Universidad de Granada

Master Profesional en Ingeniería Informática

Práctica 2

Hadoop

Autor:
Manuel Jesús García Manday
(nickter@correo.ugr.es)

Master en Ingeniería Informática

20 de mayo de 2017

Índice

1. Objetivo.

El objetivo de esta práctica es realizar programas escalables para mejorar la eficiencia en entornos Big Data.

2. Introduccíon.

Para comenzar a realizar las tareas que se piden en esta práctica, es necesario en primer lugar realizar una serie de pasos iniciales que se describen a continuación.

Realizamos una conexión remota hacía el servidor **haddop.ugr.es** y una vez dentro creamos una carpeta nueva donde descargaremos el código Java de los programas. Comprobamos tambien que los datos de entrada se encuentran disponibles.

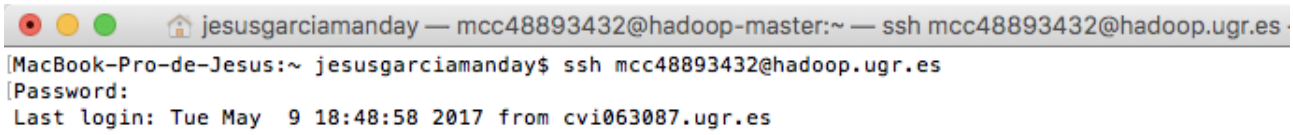


Figura 1: Conexion remota a **hadoop.ugr.es**.

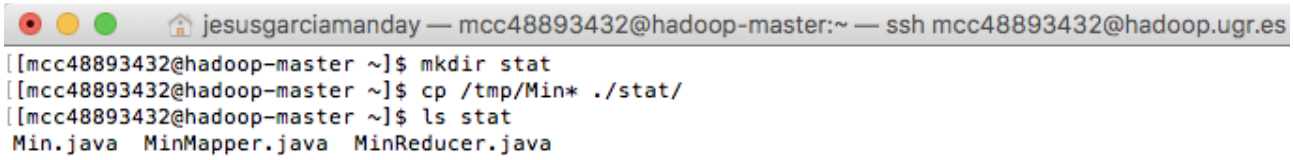


Figura 2: Copiando los ficheros Java.

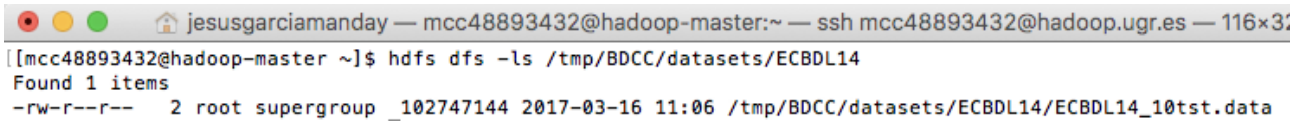


Figura 3: Datos de entrada.

Viendo que están disponibles, ahora nos creamos un directorio local para las clases de java.

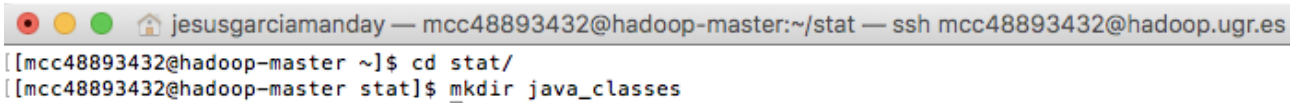


Figura 4: Directorio local.

Con la preconfiguración realizada pasamos a realizar las diferentes tareas que es exponen en la práctica.

3. Calcula el valor mínimo de la variable (columna) 5.

La primera de ellas es calcular el mínimo sobre el conjunto de valores del dataset, por lo que una vez que tenemos todos los ficheros java correspondientes, ahora toca compilarlos para crear el fichero **.jar** a continuación y ejecutarlo en **hadoop**.

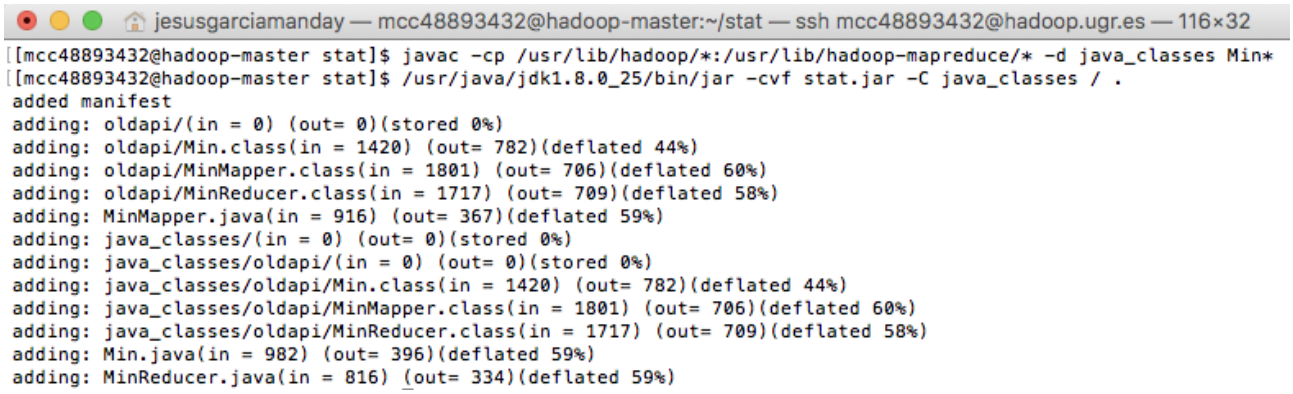


Figura 5: Compilamos y ejecutamos (I).

