Hadoop / Map-Reduce
Entrepôts de données et Big Data

Chamy Kaci et Oussama Andre

Décembre 2022

Table des matières

1 WordCount

1.1 WordCount

Le programme lit les fichiers contenus dans le répertoire "input-wordCount/", puis effectue un comptage des mots et stock le resultats dans le repertoire "output/" :
Il va dans un premier temps faire un map. Map consiste a produire des couples clé-valeur. Dans notre cas, il va attribuer a chaque mot la valeur 1. Ensuite, il va faire un reduce qui dans notre cas, va reduire le nombre de couple clé-valeur en unifiant les couples qui ont la même clé et en incrementant de 1 sa valeur si c'est le cas.
Par exemple pour le fichier "hadoopPoem.sh", on obtient le resultat suivant :
Map : 72 couples clé-valeur.
Reduce : 59 couples clé-valeur.

1.2 WordCount + Filter

Il suffit de reprendre l'exemple d'origine et de seulement écrire le nombre d'occurrences calculé s'il est supérieur à 2.

```
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));
    }
```

Listing 1 – Filter

2 Group-by

2.1 Group-by $\langle ID - Costumer \rangle$

Avec le map-reduce le groupe-by est assez simple. Le REDUCE s'opère sur un ensemble de valeurs qui partagent une même clé. Dans ce cas notre clé sera la colonne ID-Costumer sur laquelle s'effectuera le groupe-by, et la valeur c'est la colonne profit.

Dans les deux listings suivants, on trouve le code des méthodes "map" et "reduce" pour cette partie.

```
    public final static String emptywords[] = {" "};
    @Override
    public void map(LongWritable key, Text value, Context context)
    throws IOException ,InterruptedException {
      String line = value.toString().trim();
      String[] columns = line.split(",");

```

```
      if (Arrays.equals(columns, emptywords)){
        return;}
      double profit;
      try {
      profit = Double.parseDouble(columns[20]);
      }catch(Exception e ){return;}

      context.write(new Text(columns[5]), new DoubleWritable(profit
    ));
    }
```

Listing 2 – Map

```
  @Override
   public void reduce(Text key, Iterable<DoubleWritable> values,
   Context context)
       throws IOException, InterruptedException {
     double sum = 0;

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

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

Listing 3 – Reduce

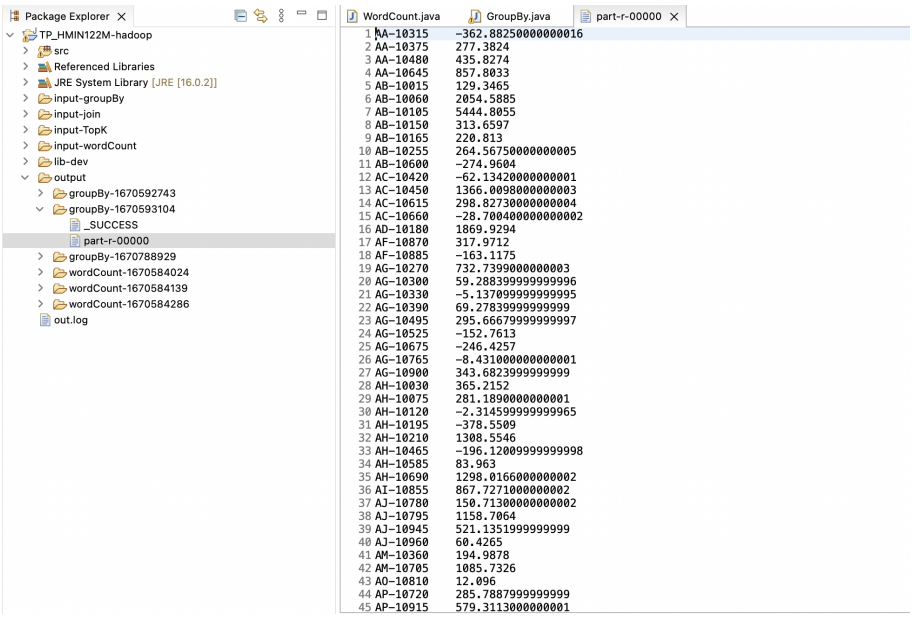

FIGURE 1 – Montant total des achats par client

2.2 Group - by $\langle DATE - STATE \rangle$

Pour gérer plusieurs groupes by, on a concaténé les contenus "Ordre Date" et "State" pour créer la clé (clé composite). La valeur est "Sales". La fonction Reduce est la même que pour l'exercice 3.

