PROJECT #1

MapReduce - Hadoop

Abstract

A project implementing K-Means clustering algorithm in Map-Reduce using synthetic data as a sample.

Chiotis Nikolaos

8150149

Contents

Introduction

This project describes the implementation of the K-Means clustering algorithm using the Hadoop MapReduce framework. In order to test the effectiveness, the algorithm will be applied on synthetic data. The project contains two code files. PointsGenerator.py, using python, generates a text file that contains data points in the form of (x, y), where x and y are real numbers. These points are parts of three clusters. The second file, KMeans.java, implements the K-Means clustering algorithm using Apache Hadoop Java API. These files can be found here: https://github.com/chiotisn/K-Means-X-MapReduce. These code files are developed and ran in Linux (Ubuntu 16.04) OS.

Hadoop Installation

Apache Hadoop has Java as a dependency, for this reason JDK has to be installed before installing Hadoop. I had already installed the JDK so I moved on to Hadoop installation. On Hadoop's website[1] there are the releases, I chose to install 3.2 version of Hadoop, so I downloaded the relevant binary tarball from a suggested mirror:

```
wget http://apache.forthnet.gr/hadoop/common/hadoop-3.2.0/hadoop-3.2.0.tar.gz
```

Hadoop's website recommends to check if the downloaded file is altered, so after downloading `hadoop-2.7.3.tar.gz.mds` file I did a check using SHA-256:

```
root@ubuntu:~# shasum -a 256 hadoop-3.2.0.tar.gz
226b6cbdf769467250054b3abdf26df9f05fde44bbb82fe5d12d6993ea848f64  hadoop-3.2.0.tar.gz
```

This output should match the SHA256 value in the `hadoop-2.7.3.tar.gz.mds` file:

```
/build/source/target/artifacts/hadoop-3.2.0.tar.gz:
SHA256 = 226B6CBD F7694672 50054B3A BDF26DF9 F05FDE44 BBB82FE5 D12D6993 EA848F64
```

Then, I uncompressed the downloaded file and moved the extracted files into `/usr/local`. After setting the path to Java at the `hadoop-env.sh` file, Hadoop was able to run:

```
root@ubuntu:~# /usr/local/hadoop/bin/hadoop
Usage: hadoop [OPTIONS] SUBCOMMAND [SUBCOMMAND OPTIONS]
 or    hadoop [OPTIONS] CLASSNAME [CLASSNAME OPTIONS]
  where CLASSNAME is a user-provided Java class
```

[1] https://hadoop.apache.org/releases.html

Points Generation

PointsGenerator.py generates a text file, containing a number of data points in the form of "x,y", where x and y are real numbers. The generation of these points are biased toward the creation of three clusters with random centers. In this case, random centers are: (10,15), (35,2), (50,30). The rest of the points are generated around these centers, using some random distance following a Gamma(1,5) skewed distribution[2].

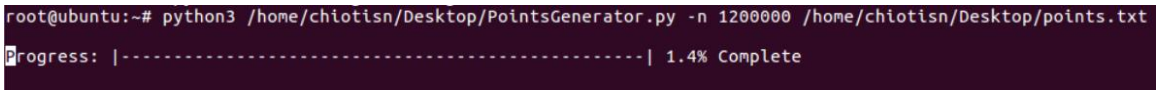
Arguments

```
$python3   PointsGenerator.py    [-n NUM]     out

positional arguments:
  out                   output file

optional arguments:
  -n NUM, --number      number of points to generate  (default: 1.000.000)
```

PointsGenerator.py has two arguments, the first one is the output file, in other words, the text file that the generated points will be stored. In the case this file does not exist it will be created by the program, otherwise it must be empty. The second one is the number of the points that will be generated. This argument is optional and its default value is 1M.

Execution

When executed the program prints a simple progress bar. This program must run with python3, because python2 doesn't support a command which is used for the printing of the progress bar.