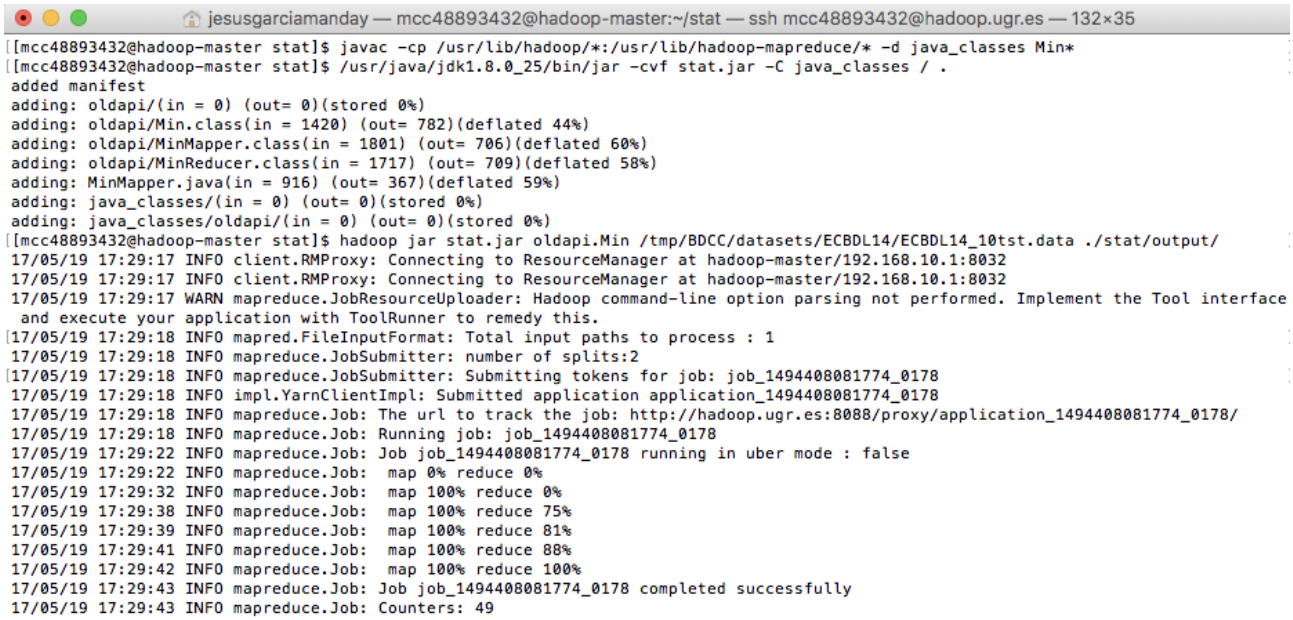


Figura 6: Compilamos y ejecutamos (II).

```
File System Counters
        FILE: Number of bytes read=2142847
        FILE: Number of bytes written=6470142
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=102749934
        HDFS: Number of bytes written=8
        HDFS: Number of read operations=54
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=32
Job Counters
        Launched map tasks=2
        Launched reduce tasks=16
        Rack-local map tasks=2
        Total time spent by all maps in occupied slots (ms)=112574
        Total time spent by all reduces in occupied slots (ms)=1894977
        Total time spent by all map tasks (ms)=16082
        Total time spent by all reduce tasks (ms)=38673
        Total vcore-seconds taken by all map tasks=16082
        Total vcore-seconds taken by all reduce tasks=38673
        Total megabyte-seconds taken by all map tasks=112574000
        Total megabyte-seconds taken by all reduce tasks=1933650000
```

Figura 7: Compilamos y ejecutamos (III).

```
Map-Reduce Framework
        Map input records=2897917
        Map output records=2897917
        Map output bytes=28979170
        Map output materialized bytes=2143005
        Input split bytes=234
        Combine input records=0
        Combine output records=0
        Reduce input groups=1
        Reduce shuffle bytes=2143005
        Reduce input records=2897917
        Reduce output records=1
        Spilled Records=5795834
        Shuffled Maps =32
        Failed Shuffles=0
        Merged Map outputs=32
        GC time elapsed (ms)=347
        CPU time spent (ms)=37010
        Physical memory (bytes) snapshot=7926947840
        Virtual memory (bytes) snapshot=984134000640
        Total committed heap usage (bytes)=19421724672
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=102749700
File Output Format Counters
        Bytes Written=8
```

Figura 8: Compilamos y ejecutamos (IV).

Por último comprobamos el resultado para ver si se ha realizado correctamente.

```
jesusgarciamanday — mcc48893432@hadoop-master:~/stat — ssh mcc48893432@hadoop.ugr.es
[mcc48893432@hadoop-master stat]$ hdfs dfs -cat stat/output/*
1       -11.0
```

Figura 9: Comprobando resultado.

4. Calcula el valor máximo de la variable (columna) 5.

Para calcular el valor máximo de la variable (columna) 5 vamos a crear clases de java correspondientes para hacer dicha operación. Comenzaremos por crear la clase **MaxMapper** con su fichero java correspondiente.

```
jesusgarciamanday — mcc48893432@hadoop-master:~/stat — ssh mcc48893432@hadoop.ugr.es — 132×35
 GNU nano 2.0.9                    File: MaxMapper.java                              Modified

package oldapi;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
public class MaxMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, DoubleWritable> {
        private static final int MISSING = 0;
        public static int col=5;

                public void map(LongWritable key, Text value, OutputCollector<Text, DoubleWritable> output, Reporter reporter) thro$
                String line = value.toString();
                String[] parts = line.split(",");
                output.collect(new Text("1"), new DoubleWritable(Double.parseDouble(parts[col])));
        }
}
```

Figura 10: Clase MaxMapper.

