cas, nous allons effectué les calculs silmutanément sur 32 machines (parfois il peut y avoir plusieurs instance sur la même machine si le processeur dispose de plusieurs coeur) en fonction des ressources disponibles.

Le deuxième argument permet de définir le nombre de points qu'on va placer dans le carré.

Le fonctionnement de ce programme consiste à placer des points dans un carré, ou un cercle y est inscrit dans ce carré. Le but du programme est de calculer une valeur approximative de Pi en faisant une comparaison entre le nombre de point dans le cercle et ceux qui sont en dehors.

Plus nous allons placer de point, et plus la valeur de Pi serait plus détaillés.

Voici les résultats du programme de démo :

```

HDFS: Number of bytes read=8566
HDFS: Number of bytes written=215
HDFS: Number of read operations=131
HDFS: Number of large read operations=0
HDFS: Number of write operations=3
Job Counters
  Launched map tasks=32
  Launched reduce tasks=1
  Data-local map tasks=25
  Rack-local map tasks=7
  Total time spent by all maps in occupied slots (ms)=665326
  Total time spent by all reduces in occupied slots (ms)=47158
  Total time spent by all map tasks (ms)=665326
  Total time spent by all reduce tasks (ms)=47158
  Total vcore-milliseconds taken by all map tasks=665326
  Total vcore-milliseconds taken by all reduce tasks=47158
  Total megabyte-milliseconds taken by all map tasks=681293824
  Total megabyte-milliseconds taken by all reduce tasks=48289792
Map-Reduce Framework
  Map input records=32
  Map output records=64
  Map output bytes=576
  Map output materialized bytes=896
  Input split bytes=4790
  Combine input records=0
  Combine output records=0
  Reduce input groups=2
  Reduce shuffle bytes=896
  Reduce input records=64
  Reduce output records=0
  Spilled Records=128
  Shuffled Maps =32
  Failed Shuffles=0
  Merged Map outputs=32
  GC time elapsed (ms)=3563
  CPU time spent (ms)=15430
  Physical memory (bytes) snapshot=9425571840
  Virtual memory (bytes) snapshot=94073323520
  Total committed heap usage (bytes)=9193914368
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=3776
File Output Format Counters
  Bytes Written=97
Job Finished in 62.633 seconds
Estimated value of Pi is 3.148437500000000000000000
antgamelin@goldberry:/espace/alperrot-cluster1/hadoop/share/hadoop/mapreduce$

```

FIGURE 2.1 – exécution pi 32 32

PROJET

3.1 Exercice 1 : Filtrage

3.1.1 Structure du fichier

Afin de connaître la structure du fichiers, nous avons utilisé la commande `hdfs tail`, pour afficher les dernières lignes :

```
hdfs dfs -tail hdfs ://10.0.205.3 :9000/user/raw_data/worldcitiespop.txt
```

À partir de cette commande nous avons pu nous apercevoir que le fichier était de type CSV avec les informations suivante :

Country,City,AccentCity,Region,Population,Latitude,Longitude

3.1.2 Premier programme

Le premier programme consiste à réaliser un MapReduce qui récupère le fichier de villes en entrée et en sortie nous voulons afficher uniquement les villes où le nombre de population a été renseignée

Pour cela, nous avons splitter chaque ligne par le caractère de délimitation, dans notre cas la virgule. Nous avons un tableau de String dans la variable `tokens`. Puis dans une variable de type `IntWritable` que nous avons nommé `result`, nous mettons un booleen, qui est défini à (1) si la population est renseignée, (0) sinon. Et nous envoyons le résultat pour le réducteur.

Pour cet exercice nous aurions pu nous passer du reducer, et dans le mapper mettre une condition qui n'envoie que les lignes contenant une population.

Le reducer récupère la sortie de notre Mapper. En sortie il ne prend en compte que les lignes où la valeur de **value** est égale à (1) ce qui veut dire que la population est renseignée.

1. Mapper

En cours , nous avons vue que le mapper possède une entrée de type texte et nous retourne un couple `<key : value>`.

