MAPREDUCE PRACTICE

The Volvo Survey

Celada Matías Mario
71814922-K

Table of Contents:

PROBLEM CHOSEN:

The constant non-stop development of the automotive industry, is a great handicap the car manufacturing industries need to cope with, so that they can offer the latest state of the art vehicle technologies in order to attract the clients. In this line, Volvo Cars (*Volvo personvagnar AB*) is facing the development of brand-new petrol/petrol-hybrid engines, more efficient, eco-friendly and suitable for everyone. With this premises in mind, the only problem is to determine the power/horsepower they need to provide the engines with, so that they are versatile enough to please everyone. To do so, Volvo Cars has decided to survey family units in three different Swedish cities, Gothenburg, Uppsala and Stockholm to know the mean power people usually buy or need per each city.

The data collected comprise records for each of the family units surveyed. All of those disposed withing a single pain text file and in different lines each. Each record/line contains different fields separated by slash bars "/" and since in Sweden many family units have two cars, those records do contain two values in some fields but concatenated as if they were only one via symbols such as '+','@' or '&'. Also, invalid records can happen due to the survey processing program or because the surveyed people leaving the fields in blank if they do not know the specific value it is been asked for. The concrete specification of the fields Volvo Cars is been recording is:

```
Stockholm/H+H/Jul2020@Nov2012/221+91@191+73/54384&23580
1         2        3                4                5
```

No.	Description	Possible values and meaning
1	City of the surveyed family unit.	Gothenburg, Uppsala and Stockholm.
2	Type of vehicle/s the family has.	H for Hybrid; E for Electric; C for Convectional. A '+' between divides the 2 cars in that family unit.
3	Build date of the vehicle/s.	Month and Year of the manufacturing of the car. An '@' divides the two cars for this recorded family unit.
4	Horsepower of the vehicle/s.	Value of power (HP) for C and E cars or, if it is Hybrid the Powers of the two engines the car has, expressed with a + between, as in the shown example. An '@' divides the two cars for this recorded family unit
5	Actual kilometres of the vehicle/s.	The value of the kms. Driven in each. An '&' divides the two cars for this recorded family unit.

Why is Volvo Cars interested in so many different fields if its only developing new engines? Well, Volvo Cars has requested some restrictions within the data collected that is to be processed according to them:

First of all, invalid records (whenever a record contains "---" in some of its fields) are discarded altogether, because they are considered not trustable.

If the vehicle is Electric the fields for it are discarded also, because Volvo Cars is developing new petrol and petrol-hybrid engines, so processing electric cars data is not interesting for them at all. This means, if an electric car is present, either as single car of the record or with another car in the record (if the family unit has two cars), the electric car fields are discarded, but not the other´s if there is another car.

The fields of cars having more than 450000 kilometres are also discarded, because people doing so much kilometres are usually buying diesel alternatives so they would not be interested in buying these petrol/petrol-hybrid engines. Furthermore, if someone´s car has got that number of kilometres it goes almost for sure that he/she will keep it, either because it's a classic car or because they will enlarge it until it breaks down. If there are two cars in the family, and the other one has less kms. this last one is valid.

Finally, Volvo Cars has imposed that the fields of cars being manufactured before than year 2000 are discarded altogether. The have more than 20 years and in that time, the requirements of power and the advances in the metals, materials and safety equipment have meant that more and more power is being needed where more than 20 years ago was not needed, so to obtain more realistic power records, the fields are discarded. Also has relation with the previous requirements because they may be classic or collection cars that they are not going to change. Again, if the family has two cars, the other being built in 2000 or onwards has its fields kept.

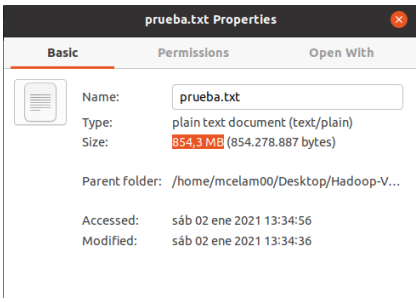
All in all, with these restrictions in mind, Volvo Cars requires the processing of this data in order to obtain the resulting mean of power (Horsepower) of all the valid records remaining per city surveyed.

It is important to clarify that whenever an hybrid car is found, the effective power as the addition of both engine Powers separated by a + is computed and passed as value.

If the family unit has two cars, the value passed to the reducer will be the mean of both Powers and if they are hybrids, the mean of the two effective Powers.

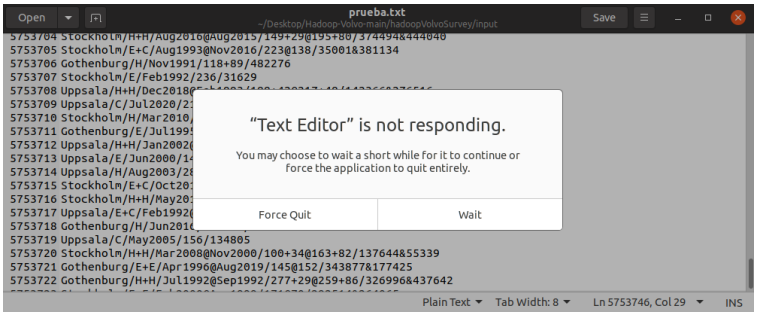
NOTE: This problem represents a totally fictitious scenario that in no case is true, nor should it be taken as such, it has only been devised as a bulwark for the development of a work through MapReduce.

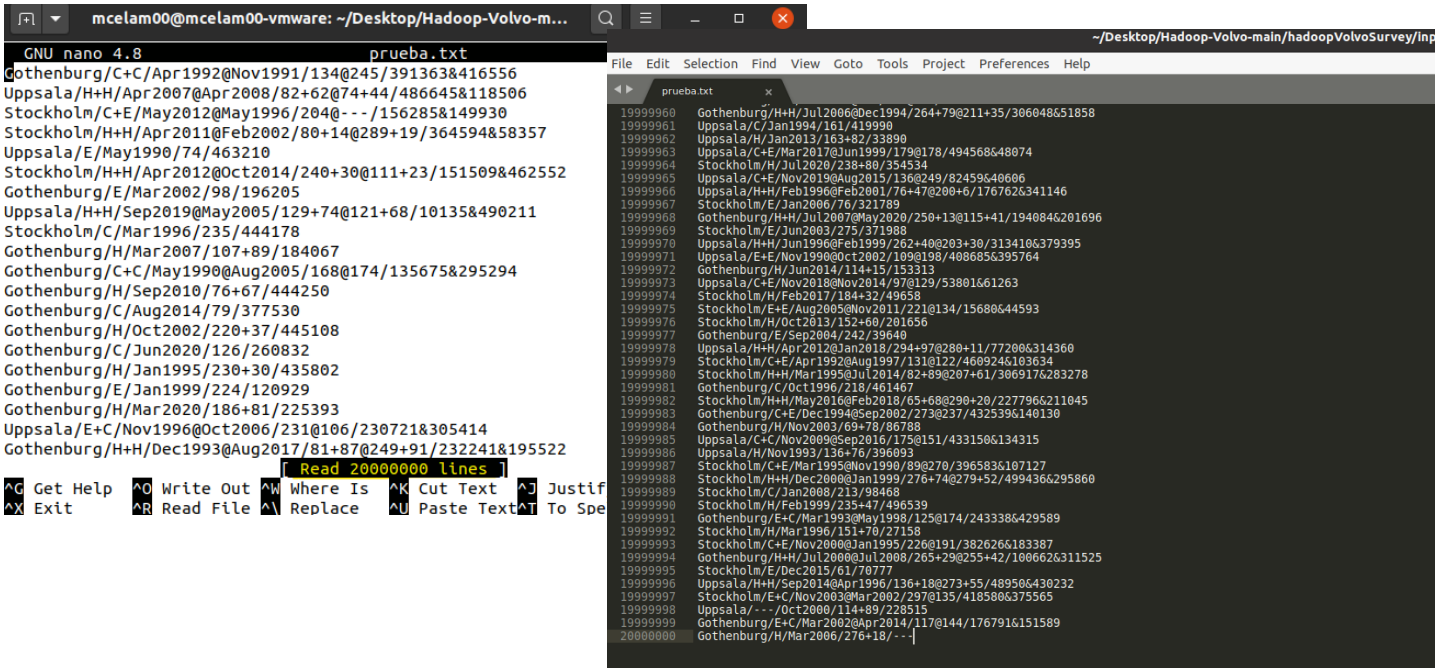
SAMPLE INPUT FILE:



The input file that was passed to the MapReduce program contained exactly 20.000.000 records, meaning a total size of 854,3 MB.