```
public static class Map extends Mapper<LongWritable, Text, Text,
    DoubleWritable> {
    private final static String emptyWords[] = { "" };

    @Override
    public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
      String[] columns = value.toString().split(",");
      if (Arrays.equals(columns, emptyWords)) {
        return;
      }
      // columns[2] <-> order date, columns[10] <-> state,columns
    [17]<->Sales;
      try {
        String cle = columns[2]+" "+columns[10];
        double valeur = Double.parseDouble(columns[17]);
        context.write(new Text(cle), new DoubleWritable(valeur));
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
  }
```

Listing 4 – Map2

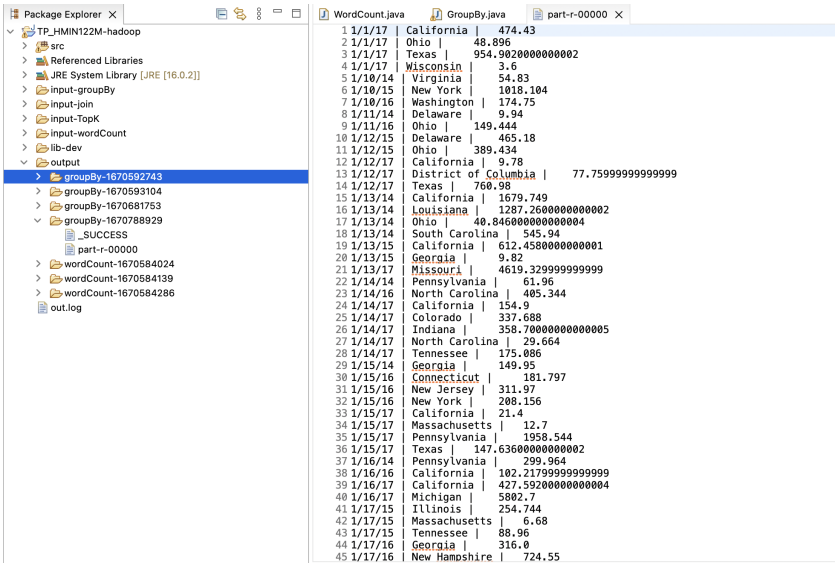

FIGURE 2 – Montant des ventes par Date et State

2.3 Group - by $\langle DATE - CATEGORY \rangle$

Même principe que pour le Group - by $\langle DATE - STATE \rangle$.

```java
public static class Map extends Mapper<LongWritable, Text, Text,
    DoubleWritable> {
    private final static String emptyWords[] = { "" };
    @Override

    public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
      //Calculer le montant des ventes par Date et Category.

