Université de Montpellier

Faculté des Scicences

30 Place E.Bataillon, 34095 Montpellier

Année 2023-2024

Rapport de Travaux Pratiques

TP n°4
Hadoop / Map-Reduce

par

Romain GALLERNE

Encadrant de TP : M. Federico Ulliana

Responsable du module : Mme Anne-Muriel Chifolleau

Table des matières

Exercice 1
WordCount + Filter

Tout au long de ce TP, l'entierté des résultats énoncé et des rendus de map/reduce seront joints dans l'archive de rendu.

Pour cette question il suffit, dans la fonction *reduce*, de modifier la ligne d'affichage en lui ajoutant une simple condition sur la valeur préalable. On passe donc de 59 mots dans le fichier de sortie à seulement 10 à l'aide de la modification suivante :

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

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

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

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

Exercice 3
Group-By

Dans cette exercice, nous avons du écrire les fonctions map et reduce nécessaire à la réalisation d'une requête Group-by et cela s'est divisé en 3 sous-problème chacun résolue par une des fonctions ci-dessous. Premièrement, il nous est nécessaire d'identifier les numéros de colonne sur lesquelles nous souhaitons réaliser notre requête group-by, nous avons pour cela écrit la fonction suivante qui nous servia par la suite :

```
private static int col_client = -1;
private static int col_profit = -1;

public void trouve_Col(String[] cols) {
    for(int i=0;i<cols.length;i++) {
        String[] cels = cols[i].split("\n");
        if(cels[0].equals("Customer ID")){
            col_client = i;
        } else if(cels[0].equals("Profit")){
            col_profit = i;
        }
    }
}
```

À présent, nous pouvons identifier les numéros de colonne de Customer ID et de Profit, nous allons donc pouvoir écrirer la mappage. Voici la fonction de mappage écrite, le fonctionnement de celle-ci est très simillaire à celle du map de Word-Count avec en plus l'appel à la fonction trouve-Col afin d'identifier dans les variables de classes les numéros de colonne client et profit, ceux-ci sont utilisés pour trier les colonnes et ne réaliser la requête que sur celles ou s'est nécessaire et ainsi créer les couples clés-valeurs uniquement sur ces deux colonnes, les autres ne nous intéressant pas.

```
public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    String line = value.toString();


    String[] cols = line.split(",");
    trouve_Col(cols); //Fonction permettant d'identifier les indices des colonnes qu'on souhaite groupBy

    // La ligne est vide : on s'arrête
    if (Arrays.equals(cols, emptyWords))
        return;

    double profit;

    try { //Pour palier à la première ligne contenant les titres des colonnes
        profit = Double.parseDouble(cols[col_profit]);
        context.write(new Text(cols[col_client]), new DoubleWritable(profit));
    } catch (NumberFormatException nfe) {}

}
```

Enfin, la fonction de Reduce est très similaire à celle de Word-Count à l'exception qu'on manipule ici des double et pas des int.

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

    @Override
    public void reduce(Text key, Iterable<DoubleWritable> values, Context context)
            throws IOException, InterruptedException {
        double sum = 0.0;

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

        //context.write(key, new IntWritable(sum));
        context.write(key, new DoubleWritable(sum));
    }
}
```

Exercice 4
Group-By

Dans un objectif de synthétisation, nous allons ici principalement aborder les requêtes concernant le comptage des produits par date/Catégorie et par date/State. En effet, les deux requêtes précédentes sont assez identiques à celles réalisés durant l'exercice 3. Commençons tout d'abord par notre fonction trouveCol qu'il a ici fallu modifier pour prendre en compte de nouvelles colonnes :

```
public void trouve_Col(String[] cols) {
    for(int i=0;i<cols.length;i++) {
        String[] cels = cols[i].split("\n");
        if(cels[0].equals("Customer ID")){
            col_client = i;
        }
        if(cels[0].equals("Profit")){
            col_profit = i;
        }
        if(cels[0].equals("State")){
            col_state = i;
        }
        if(cels[0].equals("Category")){
            col_categorie = i;
        }
        if(cels[0].equals("Order Date")){
            col_date = i;
        }
        if(cels[0].equals("Sales")){
            col_vente = i;
        }
        if(cels[0].equals("Product ID")){
            col_produit = i;
        }
        if(cels[0].equals("Quantity")){
            col_quantite = i;
        }
    }
}
```

Nous pouvons maintenant écrire notre fonction de mappage qui ressemble énormément à celle de l'exercice précédent si ce n'est que la clé devient la concaténation de la date et de la catégorie (respectivement avec l'état).

