2016/2017

TP5

InputFormat

Naji ZAOUI

Yahya BACHIR

Dans le « default Package » on a créé toutes les classes nécessaires pour ce TP, les classes crées sont bien commentées et ce rapport est seulement pour mieux détailler ce qu'on a fait.

Exercie 1: Point2DWritbale

- C'est les objets qui vont être utilisé par le mapper et reducer, pour créer cette classe on s'est basés sur la classe IntWritable car les deux classes on presque le même rôle, IntWritable écrit un « int » alors que Point2DWritable écrit un « Point2D ». cette classe implémente l'interface Writable pour utiliser les 2 méthodes :
  - *Write(DataOutput)* pour écrite une instance de « Point2DWritable », ou plutôt les points x et y de l'instance.
  - *readFields(DataInput)* pour lire les 2 points 'x et y' d'une instance.

Exercie 2: RandomPointInputFormat

La classe RandomPointInputFormat étends la classe InputFormat qui décrit la spécification d'entrée pour un map-reduce job, notre classe va deviser l'entrée en utilisant la méthode *getSplit()* et elle va générer un RandomPointRecordReader en utilisant la méthode *createRecordReader()* :

- *getSplit(JobContex)* va retourner une liste de « FakeInputSplit » en laissant à l'utilisateur le choix du nombre de ces « FakeInputSplit » dans la ligne de commande, autrement dit le choix de combien de Mapper vont être utilisé pour réaliser ce Job.
- *createRecordReader(InputSplit,TaskAttemptContext)* qui va retourner un nouveaux RandomPointRecordReader , en faisant cela, il va appeler la méthode *initialize ()* de cette dernière, qui va récupérer le nombre maximum des points par « FakeInputSplit » ce nombre lui aussi est donner comme argument par l'utilisateur dans la ligne de commande, puis elle va appeler la méthode *nextValue()* tant qu'elle retourne « TRUE », ainsi, on crée notre nombre maximum de points.
- FakeInputSplit est une classe qui étends la classe abstraite InputSplit qui représente les données qui vont être traité par Un mapper, elle implémente aussi l'interface Writable qui va lui ajouter le fait d'avoir un protocole de sérialisation basé sur DataInput and DataOutput. Dans notre cas, on ne va rien mettre dans les méthodes implémentées de Writable.

- RandomPointRecordReader est une classe qui étend la classe abstraite Record<key, Value>, le RecordReader est le responsable de deviser les données sous forme de <Clés, Valeurs> et les passer au mapper.
  Dans notre code on a utilisé IntWritable comme Key et Point2DWritable comme value :
  - les variables de cette classe :
    - **Counter** : un entier qui va nous donner le nombre de point créés par cet objet et va être aussi utilisé comme étant un key pour chaque Point2DWritable généré.
    - P2dw : le Point2DWritable qui va être retourné à chaque appel de la méthode *getCurrentValue()*.
    - **Max** : un entier qui va contenir la valeur qui va être saisie par l'utilisateur pour indiquer combien de points vont être créé par chaque mapper.
  - Les Méthodes :
    - *initialize (InputSplit,TaskAttemptContext)* : comme préciser sur la Java Doc du RecordReader, cette méthode sera appelée une seule fois dans l'initialisation, on l'a utilisée pour récupérer le nombre maximum de points qui seront créer par chaque mappeur ce nombre qui sera taper par l'utilisateur donc on a utilisé le context donné dans les paramètres on utilisant la méthode getConfiguration(), plus get(«VARIABLE_NAME»).
    - *nextKeyValue()* : dans cette méthode on vérifie si on n'est pas encore arrivez au nombre maximum de points qu'on veut créer (Counter < Max) si la condition et vérifier on incrémente le conteur et on crée un nouveau Point2DWritable et on retourne TRUE, sinon en retourne FALSE

Exercice 3: Test du RandomPointInputFormat

Pour tester notre RandomPointInputFormat, on doit modifier notre main et spécifier que notre job utilisera notre classe RandomPointInputFormat, en faisant cela, les méthodes *getSplits()* et *createRecordReader()*de InputFormat qu'on a implémentés dans RandomInputFormat vont être appelées automatiquement. On modifie aussi les sorties de notre mapper dans le main pour dire que les OutPut Keys et Values seront des IntWritable et Point2DWritable respectivement.

Exercice 4: Calcul de Pi :

Le but de cet exercice et ce TP et d'être capable de calculer une valeur approximative de π par la méthode de Monte-Carlo : Cette méthode consiste à générer des points dans un plan a deux dimension, ces points doivent avoir leurs coordonnées comprises entre 0 et 1. La méthode dit que seulement un quart de ces point tombe dans le quart du cercle, cela dit, seulement quelques-uns de ces point tombent dans la superficie $\pi/4$. Et pour calculer π il suffit donc de calculer le nombre de points qui sont dans cette superficie et le divisé par le nombre des points générés.

Une description détaillée sur la méthode de Monte-Carlo pour le calcul de π est valable sur le lien :

http://therese.eveilleau.pagesperso-orange.fr/pages/truc_mat/textes/monte_carlo.htm

(Visité pour la dernière fois le 08/10/2016)

Dans notre classe TP5 le mapper prend en entrée un IntWritable comme Key and Point2DWritable comme value, et donne un IntWritable(Key) et Point2DWritable(Value) en sortie. La méthode *map()* va seulement ecrire les keys et valeurs reçues en entrée.