Dans notre cas, le texte en entrée est : ligne par ligne du fichier `wolrdcitiespop.txt`, nous mettons une condition concernant la première ligne afin de l'ignorer car celle-ci contient l'entête du fichier.

Le champs Population est la 5 ème valeur du fichier, nous testons donc si la valeur du 4ème élément du split n'est pas vide. (Le split commence avec un indice de valeur 0).

Nous retournons le couple suivant pour chacune des ligne :

key : Ligne du fichier qu'on a obtenu en entrée

value : 0 dans le cas ou la population n'est pas renseigné, 1 sinon

2. Reducer

En cours, nous avons étudié que le reducer récupère la sortie du mapper en entrée et retourne également un couple de type : `<key : value>`.

Dans notre cas, nous faisons une condition sur la variable value, dans le cas ou elle est égale à 1, nous recopions en sortie le couple suivant. `<key , null>`

Ce programme a été exécuté avec la commande

```
yarn jar TP3.jar TP3 hdfs :/10.0.205.3 :9000/user/raw_data/worldcitiespop.txt  
hdfs :/10.0.205.3 :9000/user/antgamelin/filtrage
```

Nous avons regardé sur le WEB UI de Yarn, l'avancement du projet, et sur le WEB UI du cluster, le fonctionnement du MapReduce. Celui-ci créer un répertoire (défini lors du dernier paramètre de l'exécution du programme avec la commande yarn) avec des fichiers nommés `Part-0000.txt`. Chaque reducer envoie son résultat dans un fichier qui lui est propre. Dans notre cas, nous avons demandé à ce que le travail soit réalisé dans un unique processeur.

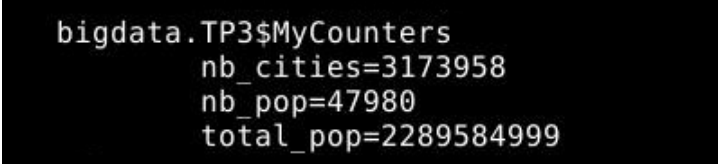
3.2 Exercice 2 : Compteurs

Le but de cet exercice est d'utiliser `getCounter(String, String)` et d'implémenter trois compteurs, `"nb_cities"`, `"nb_pop"`, `"total_pop"` , afin de compter le nombre de villes valide (nombre de villes dans le fichier `worldcitiespop.txt`); le nombre de villes avec une population renseignée et le nombre d'habitants de toutes les villes. Pour cela nous avons créé une structure `MyCounters` avec une énumération des trois variables citées ci-dessus.

Pour chaque appel du mapper, nous incrémentons la variable de `nb_cities` de 1, sauf pour la première ligne, que nous ignorons.

Dans le cas ou la population est renseignée (variable `result = 1`) nous incrémentons la variable `nb_pop` de 1, et nous additionnons la variable `total_pop` du nombre de personnes dans la ville.

En lançant le programme, nous pouvons dire que le fichier contient plus de 3 millions de ville, sur ces trois millions nous avons un peu moins de 50 000 villes où la population est renseigné et la somme de ces populations est égale à plus de 2 milliards de personnes.

A terminal window with a black background and white text. It displays the output of a command to print the 'MyCounters' field of a variable named 'bigdata.TP3'. The output shows three lines of data: the number of cities, the number of populations, and the total population.

```
bigdata.TP3$MyCounters
nb_cities=3173958
nb_pop=47980
total_pop=2289584999
```

FIGURE 3.1 – Compteurs

CONCLUSION

Ce TP nous a permis de mettre en pratique un programme très basique faisant appel au MapReduce de hadoop.

Cela nous a permis de comprendre le fonctionnement du MapReduce au niveau des datanodes. Un bloc de données est envoyé au processeur, celui fait des calculs et renvoi des résultat du type <key :value>. Puis, une fois que le réducteur a terminé son execution il enregistre les données sur le cluster dans un fichier qui lui est propre.