```java
public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    String line = value.toString();


    String[] cols = line.split(",");
    trouve_Col(cols); //Fonction permettant d'identifier les indices des colonnes qu'on souhaite groupBy

    // La ligne est vide : on s'arrête
    if (Arrays.equals(cols, emptyWords))
        return;

    String valeur;
    String cle;

    try { //Pour palier à la première ligne contenant les titres des colonnes
        cle = cols[col_date]+'|'+cols[col_categorie];
        valeur = cols[col_produit];
        context.write(new Text(cle), new Text(valeur));
    } catch (NumberFormatException nfe) {}

}
```

Le mappage a transmis une valeur textuel (ici, le code du produit) contrairement aux questions précédentes ou un Double était retourné. Il faut donc ici adapter le programme avec une List qui va stocker les produits déjà rencontrés afin de ne pas les compter deux fois, puisque on veut un distinct. Enfin on retournera le comptage des produits pour chaque clé.

```java
@Override
public void reduce(Text key, Iterable<Text> values, Context context)
        throws IOException, InterruptedException {
    List<String> prodDejaRencontre = new ArrayList<String>();
    double nbProd = 0;

    for (Text val : values) {
        String valString = val.toString();
        try {
            if(prodDejaRencontre.isEmpty() || !(prodDejaRencontre.contains(valString))) {
                prodDejaRencontre.add(valString);
                nbProd += 1;
            }
        } catch (NullPointerException npe) {
            System.out.print(valString);
            System.out.println(npe);
        }
    }

    //context.write(key, new IntWritable(sum));
    context.write(key, new DoubleWritable(nbProd));
}
```

Exercice 5
Join

Dans cette exercice, ils nous est demandé de traiter une requête Join. Pour cela l'idée à été de parcourir tour à tour les deux fichier tables .tbl et de stocker à chaque reprise dans deux tableaux ou listes les valeurs à l'indice "key" de, respectivement, chacune des deux tables.

Pour cette partie, au vu de la structure de donnée utilisé, notre habituelle fonction "trouveCol" n'est pas nécessaire, elle n'a donc pas été incluse. Cependant, on peut notre l'usage de la méthode ".getInputSplit" afin de connaître le fichier table qu'on utilise. Avec un simple test on peut alors savoir quelle fichier on est en train de traiter, ce qui est très pratique ici car les indices des attributs utiles ne sont pas identiques dans les deux tables.

```
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

        String line = value.toString();
        String[] cols = line.split("\\|");

        if (Arrays.equals(cols, emptyWords))
            return;

        String valeur;
        String cle;

        if(context.getInputSplit().toString().contains("orders.tbl")) {
            try {
                cle = cols[col_orderkey];
                valeur = cols[col_ordercomment];
                context.write(new Text(cle), new Text(valeur));
            } catch (NumberFormatException nfe) {}
        }

        else if (context.getInputSplit().toString().contains("customers.tbl")) {
            try {
                cle = cols[col_custkey];
                valeur = cols[col_custname];
                context.write(new Text(cle), new Text(valeur));
            } catch (NumberFormatException nfe) {}
        }

        else {throw new InterruptedException("Fichier tbl non-identifié");}
    }
}
```

La particularité de la fonction de réduction est la gestion des tableaux afin d'avoir toujours une place pour stocker le couple "clé/valeur" de la façon suivante : "valeur" dans la case numéro "clé" du tableau approprié. On doit pour cela effectuer une manipulation avec des tableaux temporaire afin de s'assurer que ceux-ci ont toujours assez de place pour accueillir une valeur à la case "clé" qui peut être arbitrairement grande. Dans le cas de clé non-numérique, on pourrait s'aider d'une fonction de hashage, mais ce n'est pas nécessaire ici.

```
@Override
public void reduce(Text key, Iterable<Text> values, Context context)
    throws IOException, InterruptedException {

    Integer intKey = Integer.parseInt(key.toString());
    for (Text val : values) {
        String valString = val.toString();
        if(valeursName==null || valeursName.size() < intKey) {
            ArrayList<String> tempValName =  Arrays.copyOf(valeursName,intKey);;
            ArrayList<String> tempValComment = new ArrayList<>(intKey);
            tempValName.addAll(valeursName);
            tempValComment.addAll(valeursComment);
            valeursName = (ArrayList<String>)tempValName.clone();
            valeursComment = (ArrayList<String>)tempValComment.clone();
        }
        if(valeursName.get(intKey) == null) {
            valeursComment.add(intKey,valString);
        }
        else {
            valeursName.add(intKey,valString);
        }
    }

    for(int i=0;i<valeursName.size();i++) {
        context.write(new Text(valeursName.get(i)), new Text(valeursComment.get(i)));
    }

}
```