

# NoSQL 2015

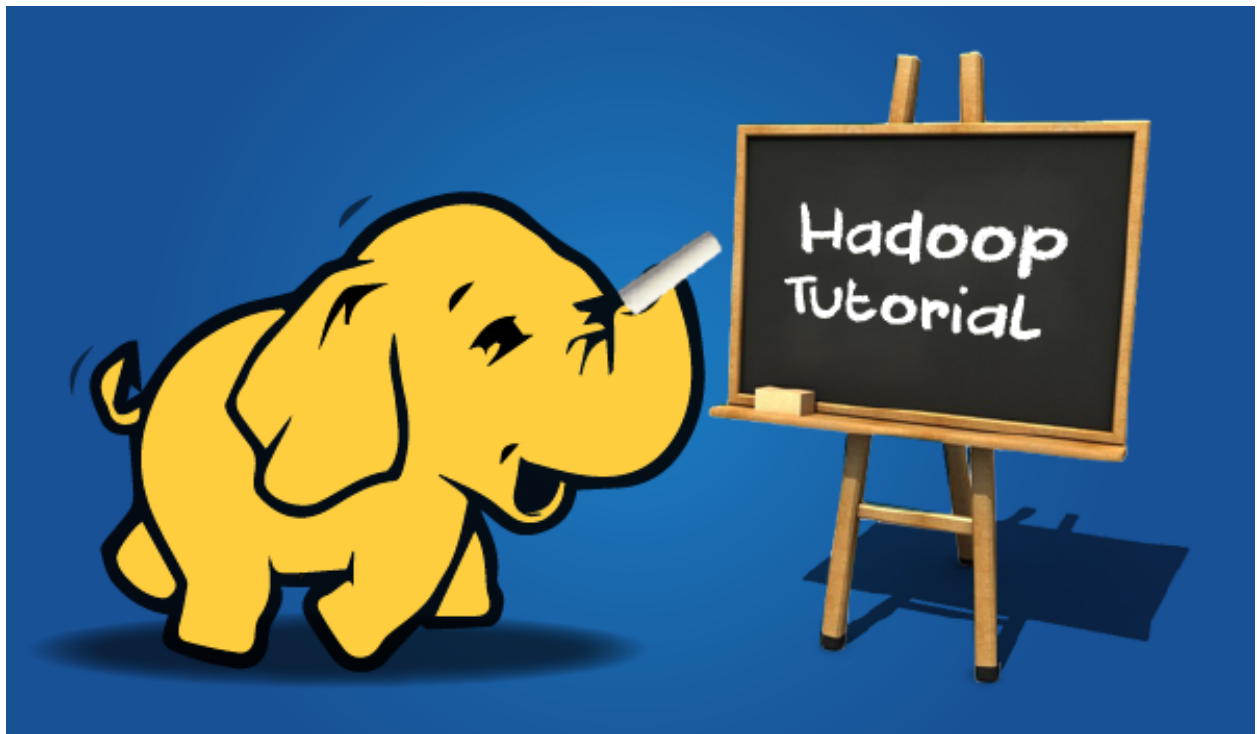
## TD 2

Professeurs

Houda Chabbi Drissi

et

Benoît Perroud



Assistant : Christophe Bovigny

## List of Frequently Asked Questions

<a href="#">Map Reduce</a> . . . . .	3
<a href="#">HIVE</a> . . . . .	5
<a href="#">PIG</a> . . . . .	6
<a href="#">Spark</a> . . . . .	7

## Map Reduce

### Problème

Le but de cette partie est de comprendre la résolution d'un problème simple au moyen de l'algorithme map/reduce. Ici nous avons un fichier contenant des livres et nous voudrions connaître combien de livres sont publiés par année.