[2] https://en.wikipedia.org/wiki/Gamma_distribution

Algorithm

```
1.  import argparse
2.  import numpy as np
3.  import random
4.
5.  parser = argparse.ArgumentParser()
6.  parser.add_argument("file", help = "the file in which the generated points will stored")
7.  parser.add_argument("-n", "--
    number", default = "1000000", help = "number of points to generate", type = int)
8.  args = parser.parse_args()
9.
10. print()
11. print('\r%s |%s| %s%% %s' % ('Progress:', '-' * 50, "0.0", 'Complete'), end = '\r')
12.
13. f = open(args.file,"w")
14.
15. #number of generated points
16. num = args.number
17.
18. #the three centers
19. c1 = (10,15)
20. c2 = (35,2)
21. c3 = (50,30)
22. centers = [c1,c2,c3]
23.
24. counter = 0 #number of points generated
25. points = set()
26.
27. while counter < num:
28.     center = centers[np.random.randint(0,3)]
29.     dist = np.random.gamma(1,5) #random distance following a gamma(1,5) distribution towar
    ds 0
30.     x = random.uniform(center[0] - dist, center[0] + dist) #abscissa of the random point
31.     sign = random.choice([-1,1])
32.     y = sign * np.sqrt( dist**2 - (x - center[0])**2 ) + center[1] #circle: (x-a)^2 + (y-
    b)^2 = r^2
33.     point = (x,y)
34.     l = len(points)
35.     points.add(point)
36.     if l != len(points):
37.         counter += 1
38.         if counter == 1 :
39.             f.write(str(point[0]) + "," + str(point[1]))
40.         else:
41.             f.write("\n" +str(point[0]) + "," + str(point[1]))
42.
43.     #Progress Bar
44.     percent = ("{0:.1f}").format(100 * (counter / float(num)))
45.     filledLength = int(50 * counter // num)
46.     bar = '' * filledLength + '-' * (50 - filledLength)
47.     print('\r%s |%s| %s%% %s' % ('Progress:', bar, percent, 'Complete'), end = '\r')
48. print()
```

Lines:

{5-8} Using argparse to add and parse arguments.

{10-11} Printing the initial progress bar.

{18-22} Setting three random centers.

{27-41} Generating points.

{28} Choosing one of the three centers to generate random point around it.

{29} Getting the random distance (d).

{30} Getting the random x of the generated point. The generated point (x,y) is on a circle around the center (a,b) with r = d . So, x can get values within the set [a-d,a+d].

{31-32} After getting x, we can find y out of the circle equation (x-a)^2 + (y-b)^2=r. We choose a random sign as this equation has two solutions.

{33-41} We add the point in the set of points, then we check if the length of the set is changed. If the length is the same, it means that there is another point with the same coordinates, if the length is not the same, then we write the point into the file.

{43-38} Progress bar.

K-Means in MapReduce

KMeans.java implements the K-Means clustering algorithm using Hadoop MapReduce framework. It uses Apache Hadoop Java API to implement the Mapper and the Reducer methods. It also use ToolRunner in order to make the Map – Reduce procedure iterative. The output file of the algorithm contains the 3 centers of the clusters and the number of nodes in each cluster. Before the execution of the program we have to move our file, which contains the data points, to HDFS:

```
root@ubuntu:~# /usr/local/hadoop/bin/hadoop fs -copyFromLocal /home/chiotisn/Desktop/points.txt /home/chiotisn/Desktop/input
```

Arguments

```
$bin/hadoop    jar    KMeans.jar    KMeans    input_dir    output_dir
```

In order to run the algorithm we have to give as an argument the input file and the output directory. Output directory must not exist.