A continuación creamos la clase correspondientes para el Reducer (**MaxReducer**) y la clase principal **Max** donde estará el main.

```
jesusgarciamanday — mcc48893432@hadoop-master:~/stat — ssh mcc48893432@hadoop.ugr.es — 132×35
 GNU nano 2.0.9                    File: MaxReducer.java                             Modified

package oldapi;
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
public class MaxReducer extends MapReduceBase implements Reducer<Text, DoubleWritable, Text, DoubleWritable> {


        public void reduce(Text key, Iterator<DoubleWritable> values, OutputCollector<Text, DoubleWritable> output, Reporter repor
                Double maxValue = Double.MIN_VALUE;
                while (values.hasNext()) {
                        maxValue = Math.max(maxValue, values.next().get());
                }
        output.collect(key, new DoubleWritable(maxValue));
        }
}
```

Figura 11: Clase MaxReducer.

jesusgarciamanday — mcc48893432@hadoop-master:~/stat — ssh mcc48893432@hadoop.ugr.es — 132×35

GNU nano 2.0.9 File: Max.java Modified

```
package oldapi;
import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
public class Max {
        public static void main(String[] args) throws IOException {
                if (args.length != 2) {
                        System.err.println("Usage: MaxTemperature <input path> <output path>");
                        System.exit(-1);
                }
                JobConf conf = new JobConf(Max.class);
                conf.setJobName("Max temperature");
                FileInputFormat.addInputPath(conf, new Path(args[0]));
                FileOutputFormat.setOutputPath(conf, new Path(args[1]));
                conf.setMapperClass(MaxMapper.class);
                conf.setReducerClass(MaxReducer.class);
                conf.setOutputKeyClass(Text.class);
                conf.setOutputValueClass(DoubleWritable.class);
                JobClient.runJob(conf);
        }
}
```

Figura 12: Clase Max.

Una vez que tenemos todos los ficheros procedemos a realizar los mismos pasos que con el ejercicio anterior compilando las clases, creando el fichero jar y ejecutandolo en **hadoop**.

jesusgarciamanday — mcc48893432@hadoop-master:~/stat — ssh mcc48893432@hadoop.ugr.es — 132×35

```
[mcc48893432@hadoop-master stat]$ javac -cp /usr/lib/hadoop/*:/usr/lib/hadoop-mapreduce/* -d java_classes Max*
[mcc48893432@hadoop-master stat]$ /usr/java/jdk1.8.0_25/bin/jar -cvf stat.jar -C java_classes / .
added manifest
adding: oldapi/(in = 0) (out= 0)(stored 0%)
adding: oldapi/MaxReducer.class(in = 1717) (out= 706)(deflated 58%)
adding: oldapi/Max.class(in = 1420) (out= 781)(deflated 45%)
adding: oldapi/MaxMapper.class(in = 1801) (out= 705)(deflated 60%)
adding: MaxReducer.java(in = 816) (out= 334)(deflated 59%)
adding: java_classes/(in = 0) (out= 0)(stored 0%)
adding: java_classes/oldapi/(in = 0) (out= 0)(stored 0%)
adding: java_classes/oldapi/MaxReducer.class(in = 1717) (out= 706)(deflated 58%)
adding: java_classes/oldapi/Max.class(in = 1420) (out= 781)(deflated 45%)
adding: java_classes/oldapi/MaxMapper.class(in = 1801) (out= 705)(deflated 60%)
adding: Max.java(in = 981) (out= 395)(deflated 59%)
adding: MaxMapper.java(in = 913) (out= 366)(deflated 59%)
```

Figura 13: Compilamos y ejecutamos (I).

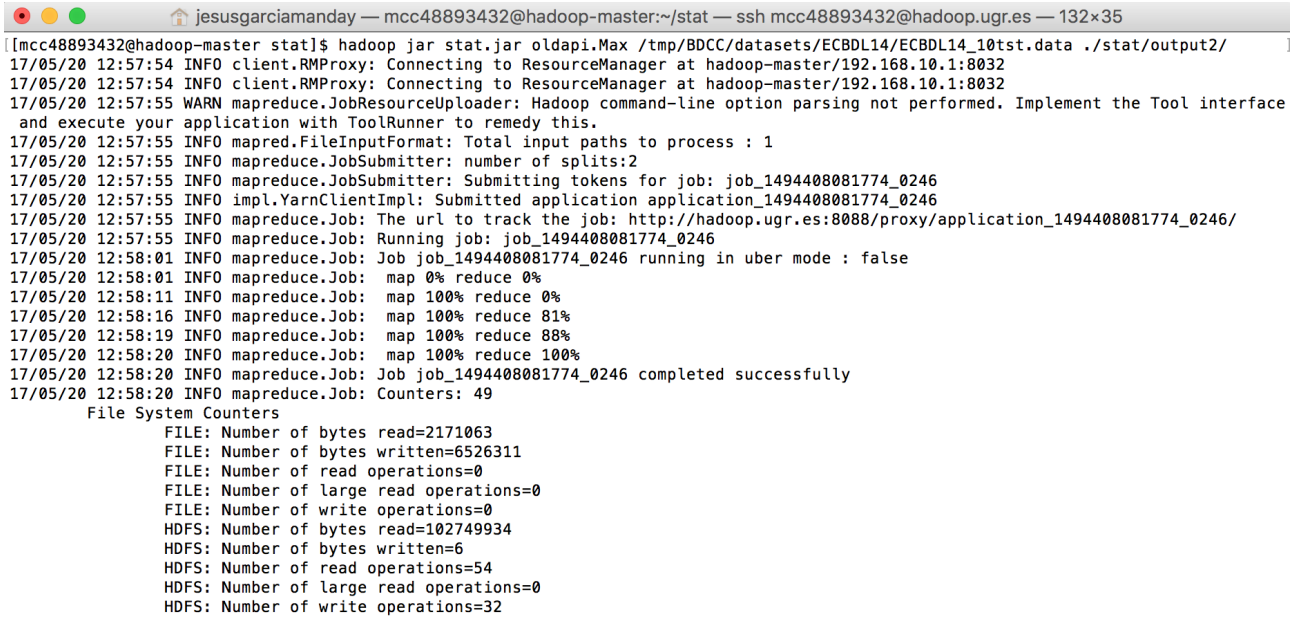


Figura 14: Compilamos y ejecutamos (II).

