Université de Bordeaux I

Programmation large échelle

Projet clustering à grande échelle avec k-means

Réalisé par :

GAMELIN Antoine
LASSOUANI Sofiane

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

TABLE DES MATIÈRES

Chapitre 1

Introduction

Le but de ce projet est d'implémenter l'algorithme K-Means qui pourra être utilisé sur différents fichiers d'entrées.

Dans un premier temps nous implémenterons un programme K-Means permettant de traiter des points de dimensions N. Le K-Means prendra fin dès que les derniers centroids convergent avec les nouveaux centroids calculés. (La condition de convergence : $\sum_{i=0}^{k} dist(old_i, new_i) <= 0.1$)

Une fois que les centroids convergent, nous faisons une concaténation du fichier d'origine avec l'indice du centroid le plus proche.

Dans un second temps, nous réalisons un KMeans Hierarchique de niveau N. Au lieu de générer un seul fichier de sortie, nous faisons K fichiers (fichier0.txt, fichier1.txt, ...), ou chaque fichier contiendra respectivement les données les plus proche du centroid. (Dans le fichier0.txt nous enregistrons toutes les lignes que nous avons affectées au cluster 0, dans le fichier1.txt toutes les lignes que nous avons affectées au cluster 1 ...).

Ensuite, nous réalisons un KMeans sur ces fichiers de sorties, ce qui implique qu'à chaque niveau de profondeur N nous aurons : K^n fichiers.

En dernier temps, nous allons réaliser un étiquetage : pour chaque niveau du KMeans hiérarchique de trouver la mesure la plus grande pour chaque centroids calculé et de lui affecter le label.

Chapitre 2

K-means nD

2.1 Introduction

Dans un premier temps nous avons réalisé un KMeans qui s'effectuait sur des points d'une dimension en utilisant un Integer. qui par la suite a été remplacée par une liste de double pour pouvoir réaliser des centroids de n dimensions.

2.2 Structure du projet

2.2.1 MultiDemPoint

Comme mentionné ci-dessus, nous devons pouvoir implémenter des centroids à N dimensions, nous avons donc créé cette classe qui permet d'instancier des points de multidimensions. Cette classe possède une fonction permettant de calculer la distance euclidienne avec un autre point multidimensionnel et une méthode toString, pour pouvoir faire un affichage des points.

2.2.2 KMeans

Cette classe est la classe Main qui permet de lancer les différents job map-reduce. Nous avons une méthode getCentroids, qui permet de récupérer les premiers centroids du fichier input, une méthode saveCentroids, pour générer un fichier qui sera envoyé en DistributedCache au MapReduce et enfin une méthode qui permet de vérifier si les anciens et nouveaux centroids convergent.

2.2.3 CentroidWritable

Le centroidWritable est une classe à pour attribut une liste de double contennant l'addition des coordonnées de chaque point appartenant au même cluster et un attribut count. Cette classe permettra de faire la moyenne pour avoir les nouveaux centroids. Elle contient une méthode merge qui permet de combiner deux CentroidWritable, une méthode clone et une méthode avg pour faire la moyenne.

2.2.4 KMeansMapper

Le Mapper retourne un integer contenant l'indice du cluster et un CentroidWritable qui contient la somme des coordonnées et le nombre de points. Cela permettra au réduceur de faire la moyenne de tous les mappeurs.

2.2.5 KMeansReducer

Le réduceur calcul la moyenne des coordonnées du point pour avoir le nouveau centroids

2.2.6 KMeansResultMapper

Une fois que les centroids ont convergé, on refait un mapper qui trouve le cluster le plus proche de chaque point et qui le concatène avec la ligne de données.

2.3 Paramètres

Pour lancer le programme, il faut utiliser le JAR du KMeans hiérarchique et mettre la valeur N à 1.

yarn jar KMeans.jar (input) (output) (k) (separateur) 1 (col0 [col.. coln]) [BOOLEAN HEADER]

- INPUT : Mettre le lien HDFS pour récupérer le fichier
- OUTPUT : Le fichier où on enregistre le résultat du job
- K : Nombre de centroids que l'on souhaite
- SEPARATEUR : Caractère de séparateur du csv (pour le worldcitiespop c'est la ,)
- 1 : Paramètre N pour le KMeans hiérarchique, ici nous voulons faire un simple KMeans donc on l'initialise à 1
- col* : Ceux sont les indices des colonnes qui permettront de définir les coordonnées des points sur lequel on appliquera l'algorithme de KMeans
- Header : Paramètre facultatif qui permet de dire si le fichier contient un HEADER ou non.

