UNIVERSITE DE BORDEAUX

Programmation large échelle
Rapport du TP4
Map Reduce 2/2

Par :
HOCINI Mohamed Fouad
MAURICE Bastien

Enseignant :
David Auber

22 octobre 2017

Ce TP4, 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 de réaliser n histogramme de fréquence des villes dont ont a la population avec l'utilisation d'une échelle logarithmique pour déterminer les classes d'équivalences des villes, ainsi que la population moyenne de chaque catégorie et le minimum et maximum population 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 réaliser un histogramme des villes données dans le fichier worldcitiespop.txt et qui contient les catégories de ville 10, 100, 1000 ... ainsi que le nombre de ville dans chaque catégorie, pour cela nous avons commencé par enlever toutes les lignes dont la population n'est pas connue dans notre fichier worldcitiespop.txt , après l'utilisation d'un split biensur sur notre entrée value qui est une ligne dans le fichier et comme résultat on avant un tableau de String : Data[], dans lequel la cinquième colonne représente la population.

- Dans le Mapper :
  Pour celà on a utilisé la bibliothèque Math de java, on passant par plusieurs étapes :
  - log10() :
    Double popLog = Math.log10(Double.parseDouble(Data[4])) ; Si la ille contient une population entre [10, 100], cette ligne de code renvoie 100, et ça sera la même chose pour les autres catégories.
  - Floor() : Integer logRound = (int) Math.floor(popLog) ;
  - Pow() : Integer pow = (int) Math.pow(10, logRound) ;
  - Resultats : context.write(new Text(pow.toString()), new IntWritable(Integer.parseInt(Data
- Dans le Reducer :
  Le reducer après air reçu les données du Mapper regroupe les valeurs selon leurs clés, (key, value). pour chaque clé (catégorie), on parcourt la liste des valeurs et on incrémente notre variable counter par 1, à la fin cette variable contiendra le nombre de ville de la catégorie et ce qu'on va écrire dans notre fichier de sortie comme couple (key, value) avec :
  - key : catégorie.

  - value : nombre de ville dans cette catégorie.

Afin d'éviter les doublant comme demandé, on a utilisé une nouvelle classe (FilterCities) avec une nouveau job qui nous renvoie un fichier filtré des villes, et ce dernier sera utilisé pour répondre aux questions concernant l'histogramme.

2 Exercice 2

L'objectif de cet deuxième exercice était de modifier le programme précédent et de calculer la population moyenne (avg), la population minimale (min) et la population maximale(max) de chaque classe d'équivalence et de les afficher avec le nombre de ville (count) de l'exercice précédent.

Pour cela, nous avons crée une nouvelle classe (Myritable) qui hérite de la classe writable et qui nous renvoie un nouveau type qui contient tout les champs demandé pour qu'on puisse l'afficher dans notre context avec notre clés.

Au début, on a crée juste un String qui concatène les champs demandés et ça a donnée les résultats attendus, et après on est passé à la création de ce nouveau type, afin que ce traitement se passe dans une nouvelle classe.

Aussi, on a crée une nouvelle classe (Main) qui contient deux méthodes run() et run2() pour lancer les deux jobs de filtre et de l'histogramme respectivement pour la création et la configuration de nos job, ce qui était fait dans le main, et là

3 Exercice 3

Le but de cet exercice est de donner à l'utilisateur la possibilité de fragmenter les catégorie de notre histogramme au lieu de travailler avec le log10 tout le temps, du coup on peut avoir des catégories du genre 10, 20, 30 ... au lieu de 10, 100, 1000 ...

Pour cela, on a ajouter un argument, d'ailleurs, nous devons entrer trois arguments lors du lancement de la commande yarn jar MonJar.jar arg[0] arg[1] arg[2] :

— arg[0] : Le nombre de fragments souhaité par l'utilisateur.

— arg[1] : Fichier d'entrée.

— arg[2] : Chemin pour le fichier de sortie.

Dans notre main, on prend l'argument de l'utilisateur et on le met dans une variable de configuration "nbFragment" : conf.set("nbFragment", arg[0]).

Une fois notre paramètre récupéré, on peut faire notre histogramme et créer des fragments selon ce dernier.

— Mapper : Pour ajouter cette fonctionnalité, nous avons ajouté la récupération du paramètre saisi par l'utilisateur (context.getConfiguration().get("nbFragment"))

— Reducer : Nous avons rien changé.