Gedit couldn´t cope with it, but fortunately sublime and nano agreed in the number of lines



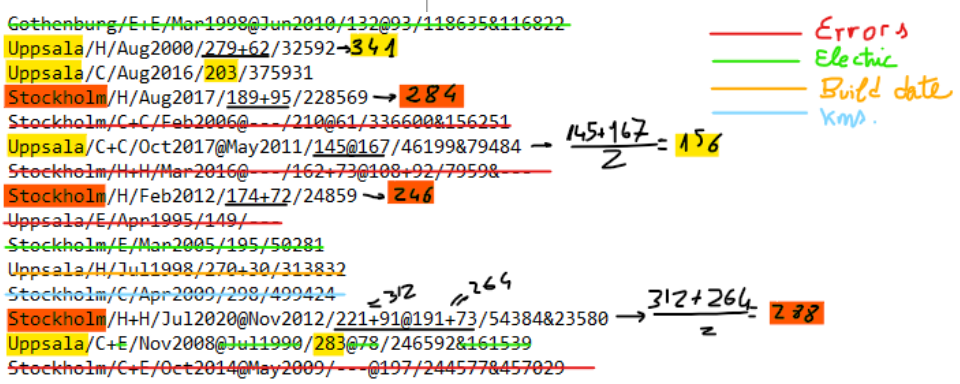


Since this definitive file is huge for me to explain the MapReduce procedure, I have a much more simpler and smaller subset in order to demonstrate the effectiveness of the program by doing myself the calculations by hand and comparing with the execution results.

With this smaller subset, let's go to explain what the program is going to Map and Reduce:

We can see the three cities present, records for family units with 2 cars and with 1 car, with various Powers, kilometres and build dates and also the three types of cars, Electric, Hybrid and Conventional and some invalid fields.

```
Gothenburg/E+E/Mar1998@Jun2010/132@93/118635&116822
Uppsala/H/Aug2000/279+62/32592
Uppsala/C/Aug2016/203/375931
Stockholm/H/Aug2017/189+95/228569
Stockholm/C+C/Feb2006@---/210@61/336600&156251
Uppsala/C+C/Oct2017@May2011/145@167/46199&79484
Stockholm/H+H/Mar2016@---/162+73@108+92/7959&---
Stockholm/H/Feb2012/174+72/24859
Uppsala/E/Apr1995/149/---
Stockholm/E/Mar2005/195/50281
Uppsala/H/Jul1998/270+30/313832
Stockholm/C/Apr2009/298/499424
Stockholm/H+H/Jul2020@Nov2012/221+91@191+73/54384&23580
Uppsala/C+E/Nov2008@Jul1990/283@78/246592&161539
Stockholm/C+E/Oct2014@May2009/---@197/244577&457029
```



So, having applied the restrictions that I explained deeply in the previous part, the mapper would pass the following data (pairs key-value) to the reducer:

```
Uppsala/H/Aug2000/279+62/32592 -->  341
Uppsala/C/Aug2016/203/375931   --> 203
Uppsala/C+C/Oct2017@May2011/145@167/46199&79484    --> 145+167/2  :: 156 resto 0
Uppsala/C/Nov2008/283/246592   --> 283


Stockholm/H/Aug2017/189+95/228569  ---> 284
Stockholm/H/Feb2012/174+72/24859  ---> 246
Stockholm/H+H/Jul2020@Nov2012/221+91@191+73/54384&23580  --> 312+264/2  ::   288 resto 0
```

The reducer will take the keys and values and calculate the mean horse power per city surveyed with the following results

```
Uppsala/H/Aug2000/279+62/32592 -->  341
Uppsala/C/Aug2016/203/375931   --> 203
Uppsala/C+C/Oct2017@May2011/145@167/46199&79484    --> 145+167/2  :: 156 resto 0
Uppsala/C/Nov2008/283/246592   --> 283
-------------------------------------------------------------
                                        245 resto 3

Stockholm/H/Aug2017/189+95/228569  ---> 284
Stockholm/H/Feb2012/174+72/24859  --->  246
Stockholm/H+H/Jul2020@Nov2012/221+91@191+73/54384&23580  --> 312+264/2  ::   288 resto 0
-----------------------------------------------------------------------
                                        272 resto 2
```

Since my program is not taking into account the reminders of the divisions the results would be

Uppsala → 245

Stockholm → 272

And if we load that file in the program and execute Hadoop it gives that result:

```
Open   part-r-00000   Save
~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey/output
1 Stockholm      272
2 Uppsala        245
```

PROGRAM IMPLEMENTED:

DRIVER:

```
//volvoSurveyDriver
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
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.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class volvoSurveyDriver extends Configured implements Tool {

  @Override
  public int run(String[] args) throws Exception {

    if (args.length != 2) {

      System.err.printf("Usage: %s [generic options] <input>
<output>\n",getClass().getSimpleName());
      ToolRunner.printGenericCommandUsage(System.err);
      return -1;

    }

    Job job = new Job(getConf(), "Mean Power per family Unit");
    job.setJarByClass(getClass());

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

    job.setMapperClass(volvoSurveyMapper.class);
    job.setCombinerClass(volvoSurveyReducer.class);
    job.setReducerClass(volvoSurveyReducer.class);

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

    return job.waitForCompletion(true) ? 0 : 1;
  }

  public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new volvoSurveyDriver(), args);
    System.exit(exitCode);
  }
}
```

NOTE: From here and onwards, some margins in the document, have been deleted in order to make the code cleaner and more readable, but in advance I do apologise if the format of the code is not the best, I hope it is understandable.

MAPPER:

```java
//volvoSurveyMapper

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class volvoSurveyMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

  private CarDataRecordParser parser = new CarDataRecordParser();

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

      /*SEPARATING RECORD PARTS*/

      parser.parse(value);

      /*ERRORS IN SOME FIELD OF THE RECORDS*/

      if(parser.isValidRecord() == false) {


          /*It is declined completely due to considering it unreliable*/


      /*NO ERRORS 1 CAR PER THIS RECORD*/

      }else if(parser.numOfCars() == 1) {

            /*If it is not electric, ok*/
          if(parser.getTypeCar1().compareTo("E") != 0) {

              /*It has to have less than 450.000km and be newer than 2000*/
              if(parser.isAfter2000(parser.getTypeCar1()) == true && parser.isLessThan450000(parser.getTypeCar1()) == true) {

                  //Precondition: Convectional or hybrids with this satisfied conditions

                  if(parser.getTypeCar1().compareTo("C") == 0) { //if Conventional: we write in the context

                      context.write(new Text(parser.getCity()),new IntWritable(Integer.parseInt(parser.getHorsePowerCar1())));


                  }else if(parser.getTypeCar1().compareTo("H") == 0) { //if Hybrid: we calculate the effective power and write in the context


                      context.write(new Text(parser.getCity()),new IntWritable(parser.hyEffectivePowerCar1()));


                  }

              }

          }


      /*NO ERRORS 2 CARS PER THIS RECORD*/

      }else {

          if(parser.getTypeCar1().compareTo("E") == 0 && parser.getTypeCar2().compareTo("E") == 0) {

              /*both are electric == record declined*/


          }else {
              //means, one is electric, the other is or none

              if(parser.getTypeCar1().compareTo("E") == 0) {//if one is electric we write the other in the context

                  /*It has to have less than 450.000km and be newer than 2000*/
                  if(parser.isAfter2000(parser.getTypeCar2()) == true && parser.isLessThan450000(parser.getTypeCar2()) == true) {
                      //Since no elect+hyb is conventional

                      context.write(new Text(parser.getCity()),new IntWritable(Integer.parseInt(parser.getHorsePowerCar2())));

                  }


              }else if(parser.getTypeCar2().compareTo("E") == 0) { //if the other is electric we do the other way round


                  if(parser.isAfter2000(parser.getTypeCar1()) == true && parser.isLessThan450000(parser.getTypeCar1()) == true) {
                      //Since no hybrid+electric is conventional

                      context.write(new Text(parser.getCity()),new IntWritable(Integer.parseInt(parser.getHorsePowerCar1())));

                  }



              }else {

                  //or the 2 conventional or the two hybrids (no persons in the survey with Hybrid and conventional)

                  if(parser.getTypeCar1().compareTo("C") == 0 && parser.getTypeCar2().compareTo("C") == 0) {

                      //We need to check if both of them satisfy the age and kms

                      if(parser.isAfter2000(parser.getTypeCar1()) == true) { //first has good age
                          if(parser.isLessThan450000(parser.getTypeCar1()) == true) { //first has good kms
                                //first is ok
                              if(parser.isAfter2000(parser.getTypeCar2()) == true) { //second has good age
                                  if(parser.isLessThan450000(parser.getTypeCar2()) == true) { //second has good kms
```

```
                //second is ok and we pass the mean to the reducer

                /*We take both cars powers and do the mean*/
            int pw1 = Integer.parseInt(parser.getHorsePowerCar1());
            int pw2 = Integer.parseInt(parser.getHorsePowerCar2());

            int mean = (pw1+pw2)/2;

            context.write(new Text(parser.getCity()),new IntWritable(mean));



        }else {//second has not good kms (declined)

            //we write the first


            context.write(new Text(parser.getCity()),new IntWritable(Integer.parseInt(parser.getHorsePowerCar1())));



        }

    }else { //second has not good age (declined)

        //we write the first


        context.write(new Text(parser.getCity()),new IntWritable(Integer.parseInt(parser.getHorsePowerCar1())));



    }

}else{ //first has not good kms (declined)

    //second is ok?

    if(parser.isAfter2000(parser.getTypeCar2()) == true && parser.isLessThan450000(parser.getTypeCar2()) == true) {
          //yes

        context.write(new Text(parser.getCity()),new IntWritable(Integer.parseInt(parser.getHorsePowerCar2())));



    }

}

}else { //first not good age (declined)

  //second is ok?

    if(parser.isAfter2000(parser.getTypeCar2()) == true && parser.isLessThan450000(parser.getTypeCar2()) == true) {
          //yes

        context.write(new Text(parser.getCity()),new IntWritable(Integer.parseInt(parser.getHorsePowerCar2())));



    }

}

}else if(parser.getTypeCar1().compareTo("H") == 0 && parser.getTypeCar2().compareTo("H") == 0) {

  //We need to check if both of them satisfy the age and kms

    if(parser.isAfter2000(parser.getTypeCar1()) == true) { //first has good age
        if(parser.isLessThan450000(parser.getTypeCar1()) == true) { //first has good kms
              //first is ok
            if(parser.isAfter2000(parser.getTypeCar2()) == true) { //second has good age
                if(parser.isLessThan450000(parser.getTypeCar2()) == true) { //second has good kms
                      //second is ok and we pass the mean to the reducer

                      /*We take both cars powers and do the mean*/
                  int pw1 = parser.hyEffectivePowerCar1();
                  int pw2 = parser.hyEffectivePowerCar2();

                  int mean = (pw1+pw2)/2;

                  context.write(new Text(parser.getCity()),new IntWritable(mean));



                }else {//second has not good kms (declined)

                    //we write the first


                    context.write(new Text(parser.getCity()),new IntWritable(parser.hyEffectivePowerCar1()));



                }

            }else { //second has not good age (declined)

                //we write the first


                context.write(new Text(parser.getCity()),new IntWritable(parser.hyEffectivePowerCar1()));



            }
```

```
                    }else{ //first has not good kms (declined)

                        //second is ok?

                        if(parser.isAfter2000(parser.getTypeCar2()) == true && parser.isLessThan450000(parser.getTypeCar2()) == true) {
                              //yes

                            context.write(new Text(parser.getCity()),new IntWritable(parser.hyEffectivePowerCar2()));


                        }

                    }

                }else { //first not good age (declined)

                  //second is ok?

                    if(parser.isAfter2000(parser.getTypeCar2()) == true && parser.isLessThan450000(parser.getTypeCar2()) == true) {
                          //yes

                        context.write(new Text(parser.getCity()),new IntWritable(parser.hyEffectivePowerCar2()));


                    }

                }

            }

        }

    }

    }
}
```

