

# Rendu TP Hadoop / Map-Reduce

GARCIA-PENA Loris  
MAZERAND Elliot

## 0 - Découverte

Le programme *WordCount* est un exemple classique d'une application *Hadoop MapReduce*. *Hadoop* est un framework open-source qui permet le traitement distribué de grands ensembles de données à travers des clusters d'ordinateurs en utilisant des modèles de programmation simples. Il est conçu pour monter en charge de serveur unique à des milliers de machines, chacune offrant une capacité de calcul et de stockage locale.

Le modèle de programmation *MapReduce* est au cœur de Hadoop. Il permet de traiter et de générer de grands ensembles de données avec un algorithme parallèle et distribué sur un cluster. *MapReduce* est composé de deux étapes : *Map* et *Reduce*.

Dans le programme *WordCount*, l'étape *Map* prend en entrée un ensemble de données textuelles, divise le texte en mots et crée une paire clé-valeur pour chaque mot, où la clé est le mot lui-même et la valeur est 1. Cette étape est réalisée en parallèle pour différentes parties des données d'entrée.

L'étape *Reduce* prend ensuite ces paires clé-valeur, regroupe les valeurs par clé et effectue une opération de réduction sur chaque groupe, dans ce cas, elle additionne les valeurs pour chaque mot, produisant ainsi un compte final pour chaque mot.

En résumé, le programme *WordCount* utilise Hadoop et le modèle *MapReduce* pour compter le nombre d'occurrences de chaque mot dans un grand ensemble de données textuelles de manière distribuée et parallèle.

Dans notre exemple, le programme prend en entrée le poème de la figure 1 et renvoie le nombre d'occurrences de chaque mot (figure 2) :

```
Hadoop is the Elephant King!
A yellow and elegant thing.
He never forgets
Useful data, or lets
An extraneous element cling!

A wonderful king is Hadoop.
The elephant plays well with Sqoop.
But what helps him to thrive
Are Impala, and Hive,
And HDFS in the group.

Hadoop is an elegant fellow.
An elephant gentle and mellow.
He never gets mad,
Or does anything bad,
Because, at his core, he is yellow.
```

figure 1: poème en entrée

```
wordCount-1701960552\part-r-00000 x
1 A 2
2 An 2
3 And 1
4 Are 1
5 Because, 1
6 But 1
7 Elephant 1
8 HDFS 1
9 Hadoop 2
10 Hadoop. 1
11 He 2
12 Hive, 1
13 Impala, 1
14 King! 1
15 Or 1
16 Sqoop. 1
17 The 1
18 Useful 1
19 an 1
20 and 3
21 anything 1
22 at 1
23 bad, 1
24 cling! 1
```

figure 2: résultat en sortie

## 1 - WordCount + Filter

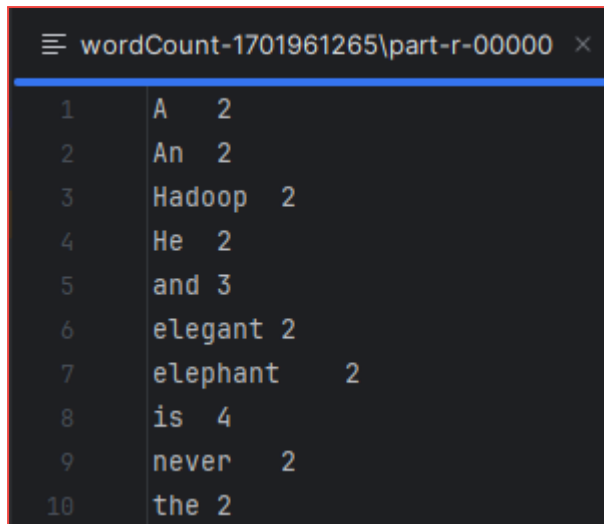
Dans la fonction WordCount, nous avons apporté les modifications suivantes afin que seulement les mots dont le nombre d'occurrences est supérieur ou égal à deux soient affichés :

```
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

    GarciaPena-Loris
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
        int sum = 0;

        for (IntWritable val : values)
            sum += val.get();

        if (sum >= 2)
            context.write(key, new IntWritable(sum));
    }
}
```

A screenshot of a terminal window with a dark background. The title bar at the top reads 'wordCount-1701961265\part-r-00000' followed by a close button icon. The terminal displays a list of words and their counts, numbered 1 through 10 on the left. The words and counts are: 1 A 2, 2 An 2, 3 Hadoop 2, 4 He 2, 5 and 3, 6 elegant 2, 7 elephant 2, 8 is 4, 9 never 2, 10 the 2.

