Parallel Distributed Systems

Albert Xavier López & Carlos Mougan &
Alexis Oger

Hadoop

Submitted To:
Toni Espinosa
Parallel Distributed
Systems
Computer Science
Department

Submitted By :
Albert Xavier López
Carlos Mougan
Alexis Oger
MSc in Modelling
CRM

1 Description

For this assignment the goal is to learn how to handle the Hadoop ecosystem. The first exercise it's going to be using the Python API and the following two exercises are made with JAVA.

2 Exercises

1) Try a version in python using hadoop-streaming library that counts the number of times each year is found in a record.

After installing Hadoop in our Nebula Caos we followed with the first exercise using Hadoop streaming with Python. First step is to open a new terminal and give permissions to our python scripts "mapper.py" and "reducer.py" and create a input data folder in HDFS and import a data file. For this exercise, the input data is the data set wheather-data.txt from the National Climatic Data Center which contains a list of meteorological measures such as: weather stations, observation dates, latitudes, longitudes, winds, temperature and so on. Second, we copy an input file from our local folder to HFDS file system by using the following command:

```
hdfs dfs -put /home/hadoop/Descargas/weather-data.txt/hduser/input/weather.txt
```

This weather-data.txt contains data in the following way so we will have to modify the mapper.py to be able to extract the information asked in the exercise:

```
0029029070999991901010106004+64333+023450FM-12+000599999V0202701N01591999..
0029029070999991901010113004+64333+023450FM-12+000599999V0202901N00821999..
0029029070999991901010120004+64333+023450FM-12+000599999V0209991C00001999..
0029029070999991901010206004+64333+023450FM-12+000599999V0201801N00821999..
0029029070999991901010213004+64333+023450FM-12+000599999V0201801N00981999..
0029029070999991901010220004+64333+023450FM-12+000599999V0201801N00981999..
0029029070999991901010306004+64333+023450FM-12+000599999V0202001N00981999..
0029029070999991901010313004+64333+023450FM-12+000599999V0202301N01181999..
....
```

For this exercise we were asked to detect the years and count how many times they appeared, so we made a simple algorithm in the "mapper.py" to detect in a string the values 15th to 19th where we know there are the years and save them using the following script:

```
#!/usr/bin/env python
import sys

for line in sys.stdin:

        line=line.strip()
        words = line.split()

        for word in words:
                a=word[15:19]
                print ('%s\t%s' % (a, "1"))
```

And we are able to run using Hadoop runtime with Hadoop-streaming tool to use our simplified Python implementation:

```
$HADOOP_HOME/bin/hadoop jar
$HADOOP_HOME/share/hadoop/tools/lib/hadoopmapreduce/hadoop-streaming-2.9.2.jar
-mapper /home/hadoop/Descargas/hadoop-data/mapper.py
-reducer /home/hadoop/Descargas/hadoop-data/reducer.py
-input /hduser/input -output /hduser/output
```

Finally, if we don't get any errors, we could check the output data using command:

```
bin/hdfs dfs -cat /hduser/output/*
```

And the final output with the number of times each of years are repeated on the weather file are:

```
1902 2
1903 2
1901 6564
```

2) Now try to modify the word count Java implementation to implement the maximum temperature of each year using the map and reduce implementations given.