Execution

First, we have to compile the java file:

```
root@ubuntu:~# /usr/local/hadoop/bin/hadoop com.sun.tools.javac.Main KMeans.java
```

Then, we have to create the .jar file:

```
root@ubuntu:~# jar cf KMeans.jar KMeans*.class
```

Finally, we can execute the jar file:

```
root@ubuntu:~# /usr/local/hadoop/bin/hadoop jar KMeans.jar KMeans /home/chiotisn/Desktop/input/points.txt /home/chiotisn/Desktop/KMeans-output
```

The output file:

```
35.93673029294865,7.40379913970796      |422348
9.417788101583147,14.9749819498817      |313936
51.39159163158015,29.706635792379732    |263716
```

Algorithm

This algorithm is based on Apache's tutorial in the documentation of Hadoop[3]. It consists of 4 classes. KMeans class that contains the main method, PointsMapper that extends Hadoop's Mapper class, PointsReducer that extends Hadoop's Reducer class and Point class, which contains methods used for points' processing. Algorithm starts with 3 random points as the centers of the three clusters, then for each point finds the nearest center and assigns the point to the corresponding cluster. After this stage, it recalculates the centers of the three clusters. Algorithm's iterations stops when the new centers have small distance (<0.3) from the old centers or it has reached the maximum number of iterations, which is 6.

Point Class

Point objects are used to represent a data point in the form of (x,y). The constructor gets a String that contains a line of the text file and stores the x and y values. The getDist method ii returns the Euclidean distance between two Points and the toString method returns a String representation of the Point, in order to be written in the "context" of MapReduce process or in the output file. Point class created to make development of the other classes simpler.

PointsMapper Class

PointsMapper class implements the map procedure of the algorithm. Each time, it gets a (x,y) point from the input file and maps it to the nearest centroid.

PointsReducer Class

PointsReducer class implements the reduce procedure of the algorithm. For each center, it calculates the new mean point of the cluster (lines 94-110), then it checks if the new center of the cluster has changed (lines 111-116), if the center have changed it removes the old center from the global variable that contains the centers and then writes it as an output to the output file. In both cases, the centroidsStatus global variable, which shows if all the three centers have changed or not, is updated.

KMeans Class

KMeans class contains the main method, which controls the iterations of the algorithm. In main method, the random centers are initialized (lines 145-147), then, while the number of iterations are less than 6 or at least one center has changed, the MapReduce process repeats using ToolRunner. The MapReduce process runs using the run method. In the run method, the necessary variables are set for the MapReduce process and the output file of the previous iteration is deleted.

