Ministry of Higher Education and Research
Higher School of Computer Science 08 May 1945 - Sidi Bel Abbes

Second Year Second Cycle - Artificial Intelligence and Data Science

Lab 01 : Hadoop

Presented By : FELLAH Abdelnour.

Date: February 17, 2024

1 INTRODUCTION

In this Big Data Lab, we will explore Apache Hadoop and MapReduce basics.where we'll understand managing vast data efficiently. In today's digital era, scalable solutions like Hadoop are essential for processing massive datasets. MapReduce complements Hadoop by providing a powerful processing model. We'll dive into setting up clusters, designing MapReduce jobs.

2 PART 01 : SETUP

1- Downloading the spark hadoop image :

Downloading the spark hadoop docker image can be acheived by either searching for the image and downloading it using docker desktop or running the following unix command :

```
      # docker pull liliasfaxi/spark-hadoop:hv-2.7.2
```

2- Creating the bridge network and the docker containers :
to create the bridge network that connects the three containers,we can use the following command :

```
      #  docker network create --driver=bridge hadoop
```

and to create the three container with the specified ports,we run the following command :

```
      #  docker run -itd --net=hadoop -p 50070:50070 -p 8088:8088 -p 7077:7077 -p 16010:16010  --name
hadoop-master --hostname hadoop-master  liliasfaxi/spark-hadoop:hv-2.7.2

      #  docker run -itd -p 8040:8042 --net=hadoop  --name hadoop-slave1
--hostname hadoop-slave1  liliasfaxi/spark-hadoop:hv-2.7.2

      #  docker run -itd -p 8041:8042 --net=hadoop  --name hadoop-slave2
--hostname hadoop-slave2  liliasfaxi/spark-hadoop:hv-2.7.2
```

3- Entering the master container :

This can be done using the following commands :

```
      # docker exec -it hadoop-master bash
      root@hadoop-master:#./start-hadoop.sh
```

3 PART 02 : FIRST STEPS WITH HADOOP :

1- Create a directory named input :

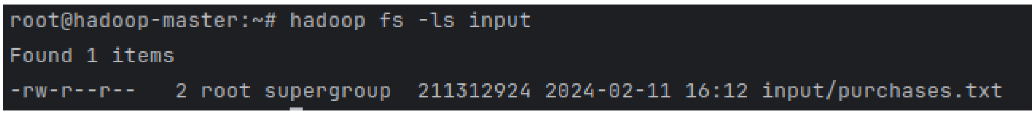
```
      root@hadoop-master:#hadoop fs -mkdir -p input
```

2- Load the puchases file into recently created input directoty :

```
      root@hadoop-master:#hadoop fs -put purchases.txt input
```

3- show the content of the directory input :

```
      root@hadoop-master:#hadoop fs -ls input
```

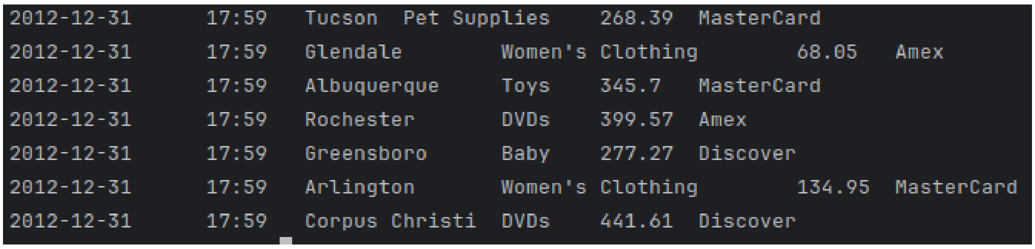


```
root@hadoop-master:~# hadoop fs -ls input
Found 1 items
-rw-r--r--   2 root supergroup  211312924 2024-02-11 16:12 input/purchases.txt
```

Figure 1: The display of the content of the directory input

4- show the tail of the purchases.txt file :

```
root@hadoop-master:# hadoop fs -tail input/purchases.txt
```



```
2012-12-31      17:59   Tucson  Pet Supplies    268.39  MasterCard
2012-12-31      17:59   Glendale        Women's Clothing        68.05   Amex
2012-12-31      17:59   Albuquerque     Toys    345.7   MasterCard
2012-12-31      17:59   Rochester       DVDs    399.57  Amex
2012-12-31      17:59   Greensboro      Baby    277.27  Discover
2012-12-31      17:59   Arlington       Women's Clothing        134.95  MasterCard
2012-12-31      17:59   Corpus Christi  DVDs    441.61  Discover
```

Figure 2: The display of the couple last rows of the file purchases.txt

4 Part 03 : Map Reduce

A map reduce job is mainly composed of tow main parts,the mappers and the reducers,the mappers are responsible about returning the key-value pairs,the reducers that are responsible about sorting and aggregating the results.