Une classe java BookXMapper est définie comme ceci :

```
public class BookXMapper extends MapReduceBase implements
    org.apache.hadoop.mapred.Mapper<LongWritable, Text, Text, IntWritable>
{
    private final static IntWritable one = new IntWritable(1);

    public void map(LongWritable _key, Text value,
        OutputCollector<Text, IntWritable> output, Reporter reporter)
        throws IOException {

        String TempString = value.toString();
        String[] SingleBookData = TempString.split("\\;\\");
        output.collect(new Text(SingleBookData[3]), one);

        // TO COMMENT
        //
        //
        //
        //
        //
        //
    }
}
```

#####  
Solution :

Chaque ligne du fichier est découpée à chaque délimiteur ";" afin d'avoir un tableau de type "String". Le quatrième élément du tableau est le champ "Year-of-Publication", lequel est la clé du mapper.

#####  
Ensuite une method reduce est crée : expliquez-la :

```
public class BookXReducer extends MapReduceBase implements org.apache.hadoop.mapred.
    Reducer<Text, IntWritable, Text, IntWritable> {

    Override
    public void reduce(Text _key,
        Iterator<IntWritable> values,
        OutputCollector<Text,IntWritable> output,
        Reporter reporter)
        throws IOException {
        Text key = _key;
        int frequencyForYear = 0;
        while (values.hasNext()) {
            // replace ValueType with the real type of your value
            IntWritable value = (IntWritable) values.next();
            frequencyForYear += value.get();
        }
    }
}
```

```

    }
    output.collect(key, new IntWritable(frequencyForYear));
}
}

```

#####

Solution :

La méthode reduce prend en paramètres : key et Iterator<IntWritable> values (valeurs groupées pour chaque key). Pour notre programme, nous utilisons à nouveau la clé de mapper comme l'output du reducer et ajoutons chaque valeur de la liste. (Output de mapper est un new IntWritable(1)). Nous ajoutons toutes les occurrences de new IntWritable(1) afin d'avoir le compte de livres publiés pour chaque clé (année de publication)

#####

Finalemnt une dernière classe BookXDriver est ajoutée et servira comme main program afin de lancer le job MapReduce.

```

public class BookXDriver {
    public static void main(String[] args) {
        JobClient client = new JobClient();
        JobConf conf = new JobConf(com.orzota.bookx.BookXDriver.class);

        // Configurations for Job set in this variable
        JobConf conf = new JobConf(com.orzota.bookx.BookXDriver.class);
        // Name of the Job
        conf.setJobName("BookCrossing1.0");
        // Data type of Output Key and Value
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        // Setting the Mapper and Reducer Class
        conf.setMapperClass(com.orzota.bookx.BookXMapper.class);
        conf.setReducerClass(com.orzota.bookx.BookXReducer.class);
        // Formats of the Data Type of Input and output
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);

        // Specify input and output DIRECTORIES (not files)
        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        client.setConf(conf);

        try {
            // Running the job with Configurations set in the conf.
            JobClient.runJob(conf);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

Maintenant nous allons lancer ce programme sur le cluster DAPLAB :

```

##### Create directory local MapReduce 2 #####
cbovigny@daplab-wn-03:~$ mkdir MapReduceTP2 ; cd MapReduceTP2
#####
##### Copy to Local /shared/tp2/StepByStepMRGuide_BookCrossing #####
# A vous de jouer :)
#####

```

```
#####
cbovigny@daplab-wn-03:~$ cd StepByStepMRGuide_BookCrossing/
#####

#####Compilez le projet #####
cbovigny@daplab-wn-03:~/MapReduceTP2/StepByStepMRGuide_BookCrossing$ ant
#####

#####
# Entrez dans le repertoire dist et lancez la commande
hadoop jar BookCrossing.jar /shared/tp2/BX-Books.csv /user/cbovigny/MapReduceTP2/
output
#####
```

Regardez l’output de votre map/reduce et expliquez-le

## HIVE

Maintenant nous allons résoudre le même problème que précédemment, mais cette fois en utilisant la technologie HIVE.

Notre fichier BX-Books.csv n’est pas clean pour l’utiliser dans hive, c’est pourquoi la commande sed s’impose :