[3] https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

Code

```
1.  import java.io.BufferedReader;
2.  import java.io.FileReader;
3.  import java.io.BufferedWriter;
4.  import java.io.FileWriter;
5.  import java.io.File;
6.  import java.io.IOException;
7.  import java.io.InputStreamReader;
8.  import java.util.ArrayList;
9.
10. import org.apache.hadoop.conf.*;
11. import org.apache.hadoop.fs.Path;
12. import org.apache.hadoop.fs.FileSystem;
13. import org.apache.hadoop.io.*;
14. import org.apache.hadoop.mapreduce.Job;
15. import org.apache.hadoop.mapreduce.Mapper;
16. import org.apache.hadoop.mapreduce.Reducer;
17. import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
18. import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
19. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
20. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
21. import org.apache.hadoop.util.*;
22.
23. public class KMeans  extends Configured implements Tool {
24.
25.     public static ArrayList<Point> centroids = new ArrayList<Point>();
26.     public static Configuration conf;
27.     public static Boolean stop = false;
28.     public static ArrayList<String> centroidsStatus = new ArrayList<String>();
29.
30.     public static class Point {
31.
32.        private double cords[] = new double[2];
33.
34.        public Point(String line) {
35.          String x,y;
36.          x = line.split(",")[0];
37.          y = line.split(",")[1];
38.          cords[0] = Double.parseDouble(x);
39.          cords[1] = Double.parseDouble(y);
40.            }
41.
42.        public double getx() {
43.          return cords[0];
44.        }
45.
46.        public double gety() {
47.          return cords[1];
48.        }
49.
50.        public static double getDist(Point a, Point b) {
51.          return Math.sqrt( Math.pow((a.getx() - b.getx()), 2) + Math.pow((a.getx() - b.getx()), 2) );
52.        }
53.
54.        public String toString() {
55.          String point = String.valueOf(cords[0]) + "," + String.valueOf(cords[1]);
56.          return point;
57.        }
```

```
58.       }
59.
60.     public static class PointsMapper extends Mapper<LongWritable, Text, Text, Text> {
61.
62.         @Override
63.         public void setup(Context context) {
64.
65.         }
66.
67.         @Override
68.         public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedE
    xception {
69.
70.             int index = -1;
71.             double minDist = Double.MAX_VALUE;
72.             Point p = new Point(value.toString());
73.
74.             for (int i = 0; i < 3; i++) {
75.                double dist = Point.getDist(p, centroids.get(i));
76.                if (dist < minDist) {
77.                    minDist = dist;
78.                    index = i;
79.                }
80.             }
81.             context.write(new Text(centroids.get(index).toString()), new Text(p.toString()));
82.         }
83.
84.         @Override
85.         public void cleanup(Context context) throws IOException,InterruptedException {
86.
87.         }
88.     }
89.
90.     public static class PointsReducer extends Reducer<Text, Text, Text, Text> {
91.
92.           @Override
93.           public void reduce(Text key,Iterable<Text> values,Context context) throws IOException,Interru
    ptedException {
94.                double sumx = 0;
95.                double sumy = 0;
96.                    int counter = 0;
97.                String line, x, y;
98.
99.                while (values.iterator().hasNext()) {
100.                                line = values.iterator().next().toString();
101.                            String[] str1=line.split(":");
102.                            x = line.split(",")[0];
103.                            y = line.split(",")[1];
104.                            sumx += Double.parseDouble(x);
105.                            sumy += Double.parseDouble(y);
106.                            counter ++;
107.                    }
108.                    double newx = sumx/counter;
109.                    double newy = sumy/counter;
110.                    Point new_centroid = new Point(String.valueOf(newx) + "," + String.valueOf(newy));
111.                     Point old_centroid = new Point(key.toString());
112.                    String changed = "not changed";
113.                    if (Point.getDist(new_centroid,old_centroid) > 0.3) {
114.                      changed = "changed";
115.                      centroids.remove(old_centroid);
116.                      centroids.add(new_centroid);
```

```
                    }
                    centroidsStatus.add(changed);
                    context.write(new Text(new_centroid.toString()),new Text("|" + String.valueOf(count
    er)));
                }
            }

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

              Job job = Job.getInstance(conf, "KMeans");
              FileSystem fs=FileSystem.get(conf);
              job.setJarByClass(KMeans.class);
              job.setMapperClass(PointsMapper.class);
              job.setReducerClass(PointsReducer.class);
              job.setOutputKeyClass(Text.class);
              job.setOutputValueClass(Text.class);
              job.setNumReduceTasks(1);
              FileInputFormat.addInputPath(job, new Path(args[0]));
              Path output = new Path (args[1]);
              fs.delete(output,true);
              FileOutputFormat.setOutputPath(job, output);

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

            public static void main(String[] args) throws Exception {

                //3 random centroids
                centroids.add(new Point("5.938284859084138,10.439430174938032"));
                centroids.add(new Point("70.818654512579638,29.559624209285044"));
                centroids.add(new Point("25.77863208190015,1.058336941667927"));

                conf = new Configuration();

                int i = 0;
                int exit = 1;
                do {
                  exit ^= ToolRunner.run(conf, new KMeans(), args);
                  i++;
                  if (!centroidsStatus.contains("changed")) {
                    stop = true;
                  }
                  centroidsStatus.clear();
                } while (exit == 1 && i < 6 && !stop); //maximum iterations: 6
                System.out.println("Number of iterations: " + String.valueOf(i));
            }

        }
```