We are going to test a MapReduce program using a very simple example, WordCount that calculates the number of occurences for each unique word in the text by following the following steps :

- First we create a new maven project,make sure you include the right java target (8) and the necessary dependencies,see the pom.xml file bellow for more details.
- Create three classes Main class (driver),the WordsCountMap class and the WordsCountReducer class.
- We then implement the code for each class (see below for more details).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd
       /maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>org.example</groupId>
   <artifactId>TP2-BDT</artifactId>
   <version>1.0-SNAPSHOT</version>

   <properties>
      <maven.compiler.source>8</maven.compiler.source>
      <maven.compiler.target>8</maven.compiler.target>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   </properties>
   <dependencies>
      <dependency>
         <groupId>org.apache.hadoop</groupId>
         <artifactId>hadoop-common</artifactId>
         <version>2.7.2</version>
      </dependency>
      <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-mapreduce-client-co
      <dependency>
         <groupId>org.apache.hadoop</groupId>
         <artifactId>hadoop-mapreduce-client-core</artifactId>
         <version>2.7.2</version>
      </dependency>
      <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-hdfs -->
      <dependency>
         <groupId>org.apache.hadoop</groupId>
         <artifactId>hadoop-hdfs</artifactId>
         <version>2.7.2</version>
      </dependency>
      <dependency>
         <groupId>org.apache.hadoop</groupId>
         <artifactId>hadoop-mapreduce-client-common</artifactId>
         <version>2.7.2</version>
      </dependency>
   </dependencies>
</project>
```

Figure 3: the content of the pom.xml file.

```java
package org.example;


import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;

import java.io.IOException;

public class Main {
   public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

      Configuration conf = new Configuration();
      String[] remainingArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
      Job job = Job.getInstance(conf, "Words Counter v1.0");

      job.setJarByClass(Main.class);
      job.setMapperClass(WordsCountMap.class);
      job.setReducerClass(WordsCountReduce.class);

      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(IntWritable.class);

      FileInputFormat.addInputPath(job, new Path(remainingArgs[0]));
      FileOutputFormat.setOutputPath(job, new Path(remainingArgs[1]));

      if (job.waitForCompletion(true)) {
         System.exit(0);
      }

      System.exit(-1);

   }
}
```

Figure 4: The Main class code

```java
package org.example;

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import java.util.StringTokenizer;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;

public class WordsCountMap extends Mapper<Object, Text, Text, IntWritable> {

   private static final IntWritable ONE = new IntWritable(1);

   @Override
   protected void map(Object key, Text value, Mapper<Object, Text, Text, IntWritable>.
       Context context) throws IOException, InterruptedException {

      // create a word tokenizer
      StringTokenizer tokenizer = new StringTokenizer(value.toString());

      while (tokenizer.hasMoreTokens()) {
         Text word = new Text(tokenizer.nextToken());
         context.write(word, ONE);
      }
   }
}
```

Figure 5: The code for the WordsCountMap class

```
package org.example;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;
import java.util.Iterator;
import java.io.IOException;

public class WordsCountReduce extends Reducer<Text,IntWritable,Text,Text> {

   @Override
   protected void reduce(Text key, Iterable<IntWritable> values, Reducer<Text,
       IntWritable, Text, Text>.Context context) throws IOException, InterruptedException
        {

      Iterator<IntWritable> i = values.iterator();
      int count = 0;

      while (i.hasNext()) {
         count += i.next().get();
      }

      context.write(key, new Text(count + " occurences"));

   }
}
```

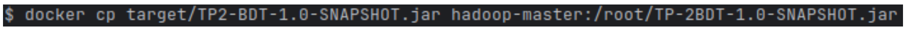
Figure 6: The code for the WordsCountReduce class

Now that we've implemented the different required class we now generate a .jar file using maven.

We then copy the generated .jar file to the master node with the following command :

