Université de Bordeaux I

Programmation large échelle

Compte rendu du TP 4
Hadoop MapReduce 2/2

Réalisé par :

GAMELIN Antoine
LASSOUANI Sofiane

Université Bordeaux 1 (2016–2017) – Master Génie Logiciel (UE PLE)

TABLE DES MATIÈRES

Chapitre 1

Introduction

Ce TP4 fait suite à l'ancien TP. Il nous est demandé de mettre en place une application qui permet de faire des statistiques sur des gros fichiers.

Ce TP est découpé en trois parties :

— Réalisation d'un histogramme avec une echelle Logarithmique (base 10).

— Ajout de statistiques sur cette histogramme avec valeur max, min et la moyenne

— Parametrage, ajout d'une variable en argument pour définir le pas entre deux valeurs de l'histogramme.

Chapitre 2

Projet

2.1 Exercie 1 : Histogramme

2.1.1 Mapper

Dans cet exercice, on doit écrire un programme MapReduce qui calcule l'histogramme de fréquence des villes dont a la population est renseigné en utilisant une echelle logarithmique pour détérminer les classes d'équivalence.

Nous avons récupéré notre code du TP3, qui nous permet de filtrer les villes où la population a été renseignée. Puis nous faisons appel à la fonction getRangePop(int population) qui prend en paramètre la population et qui nous renvoi le log10 de ce nombre. Cette valeur sera la key qu'on enverra au mapper.

Rappel : Un mappeur lit et renvoit des : <key,value>
Concernant la value, ça sera un intWritable de valeur 1.

2.1.2 Reducer

Le réducer possède des messages de ce format : <intWritable, Iterable<intWritable» Le premier intWritable contient la key de la fonction getRangePop (cf paragraphe précedent), et en sortie une liste avec des intWritable de valeur 1.

Le réduceur va envoyer en sortie <intWritable,intWritable>, le premier intWritable , retourne tout simplement la clef, le second intWritable, retourne la taille de la liste pour définir le nombre de villes dans cette tranche de population.

2.2 Exercice 2 : Histogramme avec statistiques

2.2.1 Description

Le but de cet exercice est de réaliser des stats sur l'histogramme de population. En plus du nombre de population, nous souhaitons connaître la moyenne, le minimum et le maximum par tranche de population.

2.2.2 StatsWritable

StatsWritable implemente **Writable** avec en attribut le minimum, maximum, count et sum. Ces informations permettront d'enregistrer les informations pour les statistiques par tranche de population.
Cette classe demande d'implementer les 4 méthodes de Writable (readFields, write, compareTo, hashCode)

Cette classe sera utilisée comme output du Mapper et donc en input du combiner et reducer.

2.2.3 Mapper

Nous avons mis à jour le mapper, pour qu'en sortie au lieu d'envoyer un simple intWritable, nous envoyons un StatsWritable. L'appel va initialiser les attributs avec les valeurs suivante :

- count : 1
- sum : population
- min : population
- max : population

Ces données sont ensuite envoyé dans le combiner.

2.2.4 Combiner

Pour éviter d'envoyer beaucoup de données aux différents réduceurs, nous faisons appels à un combiner pour regrouper les données avec la même clefs.

Le combiner reçoit en entrée : <intWritable, Iterable<StatsWritable> Le but de ce combiner c'est de faire les calculs du minimum et maximum partiellement , en se basant sur les données du mappers qui lui ont été envoyé.

2.2.5 Reducer

Le reducer dans cet exercice récupere les sorties partiels du combiner et les regroupe pour calculer la moyenne, le maximum et le minimum pour chaque tranche de population.

Nous avons créer un setup sur le reducer afin d'afficher un entête, c'est pour cela que le out du reducer n'est plus <intWritable,Text> mais <Text, Text>

2.3 Exercie 3 : Paramètrage

Le but de cette exercice est de peaufiner l'echelle logarithme de base 10, en ajoutant des pas entre chaque tranche de population.

Cette information est renseigné en troisième argument lors du lancement du programme avec la commande yarn.

Nous enregistrons cette information dans un fichier de configuration, qui pourra être envoyer à tous les mappers. Il est enregistré grâce à la méthode : conf.set("step", args[2]) ; Et le mapper récupère l'information grâce à la méthode : context.getConfiguration().getInt("step", 10) ;

Pour réaliser ce système de pas entre intervalle de population, nous avons mis à jour la fonction getRangePop(int population). Au lieu de retourner la puissance de 10 la plus élevé pour ce nombre nous retournons une valeur plus précise en fonction du pas définit en paramètre.

Cette valeur est calculé de la manière suivante :

— On récupère la classe d'appartenance en puissance de 10 de la population (On nommera cette valeur, popBase10)

— Nous multiplions cette valeur popBase10 par 9. Pour savoir le nombre de pas possible dans la classe d'appartenance. (Exemple si la popBase10 est 100, cela signifie que la population est >= à 100 et < à 1000, nous avons popBase10 * 9 = 100 * 9 = 900 pas possible dans cette intervalle de population pour avoir plus de précision)

— Ensuite nous calculons de combien sont espacés les pas. Pour cela nous prenons la valeur de popBase10x9 , définit dans l'étape précédente, puis nous la divisons par le nombre de pas (définit en troisième argument de la commande yarn). Nous nommerons cette valeur : rangeStep

— Et enfin nous faisons une boucle avec i commençant à 0, et nous testons la condition suivante si la popBase10+(i * rangeStep) est inférieur ou égale à la population dans ces cas là nous incrementons i. Puis nous retournons la valeur popBase10+((i-1) * rangeStep)

  En effet comme dans la boucle nous avons inférieur ou égale, le i sera incrémenté au moins une fois, donc nous n'aurons pas de valeur négative avec i-1.

2.3.1 Difficulté rencontrée

Une des difficulté rencontré concerne la comprehension de cette exercice. Une fois assimilé nous avons dû trouver l'algorithme permettant de définir la nouvelle key. Nous avons rencontrés un autre soucis, cela concerne quand le pas était supérieur à la popBase10, en effet au départ nous avons pu faire que des pas de 1 à 9. En effet, si le pas était de 10, dans certain cas la valeur de rangeStep nous retourner 0 (en effet le Diviseur > Dividende) ce qui créa une boucle infini. C'est pourquoi nous avons du réalisé une condition, si la valeur step était supérieur à la valeur popBase10x9, nous affectons la valeur de step à popBase10x9.

Chapitre 3

Conclusion

Cette suite de TP nous a permi de mettre en pratique un combiner, ce que nous avons pas eu l'utilité pour l'ancien TP et également de pouvoir affecter un fichier de configuration qui pourra être lu dans les mappers.

Lorsque nous faisons appel à notre programme de MapReduce, le mapper lis les lignes du fichiers les formate avec une key specifique et les envoi au combiner. Le combiner se charge de faire un traitement partiel, il compte, fait la somme et trouve le min et le max pour chaque key présent dans le mapper et envoi le résultat au reducer.

Le reducer quand à lui récupère les données et calcul la moyenne puis les enregistre dans un fichier nommé part-r-xxxx , ou xxxx est un entier qui s'autoincremente. Le chemin est définit en deuxième argument lors du lancement du programme avec la commande yarn.