     String[] columns = value.toString().split(",");
     if (Arrays.equals(tokens, emptyWords)) {
        return;}
      // columns[2] <-> Order Date, columns[14] <->  Category,
    columns[17] <-> Sales;
      try {
        String cle = columns[2]+" "+columns[14];
        double valeur = Double.parseDouble(columns[17]);
        context.write(new Text(cle), new DoubleWritable(valeur));
      } catch (Exception e) {
        e.printStackTrace();
             }
        }
    }
```

Listing 5 – Map3

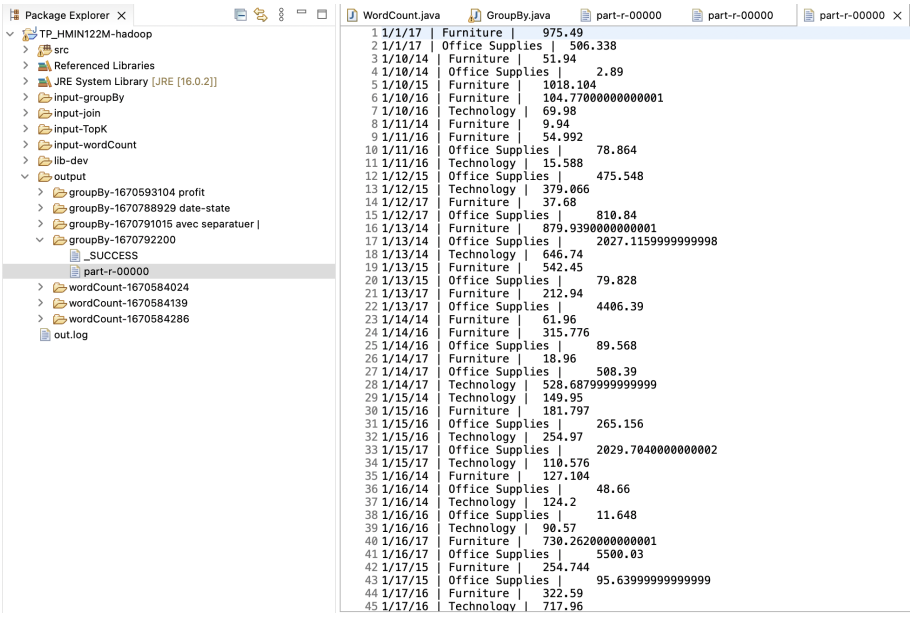

FIGURE 3 – Montant des ventes par Date et Category

2.4 Le nombre de produits différents (distincts) achètes et le nombre total d'exemplaires

Ici le MAP nous renvoie la clé qui est Ordre ID colonne sur qui le groupe-by s'effectue, la valeur est Quantity. Pour le REDUCE on doit compter le nombre de valeur pour avoir le nombre de produits différents pour satisfaire la condition (distinct).

```
public static class Map extends Mapper<LongWritable, Text, Text,
    IntWritable> {



  public final static String emptywords[] = {" "};
  public void map(LongWritable key, Text value, Context context
    ) throws IOException ,InterruptedException {
       String line = value.toString().trim();
       String[] columns = line.split(",");

       if (key.equals(new LongWritable(0)) || Arrays.equals(
    columns, emptywords)){
         return;
       }


       context.write(new Text(columns[1]), new IntWritable(
    Integer.parseInt(columns[columns.length-3] )));

  }
}

```

Listing 6 – MAP $\langle OrderID - Quantity \rangle$

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

    @Override
    public void reduce(Text key, Iterable<IntWritable> values,
    Context context)
        throws IOException, InterruptedException {
      int sum = 0;
      int count =0;
      for (IntWritable val : values) {
        sum += val.get();
        count++;
    }
      context.write(key, new Text(count+ "\t"+sum));
    }
  }
```