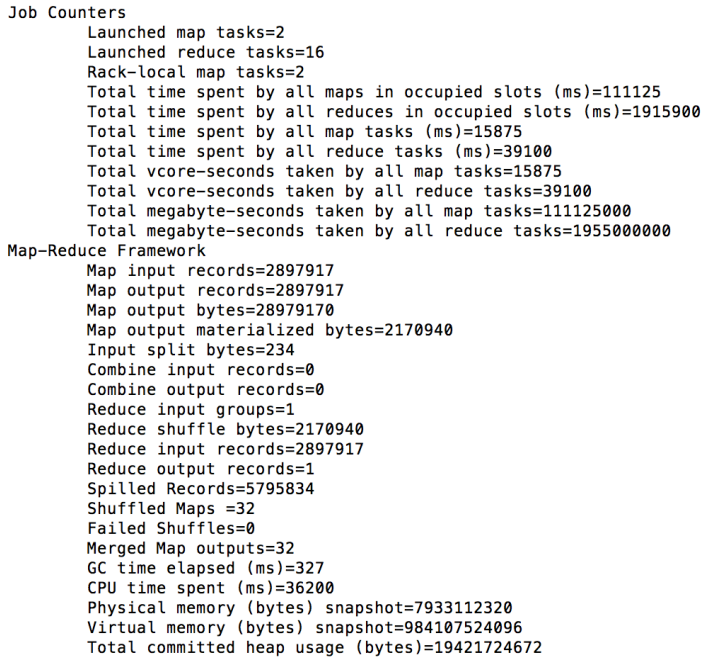


Figura 15: Compilamos y ejecutamos (III).

```
        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=102749700
        File Output Format Counters
                Bytes Written=6
```

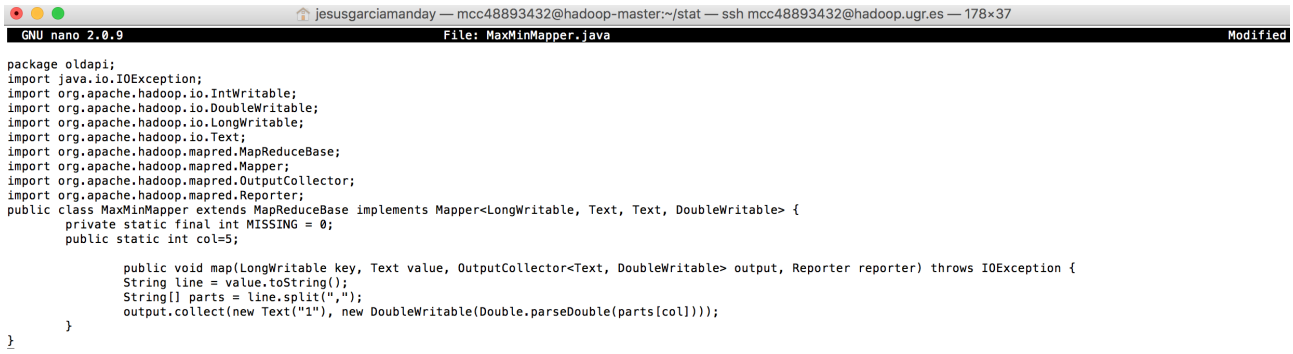
Figura 16: Compilamos y ejecutamos (IV).

Comprobamos que el resultado nos arroja el valor máximo.

Figura 17: Comprobando resultado.

5. Calcula al mismo tiempo los valores máximo y mínimo de la variable 5.

Al igual que para los anteriores cálculos creamos los ficheros correspondientes, aunque el fichero con la función **Mapper** y el principal con el main solo cambia el nombre, en la clase con la función **Reducer** es donde se realizan los cambios necesarios para obtener los resultados esperados.




```
package oldapi;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
public class MaxMinMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, DoubleWritable> {
        private static final int MISSING = 0;
        public static int col=5;

                public void map(LongWritable key, Text value, OutputCollector<Text, DoubleWritable> output, Reporter reporter) throws IOException {
                String line = value.toString();
                String[] parts = line.split(",");
                output.collect(new Text("1"), new DoubleWritable(Double.parseDouble(parts[col])));
        }
}
```

Figura 18: Clase MaxMinMapper.

jesusgarciamanday — mcc48893432@hadoop-master:~/stat — ssh mcc48893432@hadoop.ugr.es — 156×36

GNU nano 2.0.9 File: MaxMin.java Modified

```
package oldapi;
import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
public class MaxMin {
        public static void main(String[] args) throws IOException {
                if (args.length != 2) {
                        System.err.println("Usage: MaxMinTemperature <input path> <output path>");
                        System.exit(-1);
                }
                JobConf conf = new JobConf(MaxMin.class);
                conf.setJobName("MaxMin temperature");
                FileInputFormat.addInputPath(conf, new Path(args[0]));
                FileOutputFormat.setOutputPath(conf, new Path(args[1]));
                conf.setMapperClass(MaxMinMapper.class);
                conf.setReducerClass(MaxMinReducer.class);
                conf.setOutputKeyClass(Text.class);
                conf.setOutputValueClass(DoubleWritable.class);
                JobClient.runJob(conf);
        }
}
```