```
# docker cp [file].jar hadoop-master:/root/[file].jar
```

In our case the command is :

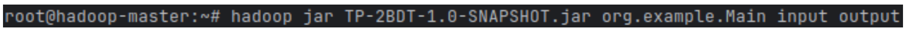


```
$ docker cp target/TP2-BDT-1.0-SNAPSHOT.jar hadoop-master:/root/TP-2BDT-1.0-SNAPSHOT.jar
```

Now we can use the run the .jar file and give it as arguments the main class,the input folder and the output folder,the command is :

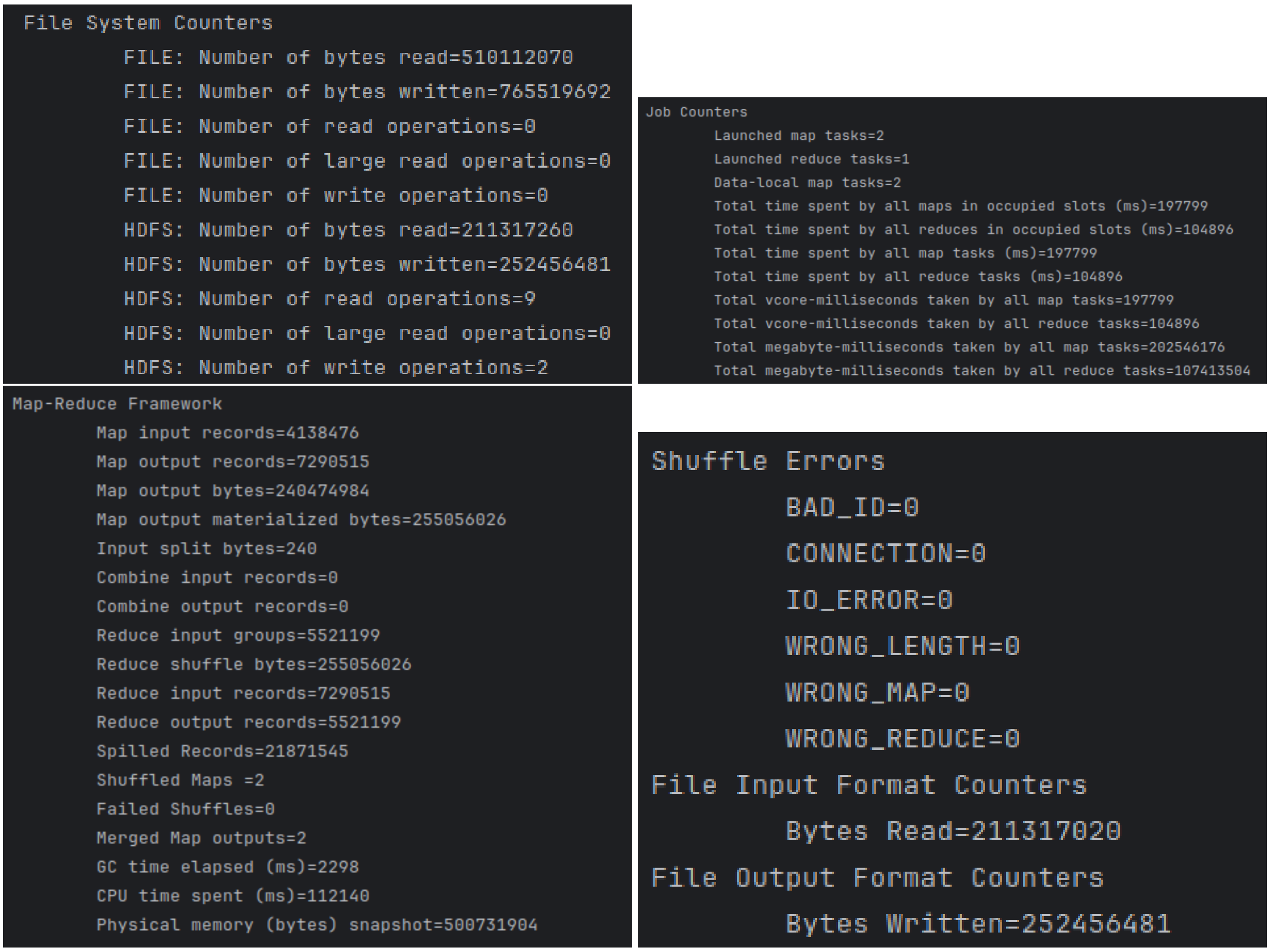
```
root@hadoop-master:#hadoop jar [file].jar [Main-CLass] [input-folder] [output-folder]
```

In our case the command is :



```
root@hadoop-master:~# hadoop jar TP-2BDT-1.0-SNAPSHOT.jar org.example.Main input output
```

the output of this command yields some statistics and information about the performed job.