REDUCER:

```
//volvoSurveyReducer
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

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

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

   int mean = 0;
   int partialSum = 0;
   int counted = 0;

   for (IntWritable value : values) {
       counted++;
       partialSum = partialSum + value.get();
   }

   mean = partialSum/counted;

   context.write(key, new IntWritable(mean));

  }
}
```

PARSER:

```java
//CarDataRecordParser
import org.apache.hadoop.io.Text;

public class CarDataRecordParser {

     private String city;    //would be the key
     private String type[]; //since we could have 2 cars I represent is as an array, if only one is full the other would not even exist
     private String buildDate[];
     private String horsePower[];
     private String kilometres[];
     private int numOfCars; //just to keep info of the number of cars using the size of the arrays

  public void parse(String record) {

      /*HERE WE HAVE 1 LINE OF THE FILE IN STRING FORMAT*/

      //We are going to split the fields by the "/" so that we can separate each of them

      String [] holeRecord;

      holeRecord = record.split("/"); //in this way we achieve, in each position of the array one record 0 would be the city; 1 the type; 2
the dates...

      this.city = holeRecord[0];

          if(areThere2Cars(holeRecord[3]) == true) {    //If there are two cars, both array positions would be created

                this.numOfCars = 2; //set the attribute first of all for later methods

              /**TYPES**/
                 this.type = holeRecord[1].split("\\+"); //holeRecord[1] would be the String representing the types concatenated by a +;
so, we split them and save them likewise we did with the "/"

              /**BUILD DATES**/
                 this.buildDate = holeRecord[2].split("@");

              /**HORSE POWERS**/
                 this.horsePower = holeRecord[3].split("@");

              /**KILOMETRES**/
                 this.kilometres    = holeRecord[4].split("&");

          }else { //whereas if there is only one car, just an array of 1 would be created

                this.numOfCars = 1; //set the attribute first of all for later methods


              /*ARRAY CREATION*/
                 this.type = new String[1];
                 this.buildDate = new String[1];
                 this.horsePower = new String[1];
                 this.kilometres = new String[1];

              /**TYPE**/
                 this.type[0] = holeRecord[1];

              /**BUILD DATE**/
                 this.buildDate[0] = holeRecord[2];

              /**HORSE POWERS**/
                 this.horsePower[0] = holeRecord[3];

              /**KILOMETRES**/
                 this.kilometres[0] = holeRecord[4];

          }



  }


            /*MISCELLANY*/

  /**
   * Parse from Text Hadoop format to String format
   * @param record
   */

  public void parse(Text record) {
    parse(record.toString());
  }

  /**
   * It looks for an @ all over the record of the horsepower and if it finds it, that would mean 2 cars
   * @return true if there are 2 cars, false in case just one
   */

  private boolean areThere2Cars(String hps) {

      boolean flag = false;

      for (int i = 0; i < hps.length(); i++){

          char c = hps.charAt(i); //I take char per char and look for the @

          if(c == '@') {
```

```
                flag = true;
                break;
            }

        }

        return flag;

    }


             /*VALIDATIONS AND VERIFICATIONS*/

    /*RECORD FIELDS VALIDATION*/


    public boolean isValidRecord() {

        boolean flag = true; //by default is valid unless --- is found in any field

        if(this.numOfCars == 1) {

            if(this.type[0].charAt(0) == '-' || this.buildDate[0].charAt(0) == '-' || this.horsePower[0].charAt(0) == '-' ||
this.kilometres[0].charAt(0) == '-') { //invalid field
                flag = false;
            }

        }else { //2cars, so we have 2 Strings in each array, so we look the first character of each of both strings to see if it is - meaning
that the second and third chars would be -- also constituting the invalid field

            if(this.type[0].charAt(0) == '-' || this.buildDate[0].charAt(0) == '-' || this.horsePower[0].charAt(0) == '-' ||
this.kilometres[0].charAt(0) == '-') { //invalid field
                flag = false;
            }else if (this.type[1].charAt(0) == '-' || this.buildDate[1].charAt(0) == '-' || this.horsePower[1].charAt(0) == '-' ||
this.kilometres[1].charAt(0) == '-') {
                flag = false;
            }

        }

      return flag;

    }


    /*GETTER NUMBER OF CARS*/


    public int numOfCars() {
        return this.numOfCars;
    }


    /*GETTER TYPE OF CAR 1*/


    public String getTypeCar1() {
        return this.type[0];
    }


    /*GETTER TYPE OF CAR 2*/


    public String getTypeCar2() {
        return this.type[1];
    }


    /*CAR AGE VALIDATION*/


    public boolean isAfter2000(String type) {
        //Depending on the car to validate one validation or the other

        boolean flag = true;

        if(type.compareTo(this.type[0]) == 0) { //is vehicle 1

            if(this.buildDate[0].charAt(3) != '2') { //Character in pos 3 is the first of the year if its not 2 not valid

                flag = false;

            }


        }else { //is vehicle 2


            if(this.buildDate[1].charAt(3) != '2') { //Character in pos 3 is the first of the year if its not 2 not valid

                flag = false;

            }



        }


        return flag;
```

```
  }

  /*CAR KMS VALIDATION*/

  public boolean isLessThan450000(String type) {

     boolean flag = true;

      if(type.compareTo(this.type[0]) == 0) { //is vehicle 1

          if(Integer.parseInt(this.kilometres[0]) > 450000) { //more than 450 000 not valid record
              flag = false;

          }

      }else { //is vehicle 2


            if(Integer.parseInt(this.kilometres[1]) > 450000) { //more than 450 000 not valid record
                  flag = false;

            }

      }


      return flag;


  }


  /*COMPUTES THE EFFECTIVE POWER OF THE HYBRID ENGINE Car 1*/


  public int hyEffectivePowerCar1() {

      String power[] = this.horsePower[0].split("\\+");

      int pw1 = Integer.parseInt(power[0]);
      int pw2 = Integer.parseInt(power[1]);

      return (pw1+pw2);
  }


  /*COMPUTES THE EFFECTIVE POWER OF THE HYBRID ENGINE Car 2*/

public int hyEffectivePowerCar2() {

      String power[] = this.horsePower[1].split("\\+");

      int pw1 = Integer.parseInt(power[0]);
      int pw2 = Integer.parseInt(power[1]);

      return (pw1+pw2);
  }



            /*GETTING THE KEY AND THE VALUE TO THE MAPPER*/

  public String getCity() { //key
    return this.city;
  }

  public String getHorsePowerCar1() { //value
    return this.horsePower[0];
  }
  public String getHorsePowerCar2() { //value'
        return this.horsePower[1];
      }

}
```