Figura 19: Clase MaxMin.

jesusgarciamanday — mcc48893432@hadoop-master:~/stat — ssh mcc48893432@hadoop.ugr.es — 178×37

GNU nano 2.0.9 File: MaxMinReducer.java Modified

```
package oldapi;
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
public class MaxMinReducer extends MapReduceBase implements Reducer<Text, DoubleWritable, Text, DoubleWritable> {


        public void reduce(Text key, Iterator<DoubleWritable> values, OutputCollector<Text, DoubleWritable> output, Reporter reporter) throws IOException {
                Double maxValue = Double.MIN_VALUE;
                Double minValue = Double.MAX_VALUE;
                Double value;

                while (values.hasNext()) {
                        value = values.next().get();
                        maxValue = Math.max(maxValue, value);
                        minValue = Math.min(minValue, value);
                }
                output.collect(key, new DoubleWritable(maxValue));
                output.collect(key, new DoubleWritable(minValue));
        }
}
```

Figura 20: Clase MaxMinReducer.

Ahora toca compilarlos para adjuntarlos en el fichero **jar** y realizar la ejecución en **hadoop**.

jesusgarciamanday — mcc48893432@hadoop-master:~/stat — ssh mcc48893432@hadoop.ugr.es

```
[mcc48893432@hadoop-master stat]$ javac -cp /usr/lib/hadoop/*:/usr/lib/hadoop-mapreduce/* -d java_classes MaxMin*
[mcc48893432@hadoop-master stat]$ /usr/java/jdk1.8.0_25/bin/jar -cvf stat.jar -C java_classes / .
added manifest
adding: oldapi/(in = 0) (out= 0)(stored 0%)
adding: oldapi/MaxMinMapper.class(in = 1807) (out= 708)(deflated 60%)
adding: oldapi/MaxMinReducer.class(in = 1815) (out= 749)(deflated 58%)
adding: oldapi/MaxMin.class(in = 1438) (out= 787)(deflated 45%)
adding: MaxMin.java(in = 999) (out= 398)(deflated 60%)
adding: MaxMinReducer.java(in = 963) (out= 360)(deflated 62%)
adding: java_classes/(in = 0) (out= 0)(stored 0%)
adding: java_classes/oldapi/(in = 0) (out= 0)(stored 0%)
adding: java_classes/oldapi/MaxMinMapper.class(in = 1807) (out= 708)(deflated 60%)
adding: java_classes/oldapi/MaxMinReducer.class(in = 1815) (out= 749)(deflated 58%)
adding: java_classes/oldapi/MaxMin.class(in = 1438) (out= 787)(deflated 45%)
adding: MaxMinMapper.java(in = 916) (out= 369)(deflated 59%)
```

Figura 21: Compilamos y ejecutamos (I).

Figura 22: Compilamos y ejecutamos (II).

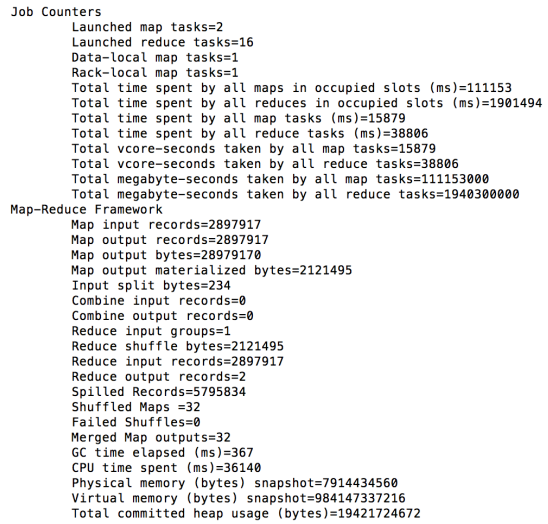


Figura 23: Compilamos y ejecutamos (III).

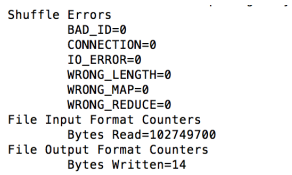


Figura 24: Compilamos y ejecutamos (IV).

Por último comprobamos el resultado obtenido.

```
jesusgarciamanday — mcc48893432@hadoop-master:~/stat — ssh mcc48893432@hadoop.ugr.es
[mcc48893432@hadoop-master stat]$ hdfs dfs -cat stat/output5/*
1       9.0
1       -11.0
```

Figura 25: Comprobando resultado.

6. Calcula los valores máximo y mínimo de todas las variables (salvo la última, que es la etiqueta de la clase).

Para este cálculo solo necesitaremos adaptar la clase con la función **Mapper** para cada columna ya que el **Reducer** es el mismo. Ahora la función **Mapper** asignará diferentes key, uno por cada columna.

```
jesusgarciamanday — mcc48893432@hadoop-master:~/stat — ssh mcc48893432@hadoop.ugr.es — 132×35
 GNU nano 2.0.9                    File: MaxMinMapper.java                              Modified

package oldapi;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
public class MaxMinMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, DoubleWritable> {
        private static final int MISSING = 0;
        public static int col=5;

                public void map(LongWritable key, Text value, OutputCollector<Text, DoubleWritable> output, Reporter reporter) thro$
                String line = value.toString();
                String[] parts = line.split(",");
                for(int i = 1; i < parts.length; i++){
                        output.collect(new Text(String.valueOf(i)), new DoubleWritable(Double.parseDouble(parts[i-1])));
                }
        }
}
```

Figura 26: Clase MaxMinMapper.