2.3.1 Implémentation

Dans le programme main, nous envoyons via les conf, les valeurs de K, la dimension des points, les différentes colonnes qui permettent de définir les coordonnées du point. Le caractère de séparation et le booléen qui permet de dire si l'input contient un HEADER.

Dans un premier temps, nous enregistrons les K premières lignes qui composeront nos K premiers centroids dans un fichier sur le cluster :

/tmp/centroids_(startTime)/(numIteration)_0_0.dat

Le startTime est le timestamp qu'on génère au début du fichier pour s'assurer que si on souhaite lancer un second programme mapreduce sur le cluster, que cela ne fasse pas conflit avec le programme qui tourne déjà. Puis le numItération, c'est l'indice qui définit le numéro de l'itération avant que les points convergent.

- Nous les envoyons au Mappeur, via le distributedCache.

— Nous lançons des jobs jusqu'à ce que les points convergent.

Le but du mapper consiste de lire le fichier en Input et de calculer la distance de chaque points avec tous les centroids et de prendre la distance la plus petite pour trouver le point le plus proche.

Pour éviter de consommer la bande passante, nous enverrons seulement des sommes calculs de moyennes partiels au réduceur. C'est-à-dire que nous allons additionné chaque coordonnée des points et d'incrémenter le nombre de points. Cette moyenne partielle nous l'enregistrons dans un CentroidWritable.

Ces CentroidWritable nous les enregistrons dans un TreeMap de dimension K, ou en key nous avons l'indice du cluster et en valeur le CentroidWritable qui permettra de calculer la moyenne.

Chaque cleanup du mapper va envoyer K data au réduceur. Une optimisation du code serait possible en fusionnant les resultats des mappers d'un même datanode avec un combiner, pour que chaque datanode envoit sur le réseau uniquement K data au réduceur.

Le réduceur récupère les différentes sorties des mappeurs, fait un merge des centroidWritable et calcul le nouveau centroids en divisant la somme de coordonnées par le nombre de points.

Ensuite, le programme main, vérifie si les nouveaux centroids convergent avec les derniers centroids calculés dans le cas contraire il relance un map reduce avec les nouveaux centroids en référence.

Une fois que les centroids convergent on lance un dernier map qui recalcule une dernière fois le centroid le plus proche, et le concatène au fichier input. Dans ce cas nous n'avons pas besoin de réduceur.

2.4 Test du programme

Pour tester le programme : yarn jar KMeans.jar /data/worldcitiespop.txt /result 5 , 3 5 6 true

Sur le moodle les résultats attendu après la première opérations sont :
(22.346382519381972,-76.59420501154081) (-1.2181034436715423,27.33060673738127)
(5.52098955120269,-19.241255058544336) (24.054157266112593,87.30842322815704)
(51.58975483768719,28.80851692655404)

Nous obtenons le même résultat après la première itération :

```
--------------------------------
>> Nouveaux centroids calculés
--------------------------------

22.34629025850866;-76.59336969933236
-1.2181034436715423;27.33060673738127
5.520579081148455;-19.241063024065213
24.054157266112593;87.30842322815704
51.58975483768719;28.80851692655404
```

Chapitre 3

KMeans Hierarchique

3.1 Lancement du programme

Pour lancer le programme, même commande que le KMeans, mais on peut choisir la profondeur de l'arbre de KMeans grâce au paramètre N.

yarn jar KMeans.jar (input) (output) (k) (separateur) (N) (col0 [col.. coln]) [BOOLEAN HEADER]

3.2 Implémentation

La difficulté de cette implémentation c'est de bien définir le nom des inputs où l'on va exécuter les différents KMeans successif.

Pour cela dans le Main nous faisons deux boucles permettant de faire un parcours en largeur de l'arbre hiérarchique.

Dans un premier temps on change la sortie du KMeansResultMapper, au lieu d'enregistrer les fichiers dans un seul output, nous enregistrons les fichiers dans un multipleoutput.

Toutes les données sont enregistrées dans /tmp/(startTime)/(n)/(indiceDuFichier). Ou startTime est le timestamp du démarrage du programme, n qui permet de donner le niveau de la hierarchie et l'indice du fichier, c'est un nombre entre 0 et K^n-1.

