

UNIVERSITE DE BORDEAUX

**Programmation large échelle**  
**Rapport du TP6**  
**MapReduce Pattern : Top K**

**Par :**

HOCINI Mohamed Fouad

MAURICE Bastien

**Enseignant :**

David Auber

19 novembre 2017

Ce TP6, nous a permis d'utiliser le pattern TopK, ce pattern consiste à sélectionner les k plus grand/petit ... etc d'un ensemble et dans ce TP nous l'avons utilisé pour trouver les k villes les plus peuplées au monde de notre fichier "worldcitiespop.txt".

## 1 Exercice 1

Dans cet exercice, nous avons écrit un Mapper et un Reducer avec un k=10. Nous avons utilisé un TreeMap qui a comme clé la population et comme valeur les informations de la ville, ce qui nous permet d'avoir un ensemble de ville triées par population.

Le Mapper place chaque ville dans notre treeMap et teste si la taille dépasse k ou non, et si c'est le cas la valeur de la clés (population) la plus petite sera supprimée.

A la fin de cette opération, notre treeMap contiendra le TopK ville recherché parmi toutes les villes de notre fichier "worldcitiespop.txt"

## 2 Exercice 2

Dans cette partie, nous avons ajouté un Combiner afin d'alléger la charge de travail du Reducer dans le cas où un grand nombre de Mapper sont utilisés.

Cette partie était la partie la plus dur à réaliser et nous avons eu un peu de problème avec parce que nous n'avons pas eu les résultats attendus directement mais nous avons réussi à la fin à fixer le problème et ça a marcher parfaitement.

Avec l'utilisation des Combiner, chaque ordinateur dans notre cluster a son propre Combiner contrairement au Reducer qui reçoit le nombre d'ordinateur \* le nombre de Mapper par ordinateur ...

## 3 Exercice 3

Dans cette dernière partie, nous avons ajouté k comme paramètre au lieu de le fixer dans notre code, cette valeur "k" passée en argument à l'appel du job est stockée dans la configuration au nom de k ensuite notre treeMap*j* récupère cet valeur afin qu'il ne l'a dépasse pas lors de son traitement de notre fichier d'entrée, l'appel au TopK avec les valeurs : 100, 1000, 10000 s'exécute à des vitesses similaires au topK à 10 et les résultats concordent.

## 4 Résultats

Nous avons fait quelques tests pour voir si nos résultats sont bien ou non, nous citons trois tests ici avec un k=1, 10 , 100 et nous pouvons voir que les résultats dans les figures suivantes :

```
mohhocini@noree:~$ hdfs dfs -tail /user/part-r-00000  
31480498,Tokyo
```

Figure3 : TopK : k=1.

```
mohhocini@noree:~$ hdfs dfs -tail /user/part-r-00000  
9797536,Istanbul  
10021437,São Paulo  
10323448,Seoul  
10381288,Moscow  
10443877,Manila  
10928270,New Delhi  
11627378,Karachi  
12692717,Bombay  
14608512,Shanghai  
31480498,Tokyo
```

Figure1 : TopK : k=10.

```

mohhocini@noree:~$ hdfs dfs -tail /user/part-r-00000
935968,Pune
3043589,Kabul
3087010,Nanjing
3102644,Madrid
3120340,Durban
3152825,Guangzhou
3229883,Harbin
3268513,Montreal
3398362,Berlin
3433504,Cape Town
3467426,Ho Chi Minh City
3469290,Riyadh
3512192,Shenyang
3519177,Ankara
3547809,Singapore
3565810,Ibadan
3598199,Hyderabad
3609698,Casablanca
3626204,Kano
3692570,Abidjan
3719933,Ahmadabad
3730212,Melbourne
3766207,Tianjin
3811512,Alexandria
3877129,Los Angeles
3950437,Chengdu
3953191,Xian
3967028,Chongqing
4039751,Saint Petersburg
4184206,Wuhan
4328416,Madras
4394585,Sydney
4477782,Rangoon
4612187,Toronto
4631819,Calcutta
4837248,Santiago
4931603,Bangalore
5104475,Bangkok
5672516,Baghdad
6023742,Rio de Janeiro
6312576,Lahore
6493177,Dhaka
7102602,Bogota
7421228,London
7480601,Peking
7646786,Lima
7734602,Cairo
7787832,Kinshasa
8107916,New York
8540306,Jakarta
8720916,Mexico
8789133,Lagos
9797536,Istanbul
10021437,Sao Paulo
10323448,Seoul
10381288,Moscow
10443877,Manila
10928270,New Delhi
11627378,Karachi
12692717,Bombay
14608512,Shanghai
31480498,Tokyo

```

Figure2 : TopK : k=100.