

Hadoop / Map-Reduce

Entrepôts de données et Big Data

Chamy Kaci et Oussama Andre

Décembre 2022



Table des matières

1	WordCount	3
1.1	WordCount	3
1.2	WordCount + Filter	3
2	Group-by	3
2.1	Group-by $\langle ID - Costumer \rangle$	3
2.2	Group - by $\langle DATE - STATE \rangle$	5
2.3	Group - by $\langle DATE - CATEGORY \rangle$	6
2.4	Le nombre de produits différents (distincts) achètes et le nombre total d'exemplaires	7
3	Join	8

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 resultat 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.

```
1 public void reduce(Text key, Iterable<IntWritable> values, Context
   context) throws IOException, InterruptedException {
2     int sum = 0;
3     for (IntWritable val : values)
4         sum += val.get();
5     if(sum>=2)
6         context.write(key, new IntWritable(sum));
7 }
```

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.

```
1 public final static String emptywords[] = {" "};
2 @Override
3 public void map(LongWritable key, Text value, Context context)
   throws IOException, InterruptedException {
4     String line = value.toString().trim();
5     String[] columns = line.split(",");
6 }
```

```

7      if (Arrays.equals(columns, emptywords)){
8          return;}
9      double profit;
10     try {
11         profit = Double.parseDouble(columns[20]);
12     }catch(Exception e ){return;}
13
14     context.write(new Text(columns[5]), new DoubleWritable(profit
15     ));
16 }

```

Listing 2 – Map

```

1  @Override
2  public void reduce(Text key, Iterable<DoubleWritable> values,
3  Context context)
4      throws IOException, InterruptedException {
5      double sum = 0;
6
7      for (DoubleWritable val : values)
8          sum += val.get();
9
10     context.write(key, new DoubleWritable(sum));
11 }

```

Listing 3 – Reduce

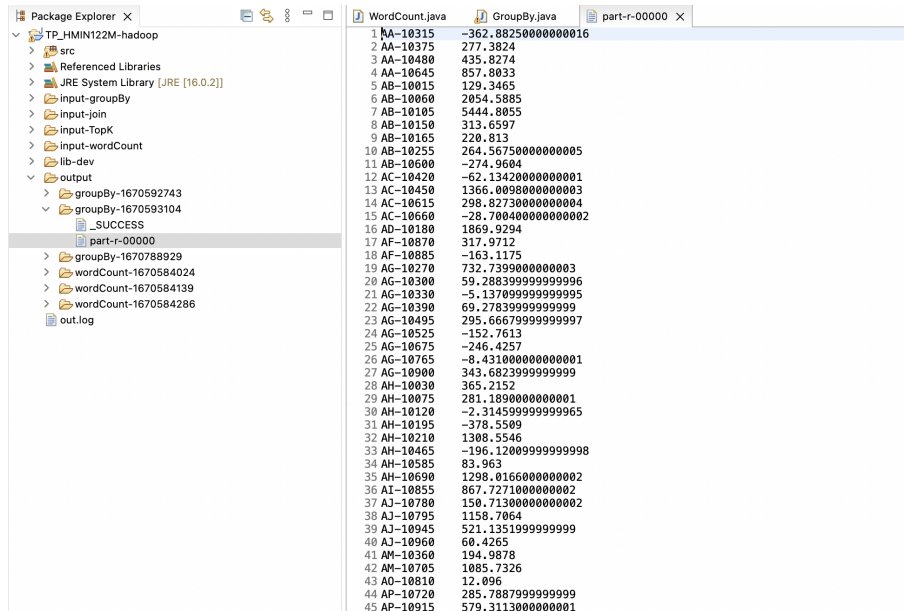


FIGURE 1 – Montant total des achats par client

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.

```

1 public static class Map extends Mapper<LongWritable, Text, Text,
2   DoubleWritable> {
3   private final static String emptyWords[] = { "" };
4
5   @Override
6   public void map(LongWritable key, Text value, Context context)
7     throws IOException, InterruptedException {
8     String[] columns = value.toString().split(",");
9     if (Arrays.equals(columns, emptyWords)) {
10      return;
11    }
12    // columns[2] <-> order date, columns[10] <-> state, columns
13    [17]<->Sales;
14    try {
15      String cle = columns[2]+" "+columns[10];
16      double valeur = Double.parseDouble(columns[17]);
17      context.write(new Text(cle), new DoubleWritable(valeur));
18    } catch (Exception e) {
19      e.printStackTrace();
20    }
21  }
22 }