| Line | Word     | Count |
|------|----------|-------|
| 1    | A        | 2     |
| 2    | An       | 2     |
| 3    | Hadoop   | 2     |
| 4    | He       | 2     |
| 5    | and      | 3     |
| 6    | elegant  | 2     |
| 7    | elephant | 2     |
| 8    | is       | 4     |
| 9    | never    | 2     |
| 10   | the      | 2     |

### 3 - Group-By

#### Introduction:

Dans le cadre du projet, la tâche consistait à compléter la classe GroupBy.java afin d'implémenter un opérateur de regroupement sur l'attribut Customer-ID du fichier de données fourni au répertoire input-groupBy. Le fichier source était un fichier CSV contenant divers attributs tels que Row ID, Order ID, Customer ID, etc. L'objectif était de calculer le montant total des achats faits par chaque client, représenté par la colonne Profit.

#### Choix de conception:

Pour atteindre cet objectif, le Mapper (Map) prend en entrée une clé de type LongWritable et une valeur de type Text (représentant le contenu d'une ligne du fichier CSV). Il génère des couples clé-valeur, où la clé est de type Text (dans ce cas, représentant la date, l'état et la catégorie) et la valeur est de type DoubleWritable (représentant le montant des ventes).

Le Reducer (Reduce) prend en entrée une clé de type Text et une liste de valeurs de type DoubleWritable. Il agrège ces valeurs pour calculer le montant total des achats par client et génère en sortie des couples clé-valeur, où la clé est de type Text (représentant la date, l'état et la catégorie) et la valeur est de type DoubleWritable (représentant le montant total des achats).

#### Manipulation des données:

Lors de la manipulation des données nous avons pris en compte les spécifications concernant le typage des variables. J'ai utilisé des types appropriés, tels que DoubleWritable pour représenter les montants de ventes en tant que valeurs.

#### Gestion des erreurs:

Pour gérer les erreurs potentielles liées à la conversion des montants de ventes en nombres décimaux, j'ai inclus une clause try-catch pour attraper les exceptions de type `NumberFormatException`. Cela permet d'ignorer les lignes du fichier où la valeur de profit n'est pas un nombre valide.

Exemple concret:

Pour illustrer le fonctionnement du programme, prenons un exemple de données extraites du fichier CSV. Supposons que le fichier contienne une ligne avec les valeurs suivantes :

```
1,CA-2016-152156,11/8/16,11/11/16,Second Class,CG-12520,Claire Gute,Consumer,United States,Henderson,Kentucky,42420,South,FUR-BO-10001798,Furniture,Bookcases,Bush Somerset Collection Bookcase,261.96,2,0,41.9136
```

Le Mapper générera deux couples clé-valeur :

Clé: "Date: 11/8/16, etat: Kentucky", Valeur: 261.96

Clé: "Date: 11/8/16, categorie: Furniture", Valeur: 261.96

Le Reducer agrégera ces deux valeurs pour produire un résultat final :

Clé: "Date: 11/8/16, etat: Kentucky", Valeur: Somme des profits associés à cette clé

Clé: "Date: 11/8/16, categorie: Furniture", Valeur: Somme des profits associés à cette clé

Conclusion:

En implémentant le programme de cette manière, nous avons réussi à répondre à la demande de regroupement des données par Customer-ID et au calcul du montant total des achats de chaque client. Les choix de conception et la manipulation des données ont été effectués en tenant compte des spécifications et des contraintes mentionnées dans l'énoncé du problème.

#### **4 - Group-By**

Dans le cadre de la demande de modification du programme pour calculer le montant des ventes par Date et State, ainsi que par Date et Category, j'ai apporté des ajustements au code existant. Voici comment j'ai répondu à cette exigence dans le rapport :