The idea here is the same as before but in this case we have modified the script WordCount.java to perform the query task. The important task here is to locate the position of the temperature in the characters (88,93). Once located we just we print the temperature and the year. And finally we reduce to compute the maximum values for each year in this case for 1901, 1902 and 1903:

```
import java.io.IOException;
```

```
2  import java.util.StringTokenizer;
3  import org.apache.hadoop.conf.Configuration;
4  import org.apache.hadoop.fs.Path;
5  import org.apache.hadoop.io.IntWritable;
6  import org.apache.hadoop.io.Text;
7  import org.apache.hadoop.mapreduce.Job;
8  import org.apache.hadoop.mapreduce.Mapper;
9  import org.apache.hadoop.mapreduce.Reducer;
10 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
11 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
12
13 public class WordCount {
14
15         public static class TokenizerMapper
16         extends Mapper<Object, Text, Text, IntWritable>{
17             private final static IntWritable one = new IntWritable(1);
18             private Text word = new Text();
19
20             public void map(Object key, Text value, Context context) throws IOException,
21             InterruptedException {
22               StringTokenizer itr = new StringTokenizer(value.toString());
23                while (itr.hasMoreTokens()) {
24                  word.set(itr.nextToken());
25                  String line = word.toString();
26
27                  String year = line.substring(15,19);
28                  int temp;
29                  if(line.charAt(87) == '+')
30                  {
31                          temp = Integer.parseInt(line.substring(88,93));
32                  }
33                  else
34                  {
35                          temp = Integer.parseInt(line.substring(87,93));
36                  }
37                  temp=temp/10;
38                  //sign = str.substring(87,88)
39                  //temp = str
40                  Text years=new Text(year);
41                  IntWritable temps=new IntWritable(temp);
42                  context.write(years, temps);
43                  }
44             }
```

```
45          }
46
47          public static class IntSumReducer
48          extends Reducer<Text,IntWritable,Text,IntWritable> {
49                  private IntWritable result = new IntWritable();
50
51                  public void reduce(Text key, Iterable<IntWritable> values, Context context)
52                  throws IOException, InterruptedException {
53                      int maximum = 0;
54                      for (IntWritable val : values) {
55                          maximum = Math.max(maximum,val.get());
56                      }
57                      result.set(maximum);
58                      context.write(key, result);
59                  }
60          }
61
62
63          public static void main(String[] args) throws Exception {
64              Configuration conf = new Configuration();
65              Job job = Job.getInstance(conf, "word count");
66              job.setJarByClass(WordCount.class);
67              job.setMapperClass(TokenizerMapper.class);
68              job.setCombinerClass(IntSumReducer.class);
69              job.setReducerClass(IntSumReducer.class);
70              job.setOutputKeyClass(Text.class);
71              job.setOutputValueClass(IntWritable.class);
72
73              FileInputFormat.addInputPath(job, new Path(args[0]));
74              FileOutputFormat.setOutputPath(job, new Path(args[1]));
75              System.exit(job.waitForCompletion(true) ? 0 : 1);
76          }
77  }
```

And once we have modified the WordCount.java we just compile and execute the following steps in the command line:

```
1  $HADOOP_HOME/bin/hadoop com.sun.tools.javac.Main WordCount.java
2  jar cf wc.jar WordCount*.class
3  $HADOOP_HOME/bin/hdfs dfs -rm /hduser2/output/*
4  $HADOOP_HOME/bin/hdfs dfs -rmdir /hduser2/output
5  $HADOOP_HOME/bin/hadoop jar wc.jar WordCount /hduser2/input /hduser2/output
6  $HADOOP_HOME/bin/hdfs dfs -cat /hduser2/output/part-r-00000
```

Here we have to mention that running this for the first time we got that for 1901 there was a wrong register with a temperature register 9999. Given this is temperature is clearly wrong, we just deleted this row and we run again getting the following results:

```
1901 317
1902 40
1903 40
```

3) Modify the design of the Java program to calculate the average instead of the maximum. You can start by generating the number of measures for the year and the count of all measures.

Using the previous Java script, we just need to change the max function by an average function. To show this change we will just copy here the part of the WordCount.java we changed:

```
...
public void reduce(Text key, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException {
                int sum = 0;
          int count = 0;
                for (IntWritable val : values) {
                sum += val.get();
                count += 1;
                }
                result.set(sum/count);
                context.write(key, result);
        }
...
```

And once modified the java script we can run again using this command line procedures:

```
$HADOOP_HOME/bin/hadoop com.sun.tools.javac.Main Average.java
jar cf wc.jar Average*.class
$HADOOP_HOME/bin/hdfs dfs -rm /hduser2/output/*
$HADOOP_HOME/bin/hdfs dfs -rmdir /hduser2/output
$HADOOP_HOME/bin/hadoop jar wc.jar Average /hduser2/input /hduser2/output
$HADOOP_HOME/bin/hdfs dfs -cat /hduser2/output/part-r-00000
```

and we get that the average temperature for those years were:

```
1901 46
```

```
2 1902 25
3 1903 30
```

3 Conclusions

In this assignment we have learned to manage the Hadoop environment. The installation in Nebula Caos or in our Virtual Machines was very time-consuming. However, once we finished the queries using Python or Java were quite straight forward if you have knowledge in those programming languages.