INPUTGENERATOR:

```java
package inputCreator;

import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;

import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

/**
 * Class in charge of the creation of an input file following the specifications provided
 * @author mcelam00
 */


public class InputCreator {

    String filePath = "";

    public static void main(String[] args) {

            System.out.println("Welcome to the Input creator, now, the generation of the input file will start, this could take some
minutes");

            InputCreator nuevo = new InputCreator();
            nuevo.savePath();
            System.out.println("input file succesfully generated");


    }

    /**
     * Method that brings in a dialog for us to select the path in which we want to save the input file
     **/

    private void savePath() {

        final JFileChooser fc = new JFileChooser(); //dialog box creation


        int returnVal = fc.showSaveDialog(null); //show it


        if(returnVal == JFileChooser.APPROVE_OPTION){  //if we click on accept

            File file = fc.getSelectedFile();

            //we save the path plus the file name we wish to include
            filePath = file.getAbsolutePath();

            //append of the .txt extension just in case
            if((filePath.contains(".txt")) == false) {

                filePath = filePath +".txt";

            }

        }else { //If we interrupt the saving proccess

            return;

        }


        File f = new File(filePath); //Verification of an existing equally-named file

        if(f.exists()) { //if it exists

            int respuestaBoton = JOptionPane.showConfirmDialog(null, "Warning. A file with the same name already exists.\n Do you want to
replace it?");


            if(respuestaBoton == 0) { //if the replace is wished

                //CALL TO THE FILEWRITTER
                this.generateFile();

            }else { //abort

                return;

            }

        }else { //if it does not exist

            //CALL TO THE FILEWRITTER
            this.generateFile();

        }


    }

    /**
     * Method that writes the file itself, by using the previous stored path
     */

    private void generateFile() {
```

```java
        FileWriter file = null; //Declaration of the FileWriter

        try {

            PrintWriter pw;

            file = new FileWriter(filePath); //new FileWriter to which we give the file in which we want to write. (like the paper)

            pw = new PrintWriter(file); //new PrintWritter in order to write in the file (like a pencil)

            //pw.println(); // pw.print(); function used to print one line in the indicated pathfile

            /*BEGGINING OF THE PRINTING*/

            int i = 0;
            int iterations = 100;

            while (i < 100) {
                conventionaletElectricRecord(pw);
                pw.println();
                hybridRecord(pw);
                    if(i < (iterations-1)) {
                        pw.println();
                    }

                    i++;
            }


        }catch(Exception e) {

            /*Included to avoid exception traces*/

        }finally { /*Ensuring the correct close of the file*/

            try {

                if(null != file) {

                    file.close();

                }

            }catch(Exception e2) {

                /*Included to avoid exception traces*/

            }
        }

        JOptionPane.showMessageDialog(null, "The Input file has been correctly saved", "Input Creator
v1",JOptionPane.INFORMATION_MESSAGE);



    }

    /**
     * Method that prints a record for electric/conventional car
     * @param pw PrintWritter to write in the file
     */

    private void conventionaletElectricRecord(PrintWriter pw) {

        String type = "";


        //number of cars that this family unit (represented by this record) has

        int cars = (int) Math.floor(Math.random()*2);  //1 or 2 cars


        if(cars == 1) { /*it has 2CARS, so we need to take at random the 2 types*/

            String type1= "";
            String type2= "";

            int num1 = (int) Math.floor(Math.random()*2);
            int num2 = (int) Math.floor(Math.random()*2);

            if(num1 == 0) {
                type1 = "C"; //conventional car
            }else {
                type1 = "E"; //electric car
            }

            if(num2 == 0) {
                type2 = "C"; //conventional car
            }else {
                type2 = "E"; //electric car
            }


            pw.print(randomCity()+"/"+type1+"+"+type2+"/"+randomDate(cars)+"/"+randomHps(cars)+"/"+randomKMs(cars));


        }else { /*ONLY ONE CAR*/

            int num = (int) Math.floor(Math.random()*2);  //random number between 0 and 1
```

```
            switch(num) { //depending on the random number one of the options will be chosen.

                case 0:
                    type = "C"; //conventional car
                    break;
                case 1:
                    type = "E"; //electric car
                    break;

            }



            pw.print(randomCity()+"/"+type+"/"+randomDate(cars)+"/"+randomHps(cars)+"/"+randomKMs(cars));
        }
    }

    /**
     * Method that generates a random number of kms for the cars.
     * If 2 cars are to be written there would be 2 kms values separated by an & symbol.
     */

    private String randomKMs(int cars) {

        String km = "";

        int error = (int) Math.floor(Math.random()*100); // if we get 17 out of 100 possible random numbers it simulates an error in the
record.

        if(cars == 1) {
            int km1 = (int) Math.floor(Math.random()*500000);  //random number between 0 and 999999
            int km2 = (int) Math.floor(Math.random()*500000);  //random number between 0 and 999999

            if(error == 17) {
                km = "---"+"&"+String.valueOf(km2);

            }else if(error == 18){
                km = String.valueOf(km1)+"&"+"---";

            }else {
                km = String.valueOf(km1)+"&"+String.valueOf(km2);
            }

        }else {

            int km1 = (int) Math.floor(Math.random()*500000);  //random number between 60 and 300

            if(error == 17) {
                km = "---";

            }else {
                km = String.valueOf(km1);

            }
        }

        return km;
    }

    /**
     * Method that generates a random number of cars (1 or 2) and a value of HP for each of them.
     * If 2 cars are to be written they would be separated by an @ symbol to simplify the map-reduce technique (1 pair key+value).
     */

    private String randomHps(int cars) {

        String hps = "";

        int error = (int) Math.floor(Math.random()*100); // if we get 17 out of 100 possible random numbers it simulates an error in the
record.

        if(cars == 1) {

            int h1 = (int) ((Math.random() * (300 - 60)) + 60);  //random number between 60 and 300
            int h2 = (int) ((Math.random() * (300 - 60)) + 60);  //random number between 60 and 300

            if(error == 17) {
                hps = "---"+"@"+String.valueOf(h2);

            }else if(error == 18){
                hps = String.valueOf(h1)+"@"+"---";

            }else {

                hps = String.valueOf(h1)+"@"+String.valueOf(h2);
            }

        }else {

            int num = (int) ((Math.random() * (300 - 60)) + 60);  //random number between 60 and 300

            if(error == 17) {
                hps = "---";

            }else {
                hps = String.valueOf(num);
            }
        }

        return hps;
    }

    /**
```

```java
     * Method that generates a random date between January 1990 and January 2021
     * @param cars
     */

    private String randomDate(int cars) {

        int year;
        int month;
        String date = "";

        int error = (int) Math.floor(Math.random()*100); // if we get 17 out of 100 possible random numbers it simulates an error in the
record.


            if(cars == 1) { //if there are 2 cars we need 2 dates

                /*car 1*/

                int year1 = (int) ((Math.random() * (2021 - 1990)) + 1990);

                int month1 = (int) Math.floor(Math.random()*12);

                /*car2*/

                int year2 = (int) ((Math.random() * (2021 - 1990)) + 1990);

                int month2 = (int) Math.floor(Math.random()*12);

                String mn1 = monthName(month1); //name of the month for car 1

                String mn2 = monthName(month2); //name of the month for car 2

                if(error == 17) { //if there is a simulated error, the date record is null

                    date = "---"+"@"+mn2+String.valueOf(year2);

                }else if(error == 18){
                    date = mn1+String.valueOf(year1)+"@"+"---";

                }else {


                    date = mn1+String.valueOf(year1)+"@"+mn2+String.valueOf(year2);
                }

            }else {

                /*just one car*/
                year = (int) ((Math.random() * (2021 - 1990)) + 1990);  //random number between 1990 and 2021 both inclusive

                month = (int) Math.floor(Math.random()*12);  //random number between 0 and 11

                if(error == 17) { //if there is a simulted error, the date record is null

                    date = "---";

                }else {

                    date = monthName(month)+String.valueOf(year);
                }

            }


        return date;
    }



    /**
     * Depending on the random number one of the options (month names) will be chosen.
     * @param num
     * @return
     */


    private String monthName(int num) {

        String monthS = "";

        switch(num) {

        case 0:
            monthS = "Jan";
            break;
        case 1:
            monthS = "Feb";
            break;
        case 2:
            monthS = "Mar";
            break;
        case 3:
            monthS = "Apr";
            break;
        case 4:
            monthS = "May";
            break;
        case 5:
            monthS = "Jun";
            break;
        case 6:
            monthS = "Jul";
```

```
            break;
        case 7:
            monthS = "Aug";
            break;
        case 8:
            monthS = "Sep";
            break;
        case 9:
            monthS = "Oct";
            break;
        case 10:
            monthS = "Nov";
            break;
        case 11:
            monthS = "Dec";
            break;

    }

        return monthS;

    }



    /**
     * Method that generates a random city between Uppsala, Stockholm and Gothenburg
     *
     */

    private String randomCity() {

        String city = "";

        int num = (int) Math.floor(Math.random()*3);  //random number between 0 and 2

        switch(num) { //depending on the random number one of the options will be chosen.

            case 0:
                city = "Uppsala";
                break;
            case 1:
                city = "Stockholm";
                break;
            case 2:
                city = "Gothenburg";
                break;

        }

        return city;
    }

    /**
     * Method that prints a record for hybrid car
     * @param pw PrintWritter to write in the file
     */

    private void hybridRecord(PrintWriter pw) {

        String type = "H";

        int error = (int) Math.floor(Math.random()*100); // if we get 17 out of 100 possible random numbers it simulates an error in the
record.

        if(error == 17) {
            type= "---";
        }

        //number of cars that this family unit (represented by this record) has

        int cars = (int) Math.floor(Math.random()*2);

        if(cars == 1) {
            pw.print(randomCity()+"/"+type+"+"+type+"/"+randomDate(cars)+"/"+randomHybHps(cars)+"/"+randomKMs(cars));

        }else {
            pw.print(randomCity()+"/"+type+"/"+randomDate(cars)+"/"+randomHybHps(cars)+"/"+randomKMs(cars));

        }



    }

    /**
     * Method that generates a random number of cars (1 or 2) and a value of HP for each of them taking into account that hybrid cars
have 2 engines (i.e. 2 hps).
     * If 2 cars are to be written they would be separated by an @ symbol to simplify the map-reduce technique (1 pair key+value).
     * As they are hybrid and have 2 hps the 2 values would be concatenated using a + sign.
     * @param cars
     */

    private String randomHybHps(int cars) {

        String hps = "";

        if(cars == 1) {

            int h1 = (int) ((Math.random() * (300 - 60)) + 60);  //random number between 60 and 300
            int h1_1 = (int) ((Math.random() * (100 - 6)) + 6);  //random number between 1 and 100
            int h2 = (int) ((Math.random() * (300 - 60)) + 60);  //random number between 60 and 300
            int h2_1 = (int) ((Math.random() * (100 - 6)) + 6);  //random number between 1 and 100
```

```
            hps = String.valueOf(h1)+"+"+String.valueOf(h1_1)+"@"+String.valueOf(h2)+"+"+String.valueOf(h2_1);

        }else {

            int num = (int) ((Math.random() * (300 - 60)) + 60);  //random number between 60 and 300
            int num_1 = (int) ((Math.random() * (100 - 6)) + 6);  //random number between 1 and 100

            hps = String.valueOf(num)+"+"+String.valueOf(num_1);

        }

        return hps;
    }


}
```