Como en los anteriores ejercicios, compilamos y ejecutamos las clases.

```
jesusgarciamanday — mcc48893432@hadoop-master:~/stat — ssh mcc48893432@hadoop.ugr.es — 132×35
[mcc48893432@hadoop-master stat]$ javac -cp /usr/lib/hadoop/*:/usr/lib/hadoop-mapreduce/* -d java_classes MaxMin*
[mcc48893432@hadoop-master stat]$ /usr/java/jdk1.8.0_25/bin/jar -cvf stat.jar -C java_classes / .
added manifest
adding: oldapi/(in = 0) (out= 0)(stored 0%)
adding: oldapi/MaxMinMapper.class(in = 1938) (out= 790)(deflated 59%)
adding: oldapi/MaxMinReducer.class(in = 1824) (out= 762)(deflated 58%)
adding: oldapi/MaxMin.class(in = 1438) (out= 787)(deflated 45%)
adding: MaxMin.java(in = 999) (out= 398)(deflated 60%)
adding: MaxMinReducer.java(in = 986) (out= 372)(deflated 62%)
adding: java_classes/(in = 0) (out= 0)(stored 0%)
adding: java_classes/oldapi/(in = 0) (out= 0)(stored 0%)
adding: java_classes/oldapi/MaxMinMapper.class(in = 1938) (out= 790)(deflated 59%)
adding: java_classes/oldapi/MaxMinReducer.class(in = 1824) (out= 762)(deflated 58%)
adding: java_classes/oldapi/MaxMin.class(in = 1438) (out= 787)(deflated 45%)
adding: MaxMinMapper.java(in = 964) (out= 410)(deflated 57%)
```

Figura 27: Compilamos y ejecutamos (I).

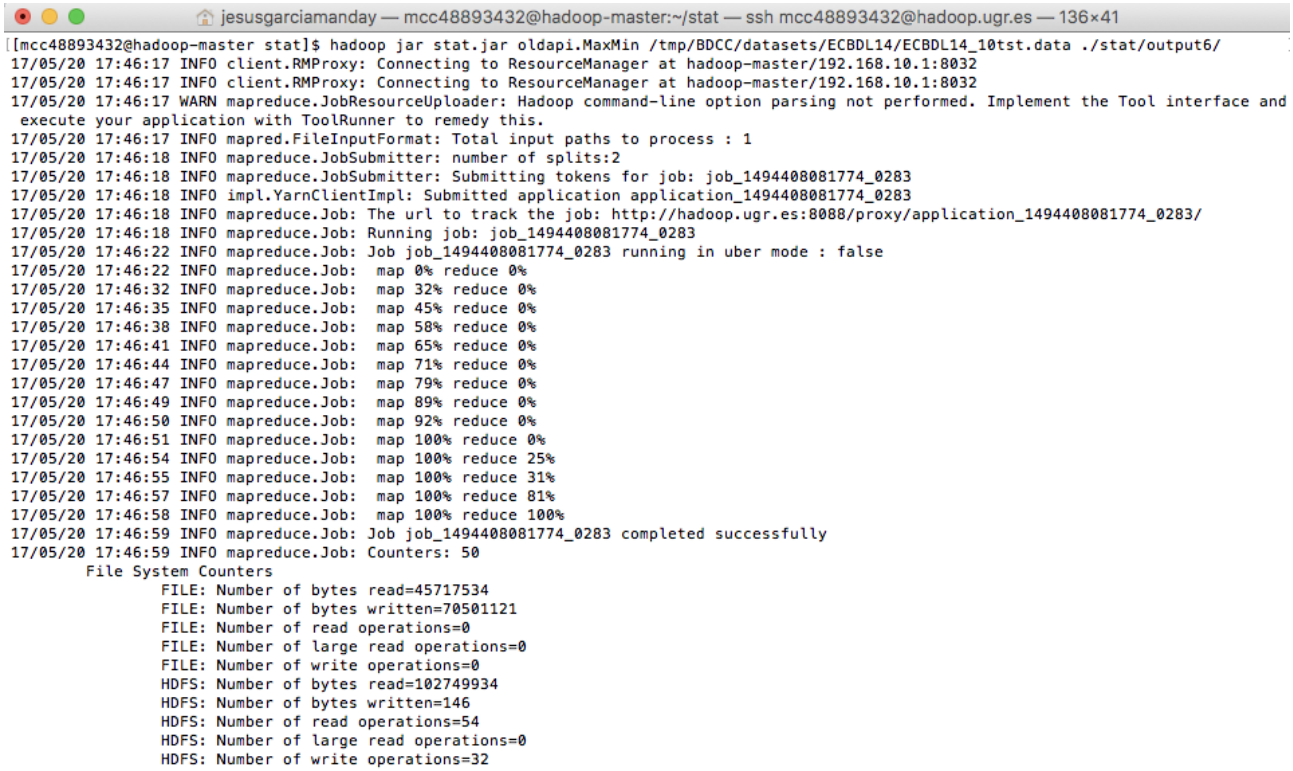


Figura 28: Compilamos y ejecutamos (II).

```
        Job Counters
                Launched map tasks=2
                Launched reduce tasks=16
                Data-local map tasks=1
                Rack-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=368760
                Total time spent by all reduces in occupied slots (ms)=3107090
                Total time spent by all map tasks (ms)=52680
                Total time spent by all reduce tasks (ms)=63410
                Total vcore-seconds taken by all map tasks=52680
                Total vcore-seconds taken by all reduce tasks=63410
                Total megabyte-seconds taken by all map tasks=368760000
                Total megabyte-seconds taken by all reduce tasks=3170500000
        Map-Reduce Framework
                Map input records=2897917
                Map output records=28979170
                Map output bytes=292689617
                Map output materialized bytes=22770570
                Input split bytes=234
                Combine input records=0
                Combine output records=0
                Reduce input groups=10
                Reduce shuffle bytes=22770570
                Reduce input records=28979170
                Reduce output records=20
                Spilled Records=86937510
                Shuffled Maps =32
                Failed Shuffles=0
                Merged Map outputs=32
                GC time elapsed (ms)=438
                CPU time spent (ms)=126880
                Physical memory (bytes) snapshot=10873622528
                Virtual memory (bytes) snapshot=984140087296
                Total committed heap usage (bytes)=21354250240
        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=102749700
        File Output Format Counters
                Bytes Written=146
```

Figura 29: Compilamos y ejecutamos (III).