```
sed 's/&;/ AND /g' BX-Books.csv | sed -e '1d' | sed 's/;/|sed's/" /"; "/g' > BX-
BooksCorrected_CHRISTOPHE.txt
```

Cette commande élimine les délimiteurs ";" (semicolon) et les remplace par des "\$\$\$", et le motif "&;" est remplacé par "AND". Cette commande élimine aussi le header du fichier. (si on ne l’élimine pas Hive le considère comme data). Ce fichier corrigé se trouve dans /shared/tp2/BX-BooksCorrected.txt du système de fichier HDFS.

Maintenant nous allons lancer hive et créer une table BXDataSet.

A l’intérieur veuillez mettre ISBN (STRING), BookTitle (STRING), BookAuthor (STRING), YearOfPublication (STRING), Publisher (STRING), ImageURLS (STRING), ImageURLM (STRING) et ImageURLL (STRING) comme champ

```
##### Create Table BXDataSet #####
CREATE TABLE IF NOT EXISTS BXDataSet (
TO COMPLETE
)
COMMENT 'BX-Books Table'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\073'
STORED AS TEXTFILE;
```

#####  
Solution :

```
##### Create Table BXDataSet #####
CREATE TABLE IF NOT EXISTS BXDataSet (
ISBN STRING,
BookTitle STRING,
BookAuthor STRING,
YearOfPublication STRING,
Publisher STRING,
ImageURLS STRING,
ImageURLM STRING,
ImageURLL STRING
```

```
)
COMMENT 'BX-Books Table'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\073'
STORED AS TEXTFILE;
```

#####

Maintenant nous avons deux solutions pour créer la table, la première interactivement et la seconde en une ligne de commande en mettant les instructions dans un fichier createtablehive.sql

##### First Solution #####

```
cbovigny@daplab-wn-03:~$ hive
hive> CREATE TABLE IF NOT EXISTS BXDataSet
> (TO COMPLETE,
> ... TO COMPLETE)
> COMMENT BX -Books Table
> ROW FORMAT DELIMITED
> FIELDS TERMINATED BY ;
> STORED AS TEXTFILE;
```

OK

##### Second Solution #####

```
cbovigny@daplab-wn-03:~$ hive -v -f createtablehive.sql
```

#####

Ensuite il faut remplir la table bxdataset en utilisant le fichier BX-BooksCorrected\_CHRISTOPHE.txt se situant dans /shared/tp2/. Tout d'abord copiez ce fichier dans votre home hdfs et ensuite lancez la commande ci-dessous.

##### First Solution #####

```
cbovigny@daplab-wn-03:~$ hive
hive> LOAD DATA INPATH '/user/cbovigny/BX-BooksCorrected_CHRISTOPHE.txt' OVERWRITE
INTO TABLE BXDataSet;
```

#####

Afin d'obtenir le même résultat que pour le MapReduce, veuillez compléter la requête dans hive.

##### First Solution #####

```
cbovigny@daplab-wn-03:~$ hive
hive> select yearofpublication, (TO COMPLETE), from bxdataset group by
yearofpublication;
```

#####

#####

Solution :

##### First Solution #####

```
cbovigny@daplab-wn-03:~$ hive
hive> select yearofpublication, count(booktitle), from bxdataset group by
yearofpublication;
```

#####

#####

Voilà votre requête dans hive donne exactement la même solution que votre MapReduce en java. 25 lignes se sont transformées en 3 lignes :)

## PIG

Maintenant nous allons effectuer la même chose mais en utilisant pig.

Nous allons lancer pig sur daplab, loader les données dans une variable BookXRecords, ensuite les grouper par année de publication et finalement concaténer le nombre de publications par année.