```
File System Counters
        FILE: Number of bytes read=510112070
        FILE: Number of bytes written=765519692
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=211317260
        HDFS: Number of bytes written=252456481
        HDFS: Number of read operations=9
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
```

```
Job Counters
        Launched map tasks=2
        Launched reduce tasks=1
        Data-local map tasks=2
        Total time spent by all maps in occupied slots (ms)=197799
        Total time spent by all reduces in occupied slots (ms)=104896
        Total time spent by all map tasks (ms)=197799
        Total time spent by all reduce tasks (ms)=104896
        Total vcore-milliseconds taken by all map tasks=197799
        Total vcore-milliseconds taken by all reduce tasks=104896
        Total megabyte-milliseconds taken by all map tasks=202546176
        Total megabyte-milliseconds taken by all reduce tasks=107413504
```

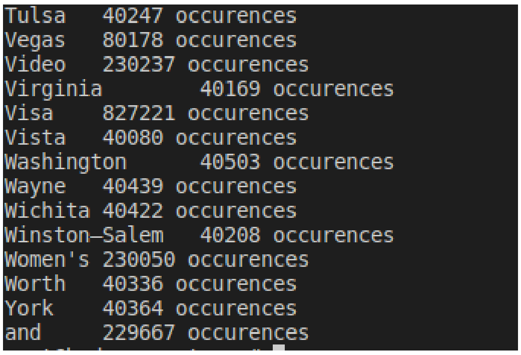
```
Map-Reduce Framework
        Map input records=4138476
        Map output records=7290515
        Map output bytes=240474984
        Map output materialized bytes=255056026
        Input split bytes=240
        Combine input records=0
        Combine output records=0
        Reduce input groups=5521199
        Reduce shuffle bytes=255056026
        Reduce input records=7290515
        Reduce output records=5521199
        Spilled Records=21871545
        Shuffled Maps =2
        Failed Shuffles=0
        Merged Map outputs=2
        GC time elapsed (ms)=2298
        CPU time spent (ms)=112140
        Physical memory (bytes) snapshot=500731904
```

```
Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
File Input Format Counters
        Bytes Read=211317020
File Output Format Counters
        Bytes Written=252456481
```

Figure 7: The oputput of the terminal after running the .jar file

to display the last rows of the generated file we run the command :

```
root@hadoop-master:#hadoop fs -tail output/part-r-00000
```



```
Tulsa   40247 occurences
Vegas   80178 occurences
Video   230237 occurences
Virginia        40169 occurences
Visa    827221 occurences
Vista   40080 occurences
Washington      40503 occurences
Wayne   40439 occurences
Wichita 40422 occurences
Winston-Salem   40208 occurences
Women's 230050 occurences
Worth   40336 occurences
York    40364 occurences
and     229667 occurences
```

Figure 8: A sample of the content of the output file.

5 Part 04 : Python version

to implement MapReduce using python,follow the steps bellow :

Step 01 : create tow python files (mapper.py and reducer.py),with the following code :

```
#!/usr/bin/env python
"""mapper.py"""

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
   # remove leading and trailing whitespace
   line = line.strip()
   # split the line into words
   words = line.split()
   # increase counters
   for word in words:
      # write the results to STDOUT (standard output);
      # what we output here will be the input for the
      # Reduce step, i.e. the input for reducer.py
      #
      # tab-delimited; the trivial word count is 1
      print("%s\t%s" % (word, 1))
```

Figure 9: The code for the mapper.py file

```
#!/usr/bin/env python
"""reducer.py"""

import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
   # remove leading and trailing whitespace
   line = line.strip()

   # parse the input we got from mapper.py
   word, count = line.split("\t", 1)

   # convert count (currently a string) to int
   try:
      count = int(count)
   except ValueError:
      # count was not a number, so silently
      # ignore/discard this line
      continue