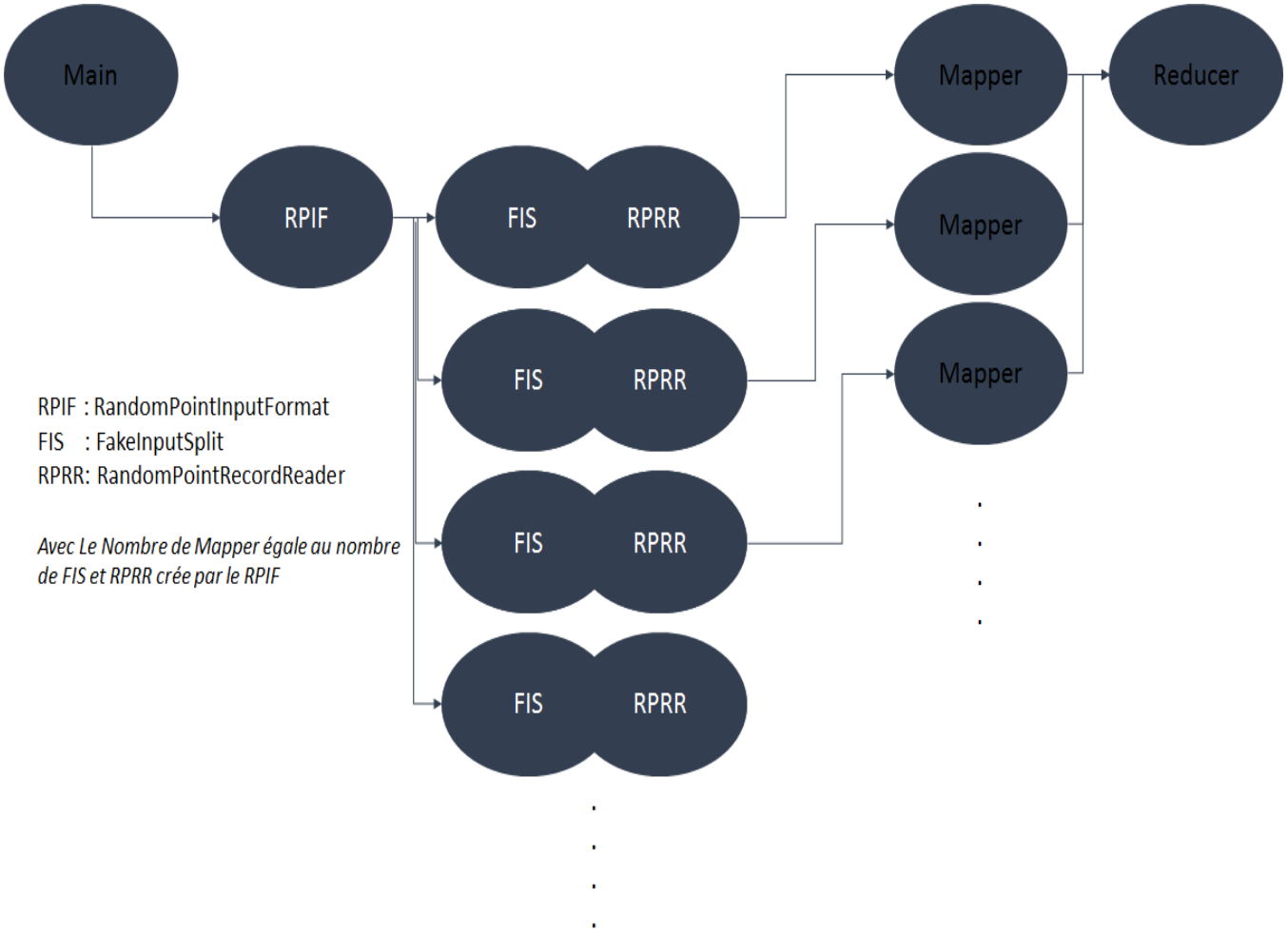
Pour notre Reducer il prend comme entrée les sorties des mapper (key : IntWritable, Value : Point2DWritable) et nous retourne en Sortie un IntWritable et un Text. La méthode *reduce()* va faire tout le traitement nécessaire pour le calcule de la valeur approchée de Pi, pour cela on a utilisé deux compteur :

- **Valid_Points** : qui va être utilisé pour calculer le nombre des points qui sont dans la zone qui représente un quart d'un cercle de rayon 1 : $(\pi/4)$
- **Nomber_Of_Points** : cela va être utilisé pour calculer le nombre des points générer par tout les mappers

Pour chaque key reçu par le reducer on balaye la liste des points qui ont le même key et pour chaque point on incrémente Number_Of_Points par 'un' et on teste si ce point appartient au quart du cercle, en d'autre termes ses coordonnes vérifies : $x^2 + y^2 < 1$, si oui , on incrémente la valeur de Valid_points par 'un'.

A la fin on aura tout ce dont on a besoin pour le calcule d'une valeur approximative de π qui va être égale à : (nombre des points valides/nombre des points générer)*4, et on écrit le résultat dans le fichier de sortie

Dans notre *main()* on a besoin de récupérer trois argument de l'utilisateur ; le nombre des points maximum par chaque mapper, le nombre des mappers et le fichier dans lequel on va ecrire notre résultat, en plus de ca on doit changer le OutPutValue et OutPutKey pour qu'ils soient adéquats avec ce qu'on a utilisés avant dans le mapper (IntWritable, Point2DWritable)



N.B : Le code a été soigneusement commenté pour expliquer son fonctionnement et son utilité. On s'est investis pour pouvoir le commenté de façon à le rendre le plus compréhensible que possible.