EXECUTION RESULTS:

LOCAL

The first step was to do a mvn clean in order to erase any possible previous garbage.

```
mcelam00@mcelam00-vmware: ~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey
mcelam00@mcelam00-vmware:~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey$ mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------< com.hadoopbook:Hadoop-Practice >-------------------
[INFO] Building Hadoop-Practice 1.0
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ Hadoop-Practice ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.299 s
[INFO] Finished at: 2021-01-02T14:29:08+01:00
[INFO] ------------------------------------------------------------------------
mcelam00@mcelam00-vmware:~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey$
```

Then I proceeded to compile the program and solve the compilation errors I had.

```
mcelam00@mcelam00-vmware: ~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey
mcelam00@mcelam00-vmware:~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey$ mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------< com.hadoopbook:Hadoop-Practice >-------------------
[INFO] Building Hadoop-Practice 1.0
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ Hadoop-Practice ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/mcelam00/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey/src/m
ain/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ Hadoop-Practice ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 4 source files to /home/mcelam00/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey/target/classe
s
[WARNING] /home/mcelam00/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey/src/main/java/volvoSurveyDriver.java:
/home/mcelam00/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey/src/main/java/volvoSurveyDriver.java uses or ove
rrides a deprecated API.
[WARNING] /home/mcelam00/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey/src/main/java/volvoSurveyDriver.java:
Recompile with -Xlint:deprecation for details.
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.428 s
[INFO] Finished at: 2021-01-02T14:29:42+01:00
[INFO] ------------------------------------------------------------------------
mcelam00@mcelam00-vmware:~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey$
```

Having compiled it successfully, I need to make the .jar executable file that Hadoop needs to work with, so to do that, in maven, we just type the next command:

```
mcelam00@mcelam00-vmware: ~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey
mcelam00@mcelam00-vmware:~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey$ mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------< com.hadoopbook:Hadoop-Practice >-------------------
[INFO] Building Hadoop-Practice 1.0
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ Hadoop-Practice ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/mcelam00/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey/src/main/resour
ces
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ Hadoop-Practice ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ Hadoop-Practice ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/mcelam00/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey/src/test/resour
ces
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ Hadoop-Practice ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ Hadoop-Practice ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:2.5:jar (default-jar) @ Hadoop-Practice ---
[INFO] Building jar: /home/mcelam00/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey/hadoop-practice.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.076 s
[INFO] Finished at: 2021-01-02T14:32:42+01:00
[INFO] ------------------------------------------------------------------------
mcelam00@mcelam00-vmware:~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey$
```

Once the .jar is ready, I launch Hadoop, taking into consideration that the output directory must not exist.

```
mcelam00@mcelam00-vmware: ~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey
mcelam00@mcelam00-vmware:~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey$ hadoop jar hadoop-practice.jar
 -conf ./conf/hadoop-local.xml ./input/ ./output
```

Once finished the hadoop Map Reduce we obtain a summary and the results in the output directory.

```
mcelam00@mcelam00-vmware: ~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey
2021-01-02 14:42:04,800 INFO mapred.LocalJobRunner: Finishing task: attempt_local992602244_0001_r_000000_0
2021-01-02 14:42:04,800 INFO mapred.LocalJobRunner: reduce task executor complete.
2021-01-02 14:42:05,409 INFO mapreduce.Job:  map 100% reduce 100%
2021-01-02 14:42:05,410 INFO mapreduce.Job: Job job_local992602244_0001 completed successfully
2021-01-02 14:42:05,433 INFO mapreduce.Job: Counters: 30
        File System Counters
                FILE: Number of bytes read=12616567814
                FILE: Number of bytes written=14530895
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
        Map-Reduce Framework
                Map input records=20000000
                Map output records=9481689
                Map output bytes=129584932
                Map output materialized bytes=1378
                Input split bytes=3770
                Combine input records=9481689
                Combine output records=78
                Reduce input groups=3
                Reduce shuffle bytes=1378
                Reduce input records=78
                Reduce output records=3
                Spilled Records=156
                Shuffled Maps =26
                Failed Shuffles=0
                Merged Map outputs=26
                GC time elapsed (ms)=922
                Total committed heap usage (bytes)=51959037952
        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=854381287
        File Output Format Counters
                Bytes Written=53
mcelam00@mcelam00-vmware:~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey$
```

```
Open    part-r-00000    Save
~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey/output
1 Gothenburg      211
2 Stockholm       211
3 Uppsala         211
```

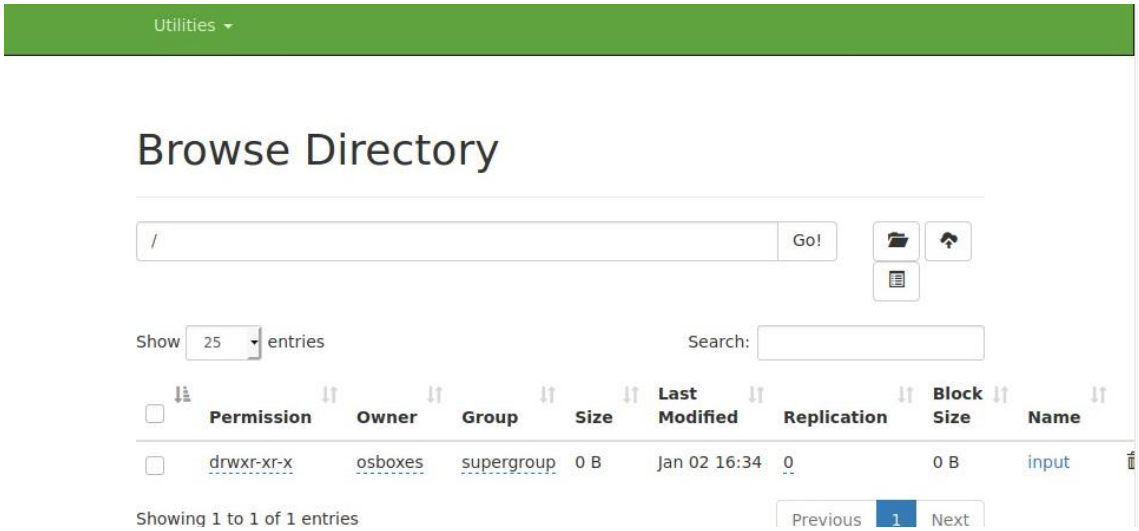
A more detailed analysis of this results is carried out in the point conclusions.

PSEUDO-CLUSTER

The execution in the pseudo-cluster shares the same steps in what comprises the creation of the .jar; so, to make a long story short, I will continue from this point since I have already come across the jar in the previous part.

