Bérénice Faltrept
Kevin Baptista

M2 Génie Logiciel
Rapport TD4 : MapReduce 2/2

usage de notre archive :

$ yarn jar ourArchive.jar inputfile outputdir ["count" | "all"] step

- count : count cities number by step.
  all : write all our informations about cities population.
- step is a double which define our granularity.

Cet usage s'affichera si vous ne mettez pas les arguments ajoutés lors de l'exercice 3 :

$ yarn jar ourArchive.jar inputfile outputdir

Exercice 1 : Histogramme

Nous avons pour la première fois utilisé le **Combiner** afin de réduire drastiquement la charge sur le réseau, et gagner en temps de traitements lors du **Reducer**.
On passe comme clé *10^(1+floor(log(population))*. Cela sous entend que nos classes d'équivalences vont d'une puissance de 10 à celle qui lui est inférieur (de 10 à 1, de 100 à 10 etc …).
On passait comme value la population, avant de réaliser l'exercice 2, en prévoyant de pouvoir travailler sur cette population.
Dans le **Combiner**, on incrémente un compteur vide au départ pour compter le nombre de ville dans une classe d'équivalence. On les passe ensuite au **Reducer** pour additionner les compteurs de chaque classe d'équivalence par namenode.

Exercice 2 : Résumé

Nous avons dans un premier temps tenté d'écrire dans le fichier (lors du **Reducer**) avec un **TupleWritable** contenant les différentes valeurs mais le fichier contenait alors le lien vers chaque tuple et non son contenu.
Nous avons aussi essayés avec un **ArrayWritable**, avec le même problème.
Nous présumions alors que les structures composites avaient un traitement spécifiques lorsqu'on les utilisait comme **OutputValues**, comme par exemple : écrites à la suites dans un String.
Cette idée s'avérant fausse, nous avons donc ajouté ce traitement puis retourné des **Text** et avons ainsi pu générer la ligne de titre dans le fichier via le setup de notre **Reducer**.

On utilise désormais une classe PopWritable qui nous sert à stocker les valeurs min, max, le nombre de villes et la population totale lors des transmissions. Le **Mapper** envoie une instance de PopWritable par ville au **Combiner**.

Celui-ci combine les valeurs de chaque objet PopWritable ayant la même clé dans un nouvel objet PopWritable qu'il envoie au **Reducer.** Le **Reducer** combine donc les valeurs objets PopWritable de chaque namenode.

Exercice 3 : Paramétrage

Nous avons fait deux changements impactés par des arguments.

- On passe un premier paramètre déterminant l'affiche. Soit on écrit seulement le nombre de ville par classe d'équivalence, soit on écrit également les minimums, maximums et moyennes. Ces options sont "**count**" ou "**all**".
- En deuxième paramètre, on précise la **base logarithmique** qui va déterminer la précision de l'échelle de notre histogramme. Cette valeur doit être supérieure à 1, autrement on la contraint à la base 10. Une valeur proche de 1 donnera une échelle plus précise, alors qu'une valeur plus grande sera moins précise. Si il n'y a pas d'argument, on détermine la encore la base logarithmique comme égale à 10.

Pour changer la base logarithmique, on utilise l'identité logarithmique suivante :

$$log_a(b) = log_c(b) \div log_c(a)$$

où a est la granularité passée en paramètre et b est la population de la ville et c est égale à exp .