```
##### Pig Execution #####
cbovigny@daplab-wn-03:~$ pig
# Put the bookpig.txt /shared/tp2/bookpig.txt into /user/yourlogin/
grunt> BookXRecords = LOAD '/user/yourlogin/bookpig.txt'
>> USING PigStorage(';') AS (ISBN:chararray,BookTitle:chararray,
>> #TOCOMPLETE
>> #TOCOMPLETE );
# create a variable GroupByYear#####
grunt> GroupByYear = GROUP BookXRecords BY #TOCOMPLETE;
# create a variable CountByYear#####
grunt> CountByYear = FOREACH GroupByYear
>> GENERATE CONCAT((chararray)0,CONCAT('',(chararray)COUNT(1)));
# Write the output #####
grunt> STORE CountByYear
>> INTO '/user/yourlogin/pigoutput2' USING PigStorage('t');
#####
```

#####  
Solution :

```
##### Pig Execution #####
cbovigny@daplab-wn-03:~$ pig
# Put the bookpig.txt /shared/tp2/bookpig.txt into /user/yourlogin/
grunt> BookXRecords = LOAD '/user/yourlogin/bookpig.txt'
>> USING PigStorage(';') AS (ISBN:chararray,BookTitle:chararray,
>> BookAuthor:chararray,YearOfPublication:chararray,
>> Publisher:chararray,ImageURLs:chararray,ImageURLM:chararray,ImageURLL:chararray);
# create a variable GroupByYear#####
grunt> GroupByYear = GROUP BookXRecords BY YearOfPublication;
# create a variable CountByYear#####
grunt> CountByYear = FOREACH GroupByYear
>> GENERATE CONCAT((chararray)0,CONCAT('',(chararray)COUNT(1)));
# Write the output #####
grunt> STORE CountByYear
>> INTO '/user/yourlogin/pigoutput2' USING PigStorage('t');
#####
```

#####  
Allez controler le résultat dans votre output hdfs et passez-le en local.

## Spark

Maintenant nous allons effectuer une requête sur la base de données de hive avec spark.

```
import com.google.common.io.{Files, ByteStreams}

import java.io.File

import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.sql._
import org.apache.spark.sql.hive.HiveContext

object HiveSpark {

  def main(args: Array[String]) {
    val sparkConf = new SparkConf().setAppName("HiveFromSpark")
```

```

val sc = new SparkContext(sparkConf)
val hiveContext = new HiveContext(sc)
val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
import hiveContext.implicits._
import hiveContext.sql

// Get the count number of BXDataset
val count = sql("SELECT TO COMPLETE FROM BXDataSet").TOCOMPLETE.head.getLong(0)
println(s"COUNT NUMBER: $$count$")

// Here you have to complete the query :
val YearCount=hiveContext.sql("from bxdataset TO COMPLETE").TOCOMPLETE.foreach(
    println)

sc.stop()

```

#####

Solution :

```

import com.google.common.io.{Files, ByteStreams}

import java.io.File

import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.sql._
import org.apache.spark.sql.hive.HiveContext

object HiveSpark {

  def main(args: Array[String]) {
    val sparkConf = new SparkConf().setAppName("HiveFromSpark")
    val sc = new SparkContext(sparkConf)
    val hiveContext = new HiveContext(sc)
    val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
    import hiveContext.implicits._
    import hiveContext.sql

    // Get the count number of BXDataset
    val count = sql("SELECT COUNT(*) FROM BXDataSet").collect().head.getLong(0)
    println(s"COUNT NUMBER: $count")

    val YearCount=hiveContext.sql("from bxdataset SELECT yearofpublication, count(
        booktitle) group by yearofpublication order by yearofpublication").collect().
        foreach(println)

    sc.stop()
  }
}

```

#####

Maintenant il s'agit de lancer ce code sur le cluster DAPLAB. : Downloader le projet github sur votre compte local de Daplab.

```

git clone https://github.com/bovigny/HiveSpark.git
cbovigny@daplab-wn-03:~$ cd HiveSpark

```



```
|| cbovigny@daplab-wn-03:~/HiveSpark$ sbt package
```

Et finalement lancez le programme au moyen de spark-submit en passant par yarn.

```
|| spark-submit --class "HiveSpark" --master yarn-client --num-executors 4 --executor-  
||     cores 4 --executor-memory 8g /home/cbovigny/HiveSpark/target/scala-2.11/hivespark_2  
||     .11-1.0.jar
```