```

Listing 4 – Map2

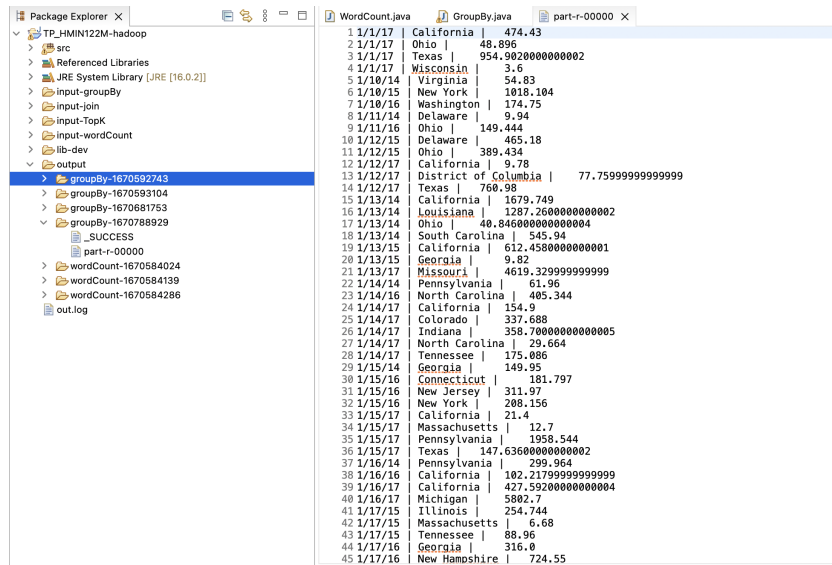


FIGURE 2 – Montant des ventes par Date et State

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

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

```

1 public static class Map extends Mapper<LongWritable, Text, Text,
    DoubleWritable> {
2     private final static String emptyWords[] = { "" };
3     @Override
4
5     public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
6         //Calculer le montant des ventes par Date et Category.
7
8         String[] columns = value.toString().split(",");
9         if (Arrays.equals(tokens, emptyWords)) {
10             return;
11             // columns[2] <-> Order Date, columns[14] <-> Category,
            columns[17] <-> Sales;
12             try {
13                 String cle = columns[2]+" "+columns[14];
14                 double valeur = Double.parseDouble(columns[17]);
15                 context.write(new Text(cle), new DoubleWritable(valeur));
16             } catch (Exception e) {
17                 e.printStackTrace();
18             }
19         }
20     }

```

Listing 5 – Map3

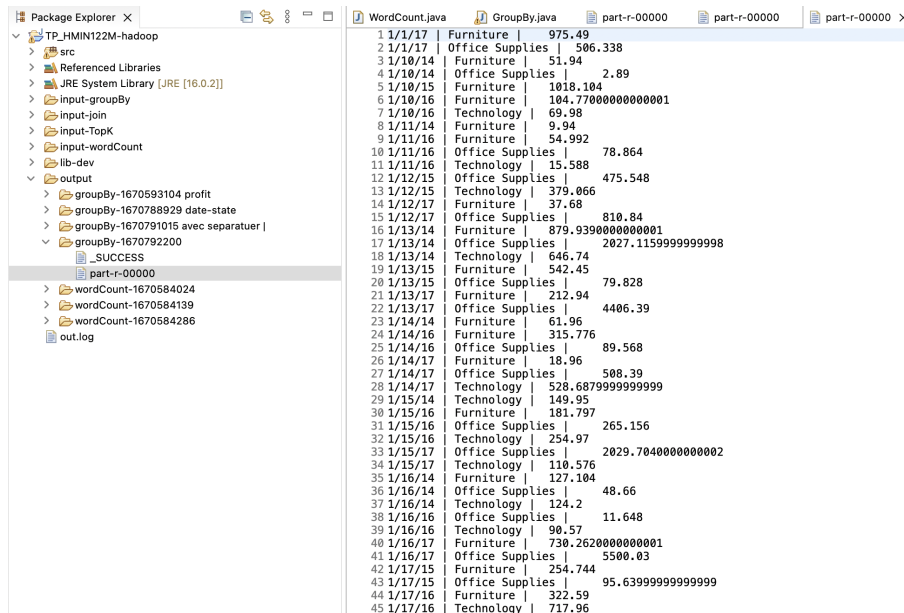


FIGURE 3 – Montant des ventes par Date et Category

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

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

```
1 public static class Map extends Mapper<LongWritable, Text, Text,
2     IntWritable> {
3
4
5     public final static String emptywords[] = {" "};
6     public void map(LongWritable key, Text value, Context context
7 ) throws IOException, InterruptedException {
8         String line = value.toString().trim();
9         String[] columns = line.split(",");
10
11         if (key.equals(new LongWritable(0)) || Arrays.equals(
12             columns, emptywords)){
13             return;
14         }
15
16         context.write(new Text(columns[1]), new IntWritable(
17             Integer.parseInt(columns[columns.length-3] )));
18     }
19 }
```

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

```
1 public static class Reduce extends Reducer<Text, IntWritable, Text,
2     Text> {
3
4     @Override
5     public void reduce(Text key, Iterable<IntWritable> values,
6         Context context)
7         throws IOException, InterruptedException {
8         int sum = 0;
9         int count = 0;
10        for (IntWritable val : values) {
11            sum += val.get();
12            count++;
13        }
14        context.write(key, new Text(count+ "\t"+sum));
15    }
16 }
```