Étant donné que le output génère un répertoire avec des part-m-0000, on réalise un merge par la suite du résultat dans un seul fichier qui se nomme : /tmp/output_(startTime)/n/(indiceDuFichier).dat qui sera par la suite le prochain input d'un KMeans pour n+1.

La première boucle du KMeans hiérarchique permet de descendre en profondeur donc c'est quand n est inférieur au N passé en paramètre. La seconde boucle permet d'accéder au fichier de 0 à K^n-1 , qui permet de faire le parcours en largeur.

A la fin des différents KMeans nous faisons un merge du /tmp/output_(startTime)/(nParam)/ dans le output définit en deuxième argument. (nParam est la valeur N passé en paramètre)

Chapitre 4

Étiquetage

4.1 Introduction

Le principe d'étiquetage c'est pour chaque niveau du KMeans hiérarchique, il faut donner l'étiquette où la mesure est la plus élevée.

4.2 Structure du projet

4.2.1 KMeans

C'est la classe Main ou on lance les N jobs.

4.2.2 MaxWritable

Cette classe permet d'avoir un couple contenant un string (l'étiquette) et une valeur (mesure).

4.2.3 KMeansMapper

Le KMeans mapper, lis le fichier KMeansHiearchique, puis récupère les combinaisons des clusters puis retourne au réduceur en clef, la combinaison des clusters et en valeur un MaxWritable contenant l'étiquette et la mesure la plus grande.

4.2.4 KMeansReducer

Le réduceur fait une comparaison des différentes sorties des mappers et sort l'étiquette avec la valeur la plus élevée pour chaque combinaison de clusters.

4.3 Paramètres

Pour lancer le programme il faut utiliser la commande avec les paramètres suivants :
yarn jar Labelize.jar (input) (output) (separateur) (mesure) (etiquette) (n0 [n...]) [header]

- INPUT : Mettre le lien HDFS du fichier KMeans Hierachique généré via la partie précédente
- OUTPUT : Le nom de fichier de sortie
- SEPARATEUR : Caractère de séparateur du fichier d'entrée
- MESURE : Indice de la colonne de mesure (format double)
- ETIQUETTE : Indice de la colonne d'étiquetage (format string)

— N0* : Liste des indices des différentes colonnes qui indiquent la succession des clusters
— Header : Paramètre facultatif qui permet de dire si le fichier contient un HEADER ou non. (format booleen)

4.4 Implémentation

Pour implémenter cette partie, dans le main nous envoyons au MapReduce via les conf, la valeur de N, le caractère de séparateur, l'indice de la colonne de mesure, l'indice de la colonne d'étiquetage et si le fichier contient un booléen ou non et les indices des colonnes contenant les numéros de cluster.

Ensuite, nous avons une boucle qui va de N à 0, qui permet de générer les différents fichiers.

À la première itération nous prenons le fichier input, puis pour les autres itérations nous prenons le dernier fichier généré. C'est un moyen d'optimiser et d'éviter de relire à chaque fois le fichier original car chaque ligne du fichier n sont une partie des fichiers n+1.

Chaque mapper possède un TreeMap contenant en clef une chaîne de caractère qui est une concaténation des combinaisons des clusters et en valeur c'est un MaxWritable, qui contient l'étiquette et la valeur maximale.

Le mapper va lire chaque mesure et vérifie si la dernière valeur sauvegardée dans le treemap avec la clef qui est la combinaison des cluster est supérieur à la valeur lue. Si c'est inférieur on met à jour le MaxWritable du treemap.

Et dans le cleanup des mappeurs on envoie les clefs avec les maxwritable au réduceur.

Le réduceur fait une comparaison des différents résultats provenant des mappeurs et fait un write uniquement du MaxWritable avec la valeur la plus élevée pour chaque étiquette.

Chapitre 5

Résultats attendu

5.1 Perfomances

Pour tester les performances sur le fichier gps avec K = 10, il suffit de lancer cette commande :

yarn jar KMeans.jar /data/simple-gps-points-120312.txt /tmp/perf.txt 10 , 1 0 1

17/01/14 18 :02 :24 INFO input.FileInputFormat : Total input paths to process : 1

17/01/14 18 :02 :24 INFO mapreduce.JobSubmitter : number of splits :412

17/01/14 18 :02 :24 INFO mapreduce.JobSubmitter : Submitting tokens for job : job_1484401368278_0345

17/01/14 18 :02 :24 INFO impl.YarnClientImpl : Submitted application application_1484401368278_0345

...