Listing 7 – REDUCE $\langle OrderID - Quantity \rangle$

Résultat Obtenue

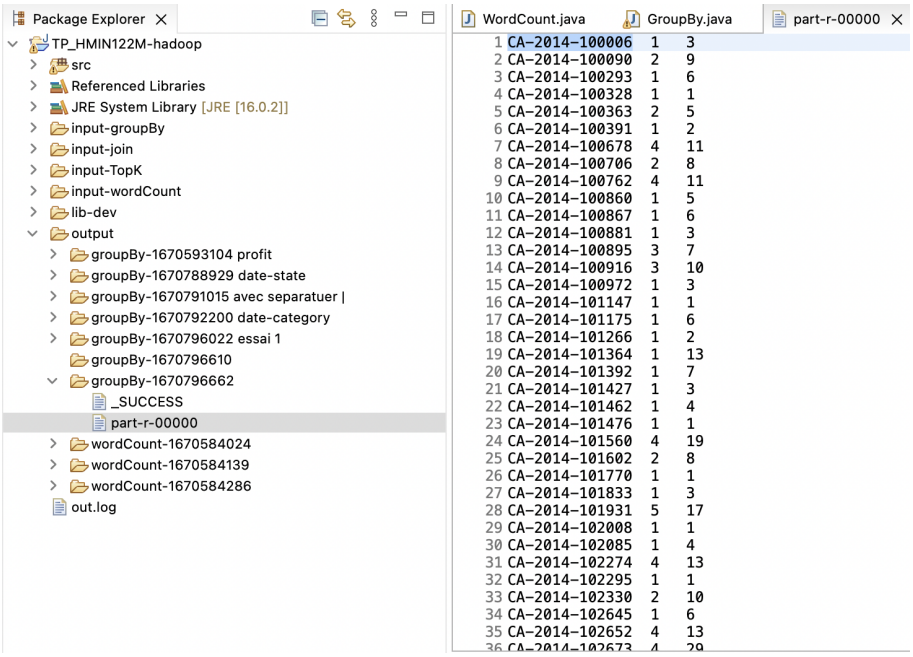

FIGURE 4 – $\langle OrderID - Count(distinct) - Quantity \rangle$

3 Join

Le programme prend en entrée les fichiers présents dans le dossier `input-join`.Pour différencier entre les données des ORDERS et celles de CUSTOMERS dans le MAP, on s'appuie sur le nombre de colonnes. Les strings des valeurs de CUSTOMERS sont précédées par un "c,&" pour faciliter la différenciation dans le REDUCE.

Pour réaliser la jointure il faut à l'avance recopier dans un tableau temporaire les valeurs de l'itérateur values dans la méthode REDUCE, puis effectuer le parcours avec deux "for" imbriqués sur ce tableau temporaire. Pour une question d'efficacité, on a choisi de mettre les valeurs dans deux listes différentes.

```
1  public static class Map extends Mapper<LongWritable, Text, Text,
     Text> {
2
3
4      public final static String emptywords[] = {" "};
5      public void map(LongWritable key, Text value, Context context
   ) throws IOException ,InterruptedException {
6          String line = value.toString().trim();
7          String[] columns = line.split("\\|");
```

```
        int index_cl  = -1;
        int index_valeur = -1;
        String id_table ;


        if (key.equals(new LongWritable(0)) || Arrays.equals(columns, emptywords)){
          return;
        }


         if(columns.length == 9) {
            index_cl  = 1;
            index_valeur = 8;
            id_table = "";
            }

          else if(columns.length == 8) {
           index_cl  = 0;
           index_valeur = 1;
           id_table = "c,&";
           }
        else {
          System.out.println("Column length is "+ columns.length);
          return;
        }


        context.write(new Text(columns[index_cl ]), new Text(id_table + columns[index_valeur] ));

    }
}
```

Listing 8 – MAP - JOIN

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

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

    ArrayList<Text> customers = new ArrayList<Text>();
    ArrayList<Text> orders = new ArrayList<Text>();

    for (Text val : values) {
      //System.out.println(val + "shamy");
      String[] txt = val.toString().split(",&");
      //System.out.println(txt + "shamy");
      if(txt.length == 2) {
        customers.add(new Text(txt[1]));
      }
```

```
        else {orders.add(val);}
        }



      for(Text customer : customers) {
        for(Text order : orders) {

            context.write(customer, order);
        }
      }
    }
  }

```

Listing 9 – REDUCE - JOIN

Résultat obtenu

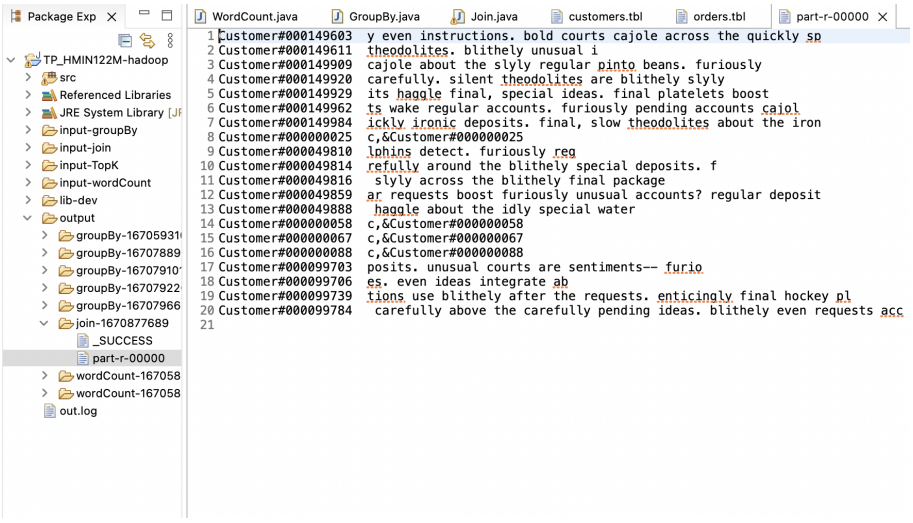

FIGURE 5 – Join