Université de Bordeaux I

Programmation large échelle

Compte rendu du TP 5
Création d'InputFormat personnalisé

Réalisé par :

GAMELIN Antoine
LASSOUANI Sofiane

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

TABLE DES MATIÈRES

Chapitre 1

Introduction

1.1 Introduction

Par défaut, hadoop utilise InputFormat , qui hérite FileInputFormat , pour traiter les fichiers d'entrée.

Dans ce TP nous allons créer notre propre inputformat pour traiter nos données.

InputFormat décrit l'entrée spécifique de hadoop. RecordReader, quand à lui, il convertit le point de vue orienté octet de l'entrée fournie par le InputSplit, et présente un enregistrement orienté vue pour le Mapper/Reducer. Il assume ainsi la responsabilité de traiter les limites des enregistrements et la présentation des tâches avec des clés et des valeurs.

1.1.1 Fonctionnement

InputFormat de hadooop fait appel au getSplit() de la classe RandomPointInputFormat qui s'occupe de génerer autant de split que demandé,puis il fait appel au RecordReader qui se charge de génerer des point2DWritable.

Mot clé de ce TP : InputSplit, RecordReader, InputFormat.

Chapitre 2

Projet

2.1 Exercice 1 : Point2DWritable

Dans cette partie, il nous a été demander de créer un nouveau type de valeur nommé Point2DWritable qui va nous permetre de manipuler les Point2D.Double de java. Nous avons donc implémenté une classe Point2DWritable qui herite de **Writable**, pour cela nous avons implémenté les mêmes fonctions que Point2D.Double avec deux attributs de type double qui sont initialiser dans le constructeur. Nous avons également implémenté les methodes hérités de **Writable** afin de manipuler ce nouveau type de donnée.

2.2 Exercie 2 : RandomPointInputFormat

Contrairement aux TPs prcédéent où il nous été demander de lire un fichier texte en entrée, ce TP nous demande de générer pleins de points (Points2DWritable). Nous avons donc réalisé notre propre InputFormat d'hadoop.

2.2.1 Etape 1 : RandomPointInputFormat

Un InputFormat de hadoop définit un couple <key,value> dans notre cas, la key est un IntWritable qui permet de définir le numéro du point, et la value est un Point2DWritable, qui contient les coordonées du points.

Cette classe doit implémenter les deux méthodes d'un inputFormat :

1. **getSplits()**
   A partir d'un paramètre définit lors du lancement du programme avec la commande yarn, permet de générer un certains nombre de split. Le split consiste à découper un fichier en petit bloc, dans le cas d'un fichier texte , chaque bloc est définit jusqu'au saut de ligne. Cela permet de faire un traitement en parallèlle sur plusieurs machine pour améliorer les performances en temps de calcul. Nous générons des FakeInputSplit (cf ci-dessous)
2. **RecordReader()**
   Cette méthode consiste à creer un RecordReader qui permettra de créer le couple <key,value>

2.2.2 Etape 2 : FakeInputSplit

Cette classe est implementé afin de simuler des blocs de données qui contiendront nos points aléatoire générés par le RecordReader. Nous avons fais de sortie que chaque split contient des données, c'est pourquoi nous renvoyons 1 à la méthode **getLength()** quand à la méthode **getLocation()** étant donné que les données vont être générés à la volé, nous avons pas besoin de savoir à quel endroit elles sont stocker, c'est pourquoi nous retournons un tableau de String vide.

2.2.3 Etape 3 : RandomPointReader

Cette classe herite de **RecordReader**.
Le Mappe appelle la méthode nextKeyValue() pour vérifier qu'il y a encore des données à traités .S'il ya encore des données, on récupere ces valeurs en appelant les méthodes getCurrentKey() et getCurrentValue().

1. initialize () : Cette fonction permet d'initialiser le nombre de point qu'on va généré pour chaque split (Définit via le dernier paramètre).
2. nextKeyValue() : Est une fonction boolean qui renvoi true s'il ya encore des données a lire et false dans le cas contraire. Dans notre cas la méthode retourne true si on a pas génére le nombre de points définit en dernier paramètre au lancement du programme.
3. getCurrentKey() : Est une fonction qui retourne la clé des points générés.
4. getCurrentValue() : Prermet de retourner le Point2DWritable.

generateDouble() : Nous avons ajotuer cette méthode qui nous permet de générer une valeur aléatoire pour les coordonnées du points2DWritable

2.3 Exercice 3 : Test du RandomPointInputFormat

Pour tester le bon fonctionnement du programme, nous avons impélementer une fonction toString() pour les points2DWritable, puis nous avons enregistrer les numéros de points et coordonnées de points dans un fichier que nous enregistrons au cluster. Le chemin de ce fichier est définit en troisième paramètre.

2.4 Exercice 4 : Calcul de PI

Le but de cet exercice est de calculer Pi avec la Méthode de Monte-Carlo qui consiste générer au hasard des nombres x et y dans l'intervalle [0,1].
Pour réaliser cette exercice nous avons récupérer le code précédemment et avons adapter la génération des points pour qu'elle se réalise dans l'intervale [0,1].
On a pris connaissance à travers le lien fourmit dans l'énnoncé que la probabilité qu'un point généré M(x,y) appartient au quart de disque de rayon 1 ; ie : $x^2 + y^2 < 1$ est :

$$\frac{\pi}{4}$$

Pour cela nous avons utilisé les compteurs vue dans l'un des précedent TP. Nous l'avons nommé MyCountersPoints qui est de type enum. Il compte le nombre de point génére a l'interieur du quart de disque de rayon 1 **"in"** et à l'éxterieur **"out"**, et le nombre total de point généré **total**. enfin il suffit de faire le calcule suivant :

$$\pi = \left(\frac{in}{total}\right) * 4$$

2.4.1 Mapper

Dans cette partie du TP on a du effectuer une legère modification au mapper. A chaque fois qu'il recupere une valeur de type Point2DWritable, le mapper verifie si le point appartient au quart de disque de rayon 1. Dans ce cas on incremente la variable **"in"** sinon on incremente la variable **"out"**, pour ensuite pouvoir faire le calcule de π.

2.4.2 Valeur estimée de pi

La valeur estimé de π avec la méthode de Monte-Carlo avec 2000 splits et 500 points est :

```
Nombre de points dans le quart de cercle :  785190
Nom de points en dehors du quart de cercle :  214810
Nombre de points total :  1000000
Valeur de pi estimée: 3.14076
Valeur de pi estimée: 3.14076
```

FIGURE 2.1 – valeur estim de π

Chapitre 3

Conclusion

Ce TP nous a permis de se familiariser avec les InputFormat de Hadoop, ce qui nous a permis de comprendre d'avantage le fonctionnements des entrées inputs d'hadoop.