   # this IF-switch only works because Hadoop sorts map output
   # by key (here: word) before it is passed to the reducer
   if current_word == word:
      current_count += count
   else:
      if current_word:
         # write result to STDOUT
         print("%s\t%s" % (current_word, current_count))
      current_count = count
      current_word = word

# do not forget to output the last word if needed!
if current_word == word:
   print("%s\t%s" % (current_word, current_count))
```

Figure 10: The code for the reducer.py file

Step 02 : Download hadoop-streaming JAR 2.7.3 from this link.

Step 03 : Move the downloaded jar file,and the tow python programs to the master container,using the following commands :

```
    # docker cp /path/to/hadoop-streaming-2.7.3.jar hadoop-master:/root/hadoop-streaming-2.7.3.jar

    # docker cp /path/to/mapper.py hadoop-master:/root/mapper.py

    # docker cp /path/to/reducer.py hadoop-master:/root/reducer.py
```

Step 04: Change the permissions of the mapper.py and reducer.py so they are executable by the user.

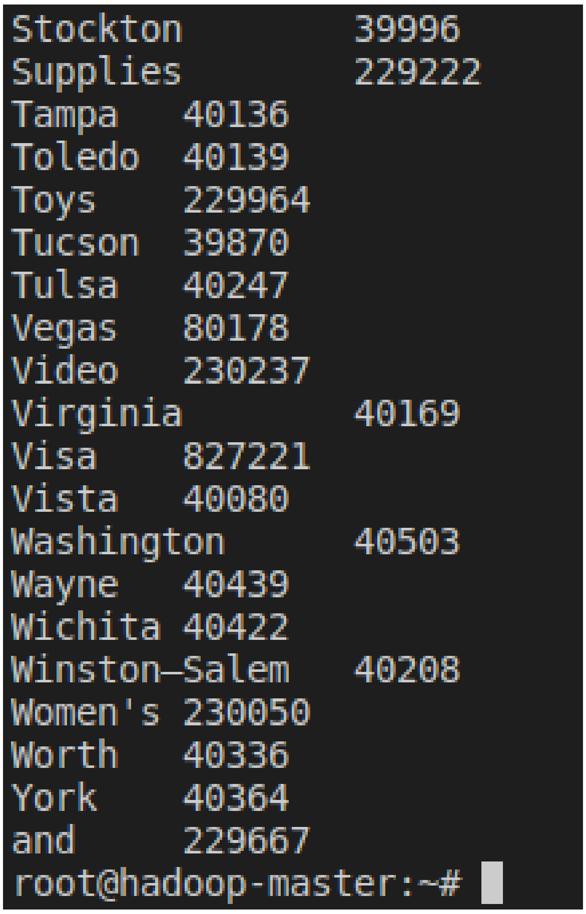
```
    root@hadoop-master:#chmod 744 mapper.py reducer.py
```

Step 05 : start the MapReduce job with the following command.

```
    root@hadoop-master:#hadoop jar hadoop-streaming-2.7.3.jar
-file ./mapper.py -mapper "python3 mapper.py"
-file ./reducer.py -reducer "python3 reducer.py"
-input input/purchases.txt  -output py-output
```

Now you can check out the result of this command :

```
    root@hadoop-master:#hadoop fs -tail py-output/part-00000
```



```
Stockton        39996
Supplies        229222
Tampa   40136
Toledo  40139
Toys    229964
Tucson  39870
Tulsa   40247
Vegas   80178
Video   230237
Virginia        40169
Visa    827221
Vista   40080
Washington      40503
Wayne   40439
Wichita 40422
Winston—Salem   40208
Women's 230050
Worth   40336
York    40364
and     229667
root@hadoop-master:~#
```

Figure 11: A sample of the content of the output file (python version).

6 CONCLUSION

Our Big Data Lab journey has been enlightening and empowering. We've explored Hadoop and MapReduce hands-on, gaining insights into distributed computing's transformative power. With Hadoop and MapReduce, we can handle complex data tasks confidently, driving positive change. As we conclude, let's carry forward our newfound skills to make an impact. In business, science, or public services, our expertise positions us at the forefront of the data-driven revolution. We're ready to shape the future of big data analytics with innovation and meaningful contributions.