Vemos el resultado que se ha obtenido.

```
jesusgarciamanday — mcc48893432@hadoop-master:~/stat — ssh mcc48893432@hadoop.ugr.es
[mcc48893432@hadoop-master stat]$ hdfs dfs -cat stat/output6/*
1       0.768
1       0.094
10      10.0
10      -13.0
2       0.154
2       0.0
3       10.0
3       -12.0
4       8.0
4       -11.0
5       9.0
5       -12.0
6       9.0
6       -11.0
7       9.0
7       -13.0
8       9.0
8       -12.0
9       7.0
9       -12.0
```

Figura 30: Comprobando resultado.

7. Realizar la media de la variable 5.

Para realizar esta tarea será necesario crear dos funciones nuevas para **Mapper** y **Reducer** con sus respectivas clases ya que difieren de las tareas anteriores.

jesusgarciamanday — mcc48893432@hadoop-master:~/stat — ssh mcc48893432@hadoop.ugr.es — 141×45

GNU nano 2.0.9 File: Average.java Modified

```
package oldapi;
import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
public class Average {
        public static void main(String[] args) throws IOException {
                if (args.length != 2) {
                        System.err.println("Usage: AverageTemperature <input path> <output path>");
                        System.exit(-1);
                }
                JobConf conf = new JobConf(Average.class);
                conf.setJobName("Average temperature");
                FileInputFormat.addInputPath(conf, new Path(args[0]));
                FileOutputFormat.setOutputPath(conf, new Path(args[1]));
                conf.setMapperClass(AverageMapper.class);
                conf.setReducerClass(AverageReducer.class);
                conf.setOutputKeyClass(Text.class);
                conf.setOutputValueClass(DoubleWritable.class);
                JobClient.runJob(conf);
        }
}
```

Figura 31: Clase Average.

jesusgarciamanday — mcc48893432@hadoop-master:~/stat — ssh mcc48893432@hadoop.ugr.es — 141×45

GNU nano 2.0.9 File: AverageMapper.java Modified

```
package oldapi;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
public class AverageMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, DoubleWritable> {
        private static final int MISSING = 0;
        public static int col=5;

        public void map(LongWritable key, Text value, OutputCollector<Text, DoubleWritable> output, Reporter reporter) throws IOException {
                String line = value.toString();
                String[] parts = line.split(",");
                output.collect(new Text("1"), new DoubleWritable(Double.parseDouble(parts[col])));
        }
}
```

Figura 32: Clase AverageMapper.

jesusgarciamanday — mcc48893432@hadoop-master:~/stat — ssh mcc48893432@hadoop.ugr.es — 141×45

GNU nano 2.0.9 File: AverageReducer.java Modified

```
package oldapi;
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
public class AverageReducer extends MapReduceBase implements Reducer<Text, DoubleWritable, Text, DoubleWritable> {


        public void reduce(Text key, Iterator<DoubleWritable> values, OutputCollector<Text, DoubleWritable> output, Reporter reporter) throw$
                Double maxValue = Double.MIN_VALUE;
                Double minValue = Double.MAX_VALUE;
                Double mValue = 0.0;
                Integer tam = 0;

                while (values.hasNext()) {
                        mValue += values.next().get();
                        tam ++;
                }
                output.collect(key, new DoubleWritable(mValue/tam));
        }
}
```

Figura 33: Clase AverageReducer.

Con las funciones creadas en su correspondiente clase pasamos a compilar y a ejecutar.

jesusgarciamanday — mcc48893432@hadoop-master:~/stat — ssh mcc48893432@hadoop.ugr.es — 1

```
[mcc48893432@hadoop-master stat]$ javac -cp /usr/lib/hadoop/*:/usr/lib/hadoop-mapreduce/* -d java_classes Average*
[mcc48893432@hadoop-master stat]$ /usr/java/jdk1.8.0_25/bin/jar -cvf stat.jar -C java_classes / .
added manifest
adding: oldapi/(in = 0) (out= 0)(stored 0%)
adding: oldapi/AverageMapper.class(in = 1809) (out= 708)(deflated 60%)
adding: oldapi/Average.class(in = 1444) (out= 785)(deflated 45%)
adding: oldapi/AverageReducer.class(in = 1920) (out= 804)(deflated 58%)
adding: java_classes/(in = 0) (out= 0)(stored 0%)
adding: java_classes/oldapi/(in = 0) (out= 0)(stored 0%)
adding: java_classes/oldapi/AverageMapper.class(in = 1809) (out= 708)(deflated 60%)
adding: java_classes/oldapi/Average.class(in = 1444) (out= 785)(deflated 45%)
adding: java_classes/oldapi/AverageReducer.class(in = 1920) (out= 804)(deflated 58%)
adding: Average.java(in = 1005) (out= 402)(deflated 60%)
adding: AverageReducer.java(in = 893) (out= 367)(deflated 58%)
adding: AverageMapper.java(in = 867) (out= 366)(deflated 57%)
```

Figura 34: Compilamos y ejecutamos (I).