In order to execute it in pseudo-cluster first of all I need to create the input directory in HDFS:

```
osboxes@osboxes:/usr/local/hadoop$ bin/hadoop fs -mkdir /input
osboxes@osboxes:/usr/local/hadoop$
```

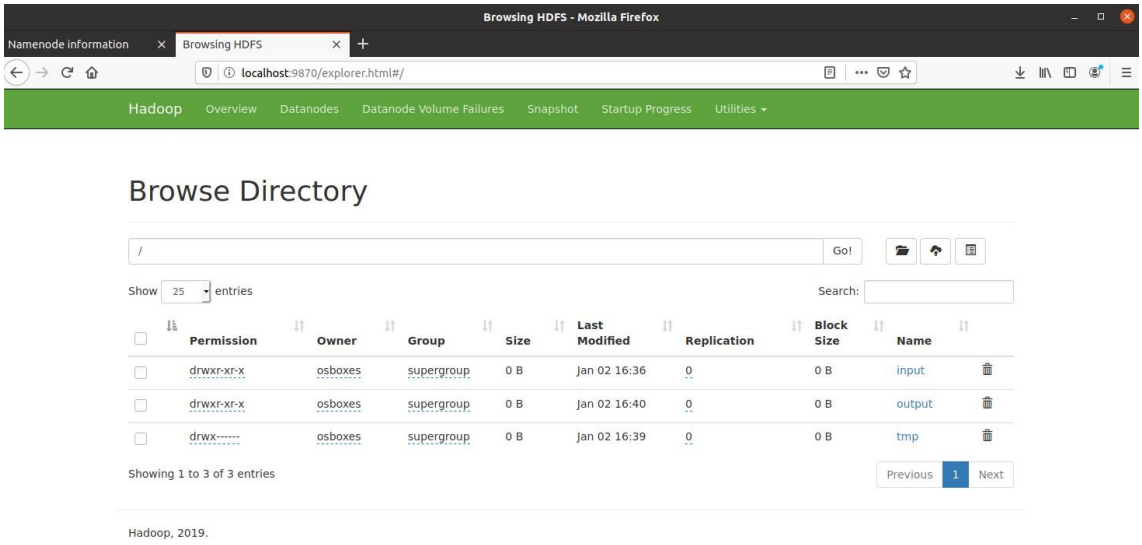


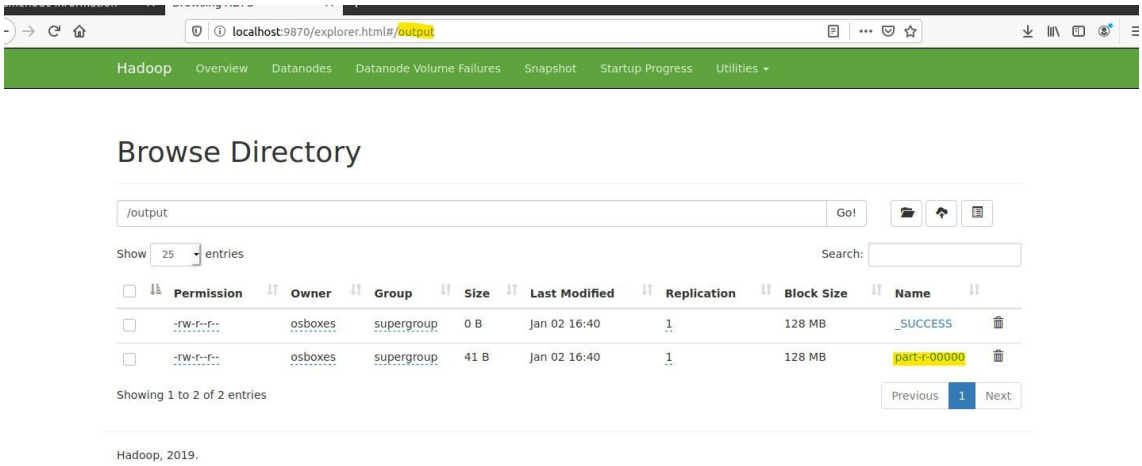
Then, I need to introduce in the input directory the input text file just like I have in the local structure. To do so, I issue the next command:

```
osboxes@osboxes:/usr/local/hadoop$ bin/hadoop fs -copyFromLocal /home/osboxes/Downloads/Hado
op-Volvo-main/hadoopVolvoSurvey/input/* /input
2021-01-02 16:36:25,635 INFO sasl.SaslDataTransferClient: SASL encryption trust check: local
HostTrusted = false, remoteHostTrusted = false
2021-01-02 16:36:27,631 INFO sasl.SaslDataTransferClient: SASL encryption trust check: local
HostTrusted = false, remoteHostTrusted = false
2021-01-02 16:36:29,077 INFO sasl.SaslDataTransferClient: SASL encryption trust check: local
HostTrusted = false, remoteHostTrusted = false
2021-01-02 16:36:30,410 INFO sasl.SaslDataTransferClient: SASL encryption trust check: local
HostTrusted = false, remoteHostTrusted = false
2021-01-02 16:36:31,829 INFO sasl.SaslDataTransferClient: SASL encryption trust check: local
HostTrusted = false, remoteHostTrusted = false
2021-01-02 16:36:33,812 INFO sasl.SaslDataTransferClient: SASL encryption trust check: local
HostTrusted = false, remoteHostTrusted = false
2021-01-02 16:36:34,847 INFO sasl.SaslDataTransferClient: SASL encryption trust check: local
HostTrusted = false, remoteHostTrusted = false
osboxes@osboxes:/usr/local/hadoop$
```

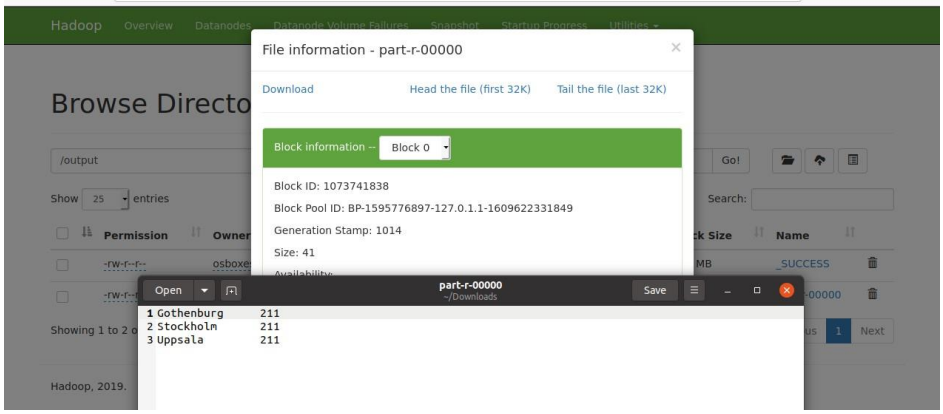
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	osboxes	supergroup	814.75 MB	Jan 02 16:36	1	128 MB	prueba.txt

Once done, we can execute the Hadoop Map Reduce in the pseudo-cluster

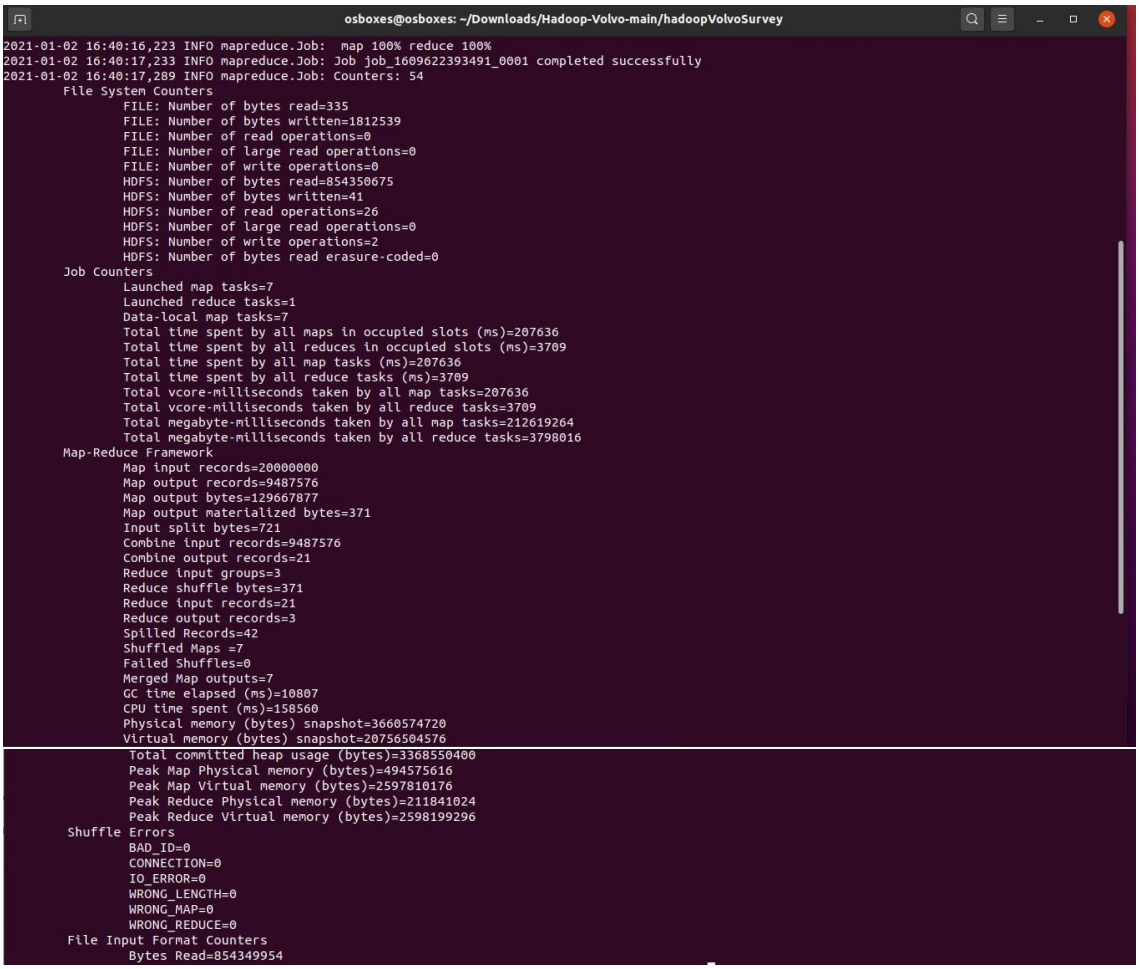




As we can see, if we examine the directories via the web browser, we can notice the output directory and if we click on it, we will see the output file resulting from the execution in HDFS and if we download it and open it, we can see that the resulting values are the same as when we executed the Map Reduce locally.