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

Résultat Obtenue

Line	Category	Count
1	CA-2014-100006	1
2	CA-2014-100090	2
3	CA-2014-100293	1
4	CA-2014-100328	1
5	CA-2014-100363	2
6	CA-2014-100391	1
7	CA-2014-100678	4
8	CA-2014-100706	2
9	CA-2014-100762	4
10	CA-2014-100860	1
11	CA-2014-100867	1
12	CA-2014-100881	1
13	CA-2014-100895	3
14	CA-2014-100916	3
15	CA-2014-100972	1
16	CA-2014-101147	1
17	CA-2014-101175	1
18	CA-2014-101266	1
19	CA-2014-101364	1
20	CA-2014-101392	1
21	CA-2014-101427	1
22	CA-2014-101462	1
23	CA-2014-101476	1
24	CA-2014-101560	4
25	CA-2014-101602	2
26	CA-2014-101770	1
27	CA-2014-101833	1
28	CA-2014-101931	5
29	CA-2014-102008	1
30	CA-2014-102085	1
31	CA-2014-102274	4
32	CA-2014-102295	1
33	CA-2014-102330	2
34	CA-2014-102645	1
35	CA-2014-102652	4
36	CA-2014-102673	4

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,
2     Text> {
3
4     public final static String emptywords[] = {" "};
5     public void map(LongWritable key, Text value, Context context
6     ) throws IOException, InterruptedException {
7         String line = value.toString().trim();
8         String[] columns = line.split("\\|");
9     }
10 }

```



```

8      int index_cl = -1;
9      int index_valeur = -1;
10     String id_table ;
11
12
13     if (key.equals(new LongWritable(0)) || Arrays.equals(
columns, emptywords)){
14         return;
15     }
16
17
18     if(columns.length == 9) {
19         index_cl = 1;
20         index_valeur = 8;
21         id_table = "";
22     }
23
24     else if(columns.length == 8) {
25         index_cl = 0;
26         index_valeur = 1;
27         id_table = "c,&";
28     }
29     else {
30         System.out.println("Column length is "+ columns.length)
;
31         return;
32     }
33
34
35     context.write(new Text(columns[index_cl ]), new Text(
id_table + columns[index_valeur] ));
36
37 }
38 }
39
40
41

```

Listing 8 – MAP - JOIN

```

1  public static class Reduce extends Reducer<Text, Text, Text, Text
> {
2
3      @Override
4      public void reduce(Text key, Iterable<Text> values, Context
context)
5          throws IOException, InterruptedException {
6
7          ArrayList<Text> customers = new ArrayList<Text>();
8          ArrayList<Text> orders = new ArrayList<Text>();
9
10         for (Text val : values) {
11             //System.out.println(val + "shamy");
12             String[] txt = val.toString().split(",&");
13             //System.out.println(txt + "shamy");
14             if(txt.length == 2) {
15                 customers.add(new Text(txt[1]));
16             }

```

```

17     else {orders.add(val);}
18     }
19
20
21
22     for(Text customer : customers) {
23         for(Text order : orders) {
24
25             context.write(customer, order);
26         }
27     }
28 }
29 }
30

```

Listing 9 – REDUCE - JOIN

Résultat obtenu

```

WordCount.java  GroupBy.java  Join.java  customers.tbl  orders.tbl  part-r-00000  x
1 Customer#000149603 y even instructions. bold courts cajole across the quickly sp
2 Customer#000149611 theodolites. blithely unusual i
3 Customer#000149909 cajole about the slyly regular pinto beans. furiously
4 Customer#000149920 carefully. silent theodolites are blithely slyly
5 Customer#000149929 its haggle final, special ideas. final platelets boost
6 Customer#000149962 ts wake regular accounts. furiously pending accounts cajol
7 Customer#000149984 ickly ironic deposits. final, slow theodolites about the iron
8 Customer#000000025 c,&Customer#000000025
9 Customer#000049810 lohins detect. furiously reg
10 Customer#000049814 refully around the blithely special deposits. f
11 Customer#000049816 slyly across the blithely final package
12 Customer#000049859 ar requests boost furiously unusual accounts? regular deposit
13 Customer#000049888 haggle about the idly special water
14 Customer#000000058 c,&Customer#000000058
15 Customer#000000067 c,&Customer#000000067
16 Customer#000000088 posits. unusual courts are sentiments-- furio
17 Customer#000099703 es. even ideas integrate ab
18 Customer#000099706 tions use blithely after the requests. enticingly final hockey pl
19 Customer#000099739 carefully above the carefully pending ideas. blithely even requests acc
20 Customer#000099784
21

```

FIGURE 5 – Join