17/01/14 18 :03 :30 INFO mapreduce.Job : Job job_1484401368278_0345 completed successfully

Soit un total de 66 secondes et les nouveaux centroids calculés sont :

» Nouveaux centroids calculés

-8.15625E8 ;1.771892185E9

-9.0E8 ;1.29375749E9

-8.9115429E8 ;1.23750107E9

-4.0482212445674825E8 ;1.7355018384180253E9

-1.715540909167421E8 ;1.6408629111369462E9

-8.0625E8 ;1.2422014066666667E9

1.4754047883899724E8 ;1.3882724874831977E9

1.1507239039900072E8 ;1.1891318832650955E9

2.836120513683964E8 ;1.0509693209775418E9

4.766428790514428E8 ;1.3235953478066735E8

Ce test de performance a été réalisé dans la salle 205 avec 18 machines.s

5.2 Test du KMeans Hierarchique

Pour réaliser le fichier KMeansHiearchique avec K = 4 et N = 3, il faut taper cette commande : yarn jar KMeans.jar /data/worldcitiespop.txt /data/kcity_n3_k4.txt 4 , 3 5 6 L'exécution dure un peu plus d'une heure et après vérification, on n'a pas le même résultat que celui dans l'archive de moodle. Après multiples recherches on a pas réussi à trouver la provenance de l'erreur. Il y a des points qui sont relié à d'autre clusters. C'est en réalisant l'étiquetage qu'on s'est aperçu de l'erreur. Pour le n=1 , on a trois villes qui sont différentes par rapport aux résultats de l'archive.

```
antgamelin@goldberry:~/espaces/travail/neon-workspace/Labelize/target$ sort /tmp/etiq1.csv
abidjan,3692570.0,2,2
bogota,7102602.0,0,1
bombay,1.2692717E7,1,3
cairo,7734602.0,1,1
cape town,3433504.0,1,0
dhaka,6493177.0,3,0
lagos,8789133.0,2,1
london,7421228.0,2,0
manila,1.0443877E7,3,1
mexico,8720916.0,0,0
moscow,1.0381288E7,2,3
new york,8107916.0,0,2
sao paulo,1.0021437E7,0,3
sydney,4394585.0,3,2
tokyo,3.1480498E7,3,3
yekaterinburg,1287586.0,1,2
antgamelin@goldberry:~/espaces/travail/neon-workspace/Labelize/target$ sort /tmp/prof1.csv
baghdad,5672516.0,1,3
bogota,7102602.0,0,3
bombay,1.2692717E7,1,2
cape town,3433504.0,1,0
dhaka,6493177.0,3,3
istanbul,9797536.0,2,1
lagos,8789133.0,2,2
london,7421228.0,2,3
manila,1.0443877E7,3,1
mexico,8720916.0,0,2
moscow,1.0381288E7,2,0
new york,8107916.0,0,0
sao paulo,1.0021437E7,0,1
sydney,4394585.0,3,2
tokyo,3.1480498E7,3,0
umm durman,2810328.0,1,1
```

Après approfondissement des recherches ils semblerait que notre algorithme de KMeans soit défaillant.

anxic@pandaroux :/tmp/res_cities_k4_n3$ awk -F, '$8==1' kmeans_cities2 > /tmp/kcity1

anxic@pandaroux :/tmp/res_cities_k4_n3$ wc -l /tmp/kcity1

809348 /tmp/kcity1

Notre fichier KMeans Hierarchique pour n = 1 et le cluster numero 1

antgamelin@goldberry :/tmp$ awk -F, '$8==1' kcity4.txt > /tmp/kcity1

antgamelin@goldberry :/tmp$ wc -l /tmp/kcity1

833417 /tmp/kcity1

Dans le fichier resultat de moodle nous avons 809 348 lignes qui sont affectées au cluster 1, tandis que dans notre fichier KMeans Hiérarchique nous avons 833 417 lignes.

5.3 Etiquetage

Pour tester l'étiquetage, il faut utiliser la commande :

yarn jar Labelize.jar /kcity4.txt /kinfo , 4 1 7 8 9

En utilisant le fichier hiérarchique fournit dans l'archive, on obtiens les mêmes résultats que les fichiers de sorties de l'étiquetage.