However, the way and the amount of displayed information while the process was running was not as big as with the local execution, where we obtained various summaries and more details.



```
osboxes@osboxes: ~/Downloads/Hadoop-Volvo-main/hadoopVolvoSurvey
2021-01-02 16:40:16,223 INFO mapreduce.Job:  map 100% reduce 100%
2021-01-02 16:40:17,233 INFO mapreduce.Job: Job job_1609622393491_0001 completed successfully
2021-01-02 16:40:17,289 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=335
                FILE: Number of bytes written=1812539
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=854350675
                HDFS: Number of bytes written=41
                HDFS: Number of read operations=26
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=7
                Launched reduce tasks=1
                Data-local map tasks=7
                Total time spent by all maps in occupied slots (ms)=207636
                Total time spent by all reduces in occupied slots (ms)=3709
                Total time spent by all map tasks (ms)=207636
                Total time spent by all reduce tasks (ms)=3709
                Total vcore-milliseconds taken by all map tasks=207636
                Total vcore-milliseconds taken by all reduce tasks=3709
                Total megabyte-milliseconds taken by all map tasks=212619264
                Total megabyte-milliseconds taken by all reduce tasks=3798016
        Map-Reduce Framework
                Map input records=20000000
                Map output records=9487576
                Map output bytes=129667877
                Map output materialized bytes=371
                Input split bytes=721
                Combine input records=9487576
                Combine output records=21
                Reduce input groups=3
                Reduce shuffle bytes=371
                Reduce input records=21
                Reduce output records=3
                Spilled Records=42
                Shuffled Maps =7
                Failed Shuffles=0
                Merged Map outputs=7
                GC time elapsed (ms)=10807
                CPU time spent (ms)=158560
                Physical memory (bytes) snapshot=3660574720
                Virtual memory (bytes) snapshot=20756504576
                Total committed heap usage (bytes)=3368550400
                Peak Map Physical memory (bytes)=494575616
                Peak Map Virtual memory (bytes)=2597810176
                Peak Reduce Physical memory (bytes)=211841024
                Peak Reduce Virtual memory (bytes)=2598199296
        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=854349954
```

These values will be analysed along with the ones dumped with the local execution in the next point.

CONCLUSIONS:

EXECUTION COUNTER SECTIONS ANALYSIS: LOCAL

```
File System Counters
        FILE: Number of bytes read=12616567814
        FILE: Number of bytes written=14530895
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
```

First of all, we see "FILE:" indicating that the bytes are read from a local file. Following them we have several different counters:

Number of bytes read: Tells the number of bytes read by both Map and Reduce tasks.

Number of bytes written: Displays the number of bytes written by both Map and Reduce tasks.

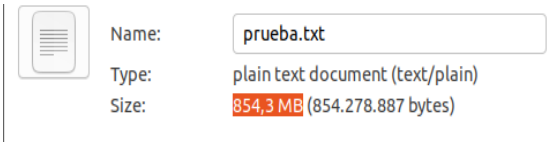
Number of read operations: Shows the number of read operations by both Map and Reduce tasks.

Number of large read operations: Displays the number of large read operations (e.g. moving in a directory tree) for both Map and Reduce tasks.

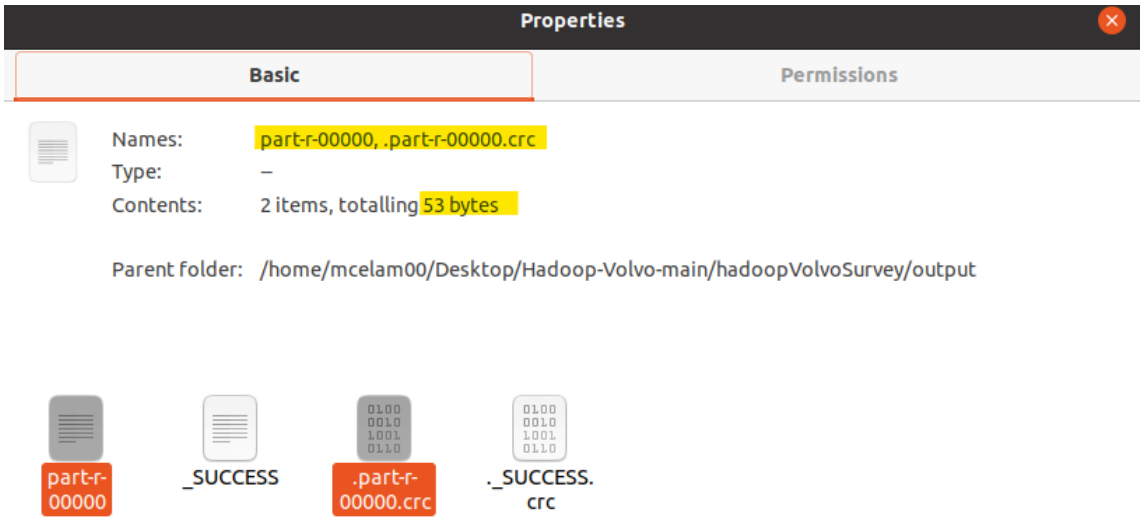
Number of write operations: Tells the number of write operations by both Map and Reduce tasks (e.g. creating a file or append new data to it).

```
File Input Format Counters
        Bytes Read=854381287
File Output Format Counters
        Bytes Written=53
mcelam00@mcelam00-vmware:~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey$
```

The File Input Format Counters, refer to the amount of bytes which are read by the Map task, in my case is the size of the input file.



The File Output Format Counters, refer to the amount of bytes which are written by the Map and Reduce tasks, in my case belongs to the size of the file containing the output values plus a hidden file related to this previous one.


Properties
Basic
Permissions
Names: part-r-00000, .part-r-00000.crc
Type: –
Contents: 2 items, totalling 53 bytes
Parent folder: /home/mcelam00/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey/output
part-r-00000
_SUCCESS
.part-r-00000.crc
._SUCCESS.crc

EXECUTION COUNTER SECTIONS ANALYSIS: PSEUDO-CLUSTER

In this case as we may see, the summary is a bit different, we have one more type of counters compared with the local execution that are the Job Counters and also some different records for the others.

```
        File System Counters
                FILE: Number of bytes read=335
                FILE: Number of bytes written=1812539
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=854350675
                HDFS: Number of bytes written=41
                HDFS: Number of read operations=26
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
```

In this case, we can see a main difference, that is, the presence of “HDFS” along with “FILE”. We can note that in this case the bytes read by both tasks from a local FILE are very very small and so they are the written compared with the precious. If we look at the HDFS we can see that here is where it reads the file (which makes sense since we uploaded it in the HDFS) and that the amount of bytes written in this case is exactly the size of the output values, no hidden files related are present. Finally, we see 26 read operations in HDFS (such as opening a file) and 2 write operations (such as creating files).

```
        Job Counters
                Launched map tasks=7
                Launched reduce tasks=1
                Data-local map tasks=7
                Total time spent by all maps in occupied slots (ms)=207636
                Total time spent by all reduces in occupied slots (ms)=3709
                Total time spent by all map tasks (ms)=207636
                Total time spent by all reduce tasks (ms)=3709
                Total vcore-milliseconds taken by all map tasks=207636
                Total vcore-milliseconds taken by all reduce tasks=3709
                Total megabyte-milliseconds taken by all map tasks=212619264
                Total megabyte-milliseconds taken by all reduce tasks=3798016
```

In these new counters we see that 7 map tasks were launched, and 1 reducer task. We also notice that 7 map tasks run on the same node where the data resided, that means all of them run on the node containing the data. Also, several time records are shown one for maps and other for reduce with the same values 2 to 2 and vcore time values representing the seconds in which Hadoop allocate vcores for the task. Finally, also data transfer rates are shown.

```
        File Input Format Counters
                Bytes Read=854349954
```

As for the local execution here, we have the total size of the input file, representing the read bytes by the map task.