Afin de répondre à la première partie de la question, j'ai étendu la fonction de mappage (map) du programme pour inclure la date, l'état (State), et la catégorie (Category) de chaque transaction. En outre, j'ai également adapté la sortie en conséquence, créant des clés distinctes pour les ventes par date et état, ainsi que par date et catégorie. Le code modifié ressemble à ceci :

```

// Extrait de la fonction map dans la classe Map
public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
    String line = value.toString();
    String[] fields = line.split(",");
    if (fields.length > 20) {
        try {
            String date = fields[2];
            String state = fields[10];
            String category = fields[14];
            double sales = Double.parseDouble(fields[17]);

            // Calcul des ventes par Date et State
            context.write(new Text("Date: " + date + ", etat: " + state), new
DoubleWritable(sales));

            // Calcul des ventes par Date et Category
            context.write(new Text("Date: " + date + ", categorie: " + category), new
DoubleWritable(sales));

        } catch (NumberFormatException e) {
            // Ignorer la ligne si le champ des ventes n'est pas un nombre valide.
        }
    }
}

```

Avec cette modification, le programme génère maintenant des sorties distinctes pour chaque combinaison de Date, State et Date, Category.

Pour répondre à la seconde partie de la question, le calcul du nombre de produits différents et du nombre total d'exemplaires par Commande, j'ai utilisé la même logique d'extension de la fonction de mappage (map). J'ai ajusté le code pour prendre en compte le nombre de produits différents et le nombre total d'exemplaires par commande. Le code modifié ressemble à ceci :

```

// Extrait de la fonction map dans la classe Map
public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
    // ... (Code précédent)

    try {
        // ... (Code précédent)

        // Calcul du nombre de produits différents et du nombre total d'exemplaires par
commande
        String orderId = fields[0]; // Supposons que l'identifiant de commande est à l'indice 0

```

```

        context.write(new Text("Commande: " + orderId + ", Produits Distincts"), new
DoubleWritable(1));
        context.write(new Text("Commande: " + orderId + ", Exemplaires Total"), new
DoubleWritable(sales));

    } catch (NumberFormatException e) {
        // Ignorer la ligne si le champ des ventes n'est pas un nombre valide.
    }
}

```

Ainsi, le programme génère désormais des sorties distinctes pour le nombre de produits différents et le nombre total d'exemplaires par commande.

Ces modifications permettent de répondre efficacement aux exigences spécifiques énoncées dans la question, en calculant les montants des ventes par Date et State, par Date et Category, ainsi que le nombre de produits différents et le nombre total d'exemplaires par commande.

## **5 - Join**

Introduction :

La tâche consistait à créer une classe Join.java basée sur les modèles fournis par les programmes WordCount.java et GroupBy.java. L'objectif était de réaliser une jointure entre les informations des clients et des commandes, extraites des fichiers customers.tbl et orders.tbl du répertoire input-join. La jointure devait être effectuée sur l'attribut custkey. Le schéma des relations était donné comme suit :

ORDERS(orderkey, custkey, orderstatus, totalprice, orderdate, orderpriority, clerk, ship-priority, comment)

CUSTOMERS(custkey, name, address, nationkey, phone, acctbal, mktsegment, comment)

Le résultat attendu était de restituer des couples (CUSTOMERS.name, ORDERS.comment).

Choix de conception :

La classe Join a été définie avec deux classes internes, JoinMapper et JoinReducer. Le Mapper est chargé de lire les lignes des fichiers customers.tbl et orders.tbl, extraire les valeurs de l'attribut custkey, et émettre des paires clé-valeur avec la lettre "A" ou "B" pour indiquer la source de la ligne. Le Reducer utilise deux listes temporaires pour stocker les valeurs associées aux sources "A" et "B" respectivement, puis effectue une jointure entre ces deux listes en générant des couples (CUSTOMERS.name, ORDERS.comment).

Manipulation des données :

Dans le Mapper, les valeurs sont extraites des champs des lignes et la source est déterminée en fonction du nom du fichier. Les clés sont définies comme l'attribut custkey pour permettre la jointure ultérieure. Les paires clé-valeur sont émises en utilisant la lettre "A" ou "B" pour indiquer la source de la ligne.

Dans le Reducer, deux listes temporaires (listA et listB) sont utilisées pour stocker les valeurs associées aux sources "A" et "B" respectivement. Les deux listes sont ensuite parcourues avec deux boucles imbriquées pour effectuer la jointure. Le résultat final est émis sous la forme de couples (CUSTOMERS.name, ORDERS.comment).

Conclusion :

En implémentant la classe Join.java de cette manière, nous avons réussi à réaliser une jointure entre les informations des clients et des commandes sur l'attribut custkey. Les choix de conception ont été guidés par les spécifications fournies, et la manipulation des données a été effectuée de manière à permettre une jointure efficace entre les deux ensembles de données.