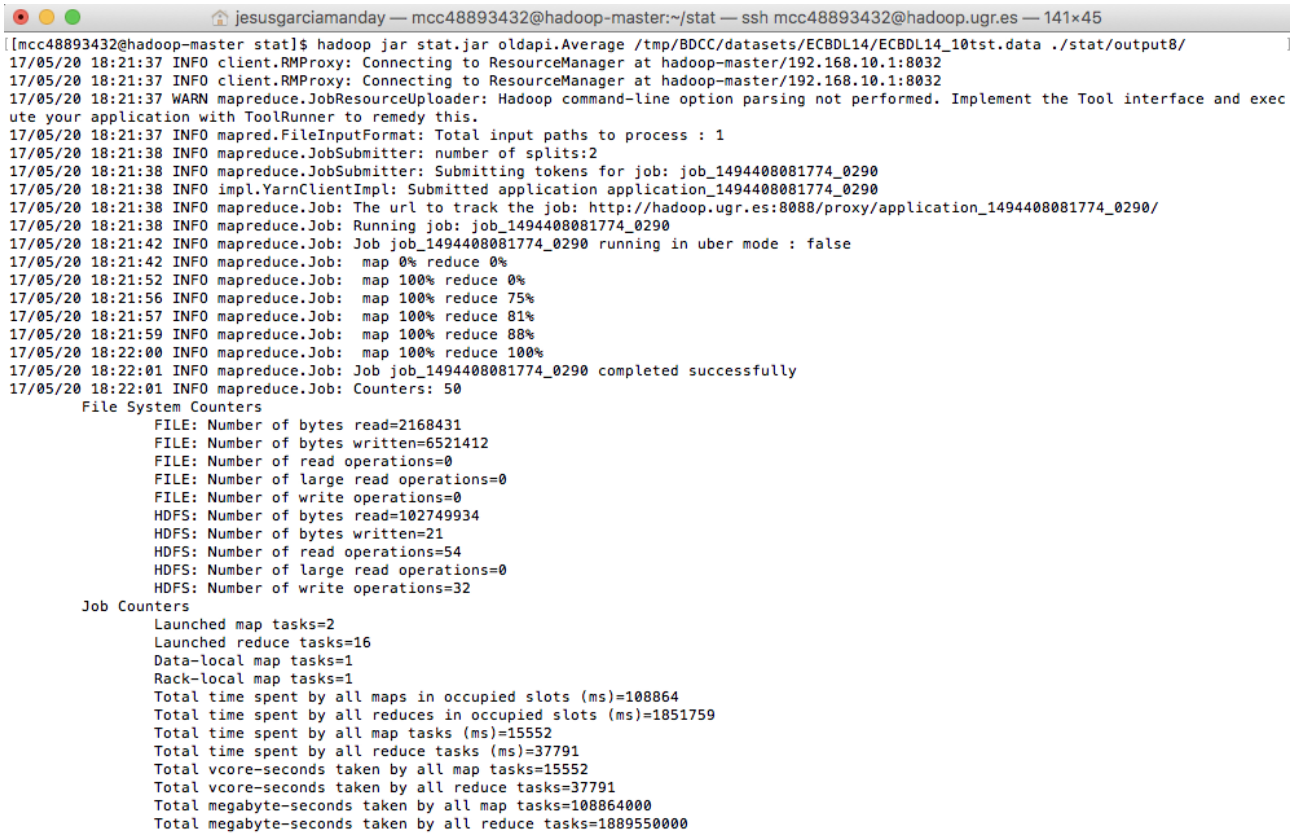


Figura 35: Compilamos y ejecutamos ()I).

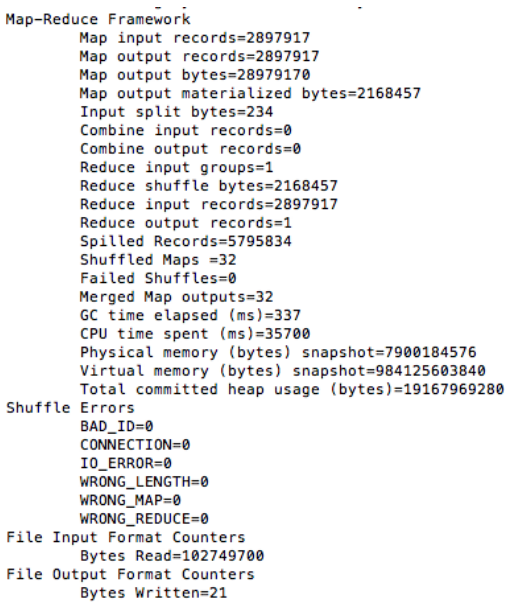


Figura 36: Compilamos y ejecutamos (III).

Y por último comprobamos el resultado.

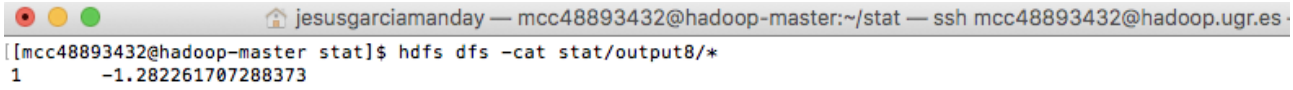


Figura 37: Comprobando resultado.

8. **Obtener la media de todas las variables (salvo la clase).**

9. **Comprobar si el conjunto de datos ECBDL es balanceado o no balanceado, es decir, que el ratio entre clases sea menor o mayor que 1.5 respectivamente.**

10. **Cálculo del coeficiente de correlación entre todas las parejas de variables.**