RESULTS ANALYSIS

It seems quite shocking that the mean of Horse Powers is the same for the three cities, but there is a reason for this to happen, related with my file.

```
Open   part-r-00000   ~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey/output   Save
1 Gothenburg      211
2 Stockholm       211
3 Uppsala         211
```

If we recall the point in which I made calculations to explain the overall working of my mapper we saw that the results weren´t the same at all, neither they are if we take a smaller file with fewer number of records.

The problem comes from my input generator program. This program, that I made to simulate the records of the survey in the file, generates for each type of car random values of records and specifically for Horse Powers but, since the random ranges are the same for the calculations of the three cities, they are completely overpassed by the number of records requested in the file, ending with every single combination possible being registered for each city, and so, every city has in the end the same mean.

Furthermore, I tested what I am saying by doing MapReduce with smaller and smaller, and the smaller the file, the more the variations.

```
199994 Stockholm/H/Sep2006/153+92/299682
199995 Uppsala/E/Jun2007/102/148416
199996 Gothenburg/H/Sep2002/248+58/209072
199997 Gothenburg/E+E/Aug1996@Jun2009/110@261/10
199998 Stockholm/H/Mar1999/167+82/348732
199999 Uppsala/C/Jul1998/216/364591
200000 Uppsala/H/Oct1994/105+84/53767
```

```
Open   part-r-00000   ~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey/output   Save
1 Gothenburg      212
2 Stockholm       211
3 Uppsala         212
```

Successive executions each with less file lines lead to a more variation in the result, since no such many combinations are achieved in the file.

```
10995 Gothenburg/E/Oct2004/138/51086
10996 Uppsala/H+H/Feb1990@Sep1999/64+32@215+71/268429&406129
10997 Uppsala/E+C/Jan1995@May2020/274@239/2251428&404178
10998 Stockholm/H+H/Dec2007@Jun2001/284+59@174+68/39741&---
10999 Uppsala/C/Mar2018/129/123196
11000 Gothenburg/H/Aug1992/269+8/188567
                                                    Plain Text
```

```
Open   part-r-00000   ~/Desktop/Hadoop-Volvo-main/hadoopVolvoSurvey/output   Save
1 Gothenburg      212
2 Stockholm       206
3 Uppsala         214
```

And the example one present in page 7.

ERRORS OR PROBLEMS FOUND:

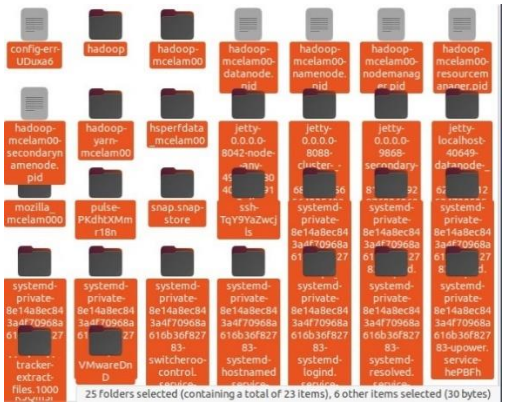
The first error I came across with was that, whenever I tried to start the jps processes always one or more are missing and not starting

```
mcelam00@mcelam00-vmware:/usr/local/hadoop$ jps
13623 Jps
12727 ResourceManager
12506 SecondaryNameNode
12893 NodeManager
12302 DataNode
mcelam00@mcelam00-vmware:/usr/local/hadoop$
```

If I try to Access to any of the commands of the pseudo-cluster, I have no connection whatsoever. I came across this issue for the first time when using the Ubuntu Server virtual machine, it happened the same, and so, I decided to switch to an Ubuntu machine. In Ubuntu, it worked perfectly well the same day I installed it, but later, I am exactly like in the Ubuntu server.

```
mcelam00@mcelam00-vmware:/usr/local/hadoop$ hadoop fs -mkdir /input
mkdir: Call From mcelam00-vmware/127.0.1.1 to localhost:9000 failed on connection ex
ception: java.net.ConnectException: Connection refused; For more details see:  http:
//wiki.apache.org/hadoop/ConnectionRefused
mcelam00@mcelam00-vmware:/usr/local/hadoop$
```

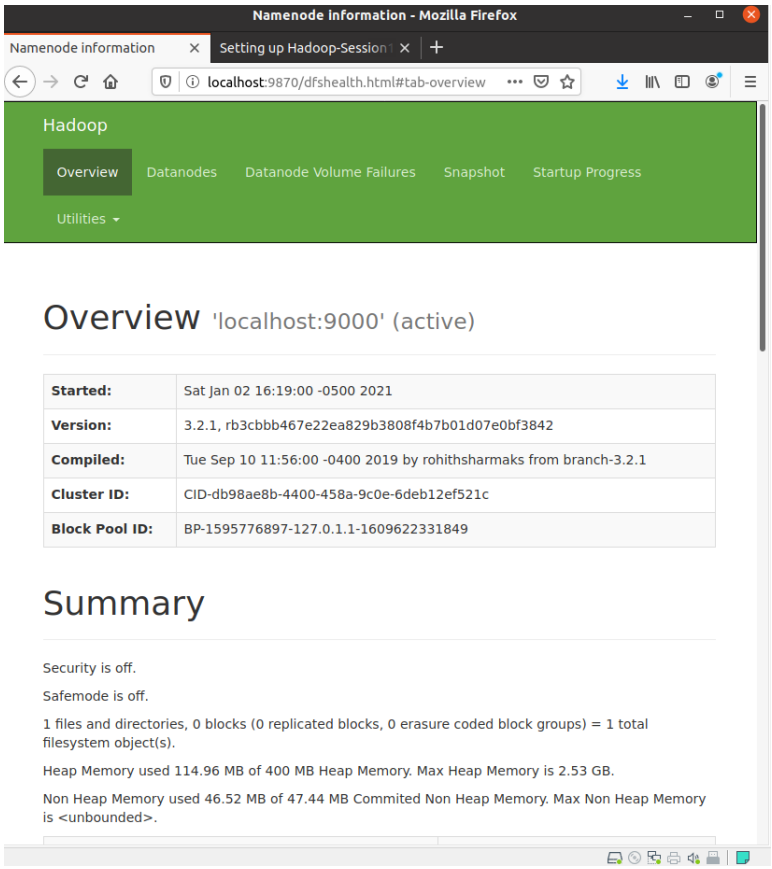
To try and solve it, I looked in internet and I attempted to re-format the namenode:

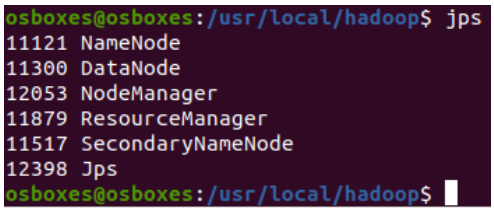


I erased the tmp directory and this time when trying to start them I already got an error

```
mcelam00@mcelam00-vmware:/usr/local/hadoop$ ./sbin/start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
localhost: ERROR: Cannot set priority of datanode process 14334
Starting secondary namenodes [mcelam00-vmware]
```

I decided to throw everything away and re-install Hadoop in a fresh new virtual machine. In that way I made it work again as it was supposed to:





In the new virtual box, the Hadoop command was not recognised nor even after indicating the path:

```
osboxes@osboxes:/usr/local/hadoop$ hadoop fs -mkdir /input
hadoop: command not found
```

So, I had to use the full path when issuing the commands.

```
osboxes@osboxes:/usr/local/hadoop$ bin/hadoop fs -mkdir /input
```

REFERENCES:

https://knpcode.com/hadoop/mapreduce/counters-in-hadoop-mapreduce/

https://www.netjstech.com/2018/07/what-are-counters-in-hadoop-mapreduce.html -:~:text=File%20System%20Counters%20in%20MapReduce,-File%20system%20counters&text=As%20example%20FILE%3A%20Number%20of,both%20Map%20and%20Reduce%20tasks.

https://hadoop.apache.org/docs/current/api/org/apache/hadoop/yarn/api/records/ApplicationResourceUsageReport.html - getVcoreSeconds()

https://stackoverflow.com/questions/31964833/what-does-vcore-seconds-in-hadoop-job-log-mean

http://www.cafeaulait.org/course/week2/43.html

https://stackoverflow.com/questions/2333618/hadoop-one-map-and-multiple-reduce

http://chuwiki.chuidiang.org/index.php?title=Generar_n%C3%BAmeros_aleatorios_en_Java

https://www.baeldung.com/java-generating-random-numbers-in-range

https://www.educative.io/edpresso/how-to-convert-an-integer-to-a-string-in-java

https://stackoverflow.com/questions/2333618/hadoop-one-map-and-multiple-reduce

https://stackoverflow.com/questions/28809143/how-to-implement-multiple-reducers-in-a-single-mapreduce-job/28863048

https://stackoverflow.com/questions/3481828/how-to-split-a-string-in-java

https://stackoverflow.com/questions/5389200/what-is-a-java-strings-default-initial-value

https://www.codejava.net/java-core/the-java-language/java-default-initialization-of-instance-variables-and-initialization-blocks

https://techlandia.com/desplazarse-traves-variable-string-java-info_231087/

https://stackoverflow.com/questions/5034580/comparing-chars-in-java

https://www.w3schools.com/jsref/jsref_substring.asp