BIG DATA MANAGEMENT SYSTEMS: PROJECT #1 – MAPREDUCE/HADOOP

April 7, 2019

Zoe Kotti and Chryssa Nampouri
Department of Management Science and Technology
Athens University of Economics and Business
Athens, Greece
{t8150062, t8150096}@aueb.gr

Supervisor: Prof. Damianos Chatziantoniou

Contents

1 Project Description 2
2 Data Generation 3
3 MapReduce Implementation of K-Means in Python 5
4 MapReduce Runner for K-Means 10
5 Execution & Results of MapReduce for K-Means 15

1 Project Description

The purpose of this project was to implement the *K-Means* clustering algorithm in *MapReduce* and apply in on synthetic data that we would create.

First, we installed Hadoop on Linux Ubuntu by following the "*How to install Hadoop on Ubuntu 18.04 Bionic Beaver Linux*", as introduced by Sandip Bhowmik. [1]

We then created a comma-separated values (**CSV**) file in Python that contained 1.2 million data points in the form (x, y), where x and y are real numbers. The generation of the data points was biased toward the creation of three clusters. In other words, we chose a-priori three centers (x1, y1), (x2, y2) and (x3, y3), and generated the rest of the data points around these, using some random distance following a skewed distribution (towards 0).

Next, we moved the particular file with the data points to **HDFS**, using the following commands in terminal:

```
$ hdfs dfs -mkdir /kmeans
$ hdfs dfs -put $HADOOP_HOME/localFilePath/data-points.csv /kmeans
```

Finally, we implemented the *K-Means* algorithm as described in the class and applied it to our data set. We used **Hadoop Streaming** for our project, and followed the example "*Writing An Hadoop MapReduce Program In Python*" by Michael G. Noll. [2]

[1] https://linuxconfig.org/how-to-install-hadoop-on-ubuntu-18-04-bionic-beaver-linux
[2] https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python

2 Data Generation

We implemented the data generation in Python. In order to generate the data points we used the *make_ blobs* function of the *scikit-learn* package. [3] In this way, our data points follow a normal distribution with standard deviation equal to **5.0**.
The initial centers that we used are the following:

1. **(-100000, -100000)**
2. **(1, 1)**
3. **(100000, 100000)**

The code that we developed in order to implement the particular process of data generation can be found below:

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

"""
generateDataset.py: Generates two csv files;
    1. data-points.csv: the data points that will be used for the
        clustering using the K-Means algorithm
    2. data-points-labels.csv: the data points accompanied by the
        label of the cluster they belong to
By default, our clusters are 3, and the data points are in total
1,200,000.

"""

__author__ = "Zoe Kotti, Chryssa Nampouri"

import numpy as np
import pandas as pd
from sklearn.datasets.samples_generator import make_blobs

# Hard-append initial centroids and number of data points
centroids = [[-100000, -100000], [1, 1], [100000, 100000]]
n_samples = 1200000

fileName_points = "data-points.csv"
fileName_points_labels = "data-points-labels.csv"

# Generate the data points by following normal distribution
```

[3] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html

```
X, labels = make_blobs(n_samples=n_samples, centers=centroids,
cluster_std=5.0, n_features=2)
# Round data point coordinates to first digit
X = np.round(X, 1)

points = pd.DataFrame(X)
points_labels = pd.DataFrame(X, labels)
# Write files
points.to_csv(fileName_points, sep=',', index=False, header=False)
points_labels.to_csv(fileName_points_labels, sep=',', header=False)
```

3 MapReduce Implementation of K-Means in Python

We implemented the *MapReduce* process in Python. We decided to also include the **Combine** process, as described in the class. For these processes, we created three Python files: *mapper.py*, *combiner.py*, *reducer.py*.

The code that we developed for the **Map** process follows below:

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

"""
mapper.py: Performs the Map process.

Input: A csv file with the current centroids of the clusters
Process: Reads the data points from HDFS (STDIN), calculates their
    Manhattan distances from the current centroids,
    and appends them to their closest cluster.
Output: For each data point, a key-value set of its cluster (key) and
    the point's coordinates (value) is returned (e.g. 1 [-5.1, 6.8]).
    The key-value sets are sorted by cluster (key).

"""

__author__ = "Zoe Kotti, Chryssa Nampouri"

import sys
import numpy as np

# Read csv file with current centroids
CENTROIDS_FILE = "old-centroids.csv"

with open(CENTROIDS_FILE, "r") as centroidsFile:
    centroidsFile = centroidsFile.readlines()
    centroids = []

    for centroid in centroidsFile:
        centroid = centroid.strip().split(",")
        centroid = [float(centroid[0]), float(centroid[1])]
        centroids.append(centroid)

```

```
# Read input with data points from HDFS (STDIN - standard input)
for line in sys.stdin:
    point = line.strip().split(",")
    point = [float(point[0]), float(point[1])]
    distances = [0] * len(centroids)

    for i in range(len(centroids)):
        # Calculate Manhattan distance
        xDistance = abs(point[0] - centroids[i][0])
        yDistance = abs(point[1] - centroids[i][1])
        distances[i] = xDistance + yDistance
    # Find closest centroid index of data point
    cluster = np.argmin(distances)
    # Write data point and its cluster to STDOUT
    print('%s\t%s' % (cluster, point))
```

The code that we developed for the **Combine** process follows below:

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

"""
combiner.py: Performs the Combine process.

Input: A key-value set of a data point's coordinates (value)
    and its cluster (key); this is the output of Map Process (mapper.py)
Process: Reads Input from HDFS (STDIN) and calculates for each cluster,
    partial sums of its data points in batches.
Output: For each batch of partial sums, a key-value set
    of its cluster (key), its partial sum and the number of data points
    related to the particular sum (value) is returned
    (e.g. 1 ([-5.1, 6.8], 4)).
    The key-value sets are sorted by cluster (key).

"""

__author__ = "Zoe Kotti, Chryssa Nampouri"

import sys
import ast

current_cluster = None
partial_sum = []
```

```
cluster = None

# Read Input from HDFS (STDIN - standard input)
for line in sys.stdin:
    cluster, point = line.strip().split('\t', 1)
    # Convert String representation of list into actual list
    # (e.g. '[-5.8, 3.6]' --> [-5.8, 3.6])
    point = ast.literal_eval(point)

    # Input is sorted by cluster (key) and IF-switch is based on this logic
    if current_cluster == cluster:
        partial_sum[0] += point[0]
        partial_sum[1] += point[1]
        num_points += 1
    else:
        if current_cluster:
            # Write cluster, partial sum and number
            # of data points to STDOUT
            print ('%s\t%s' % (current_cluster,
                (partial_sum, num_points)))
        # Initialize/Update variables
        partial_sum = point
        num_points = 1
        current_cluster = cluster

# Make sure last record is also written to STDOUT
if current_cluster == cluster:
    print ('%s\t%s' % (current_cluster, (partial_sum, num_points)))
```

The code that we developed for the **Reduce** process follows below:

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

"""
reducer.py: Performs the Reduce process.

Input: A key-value set of a cluster (key), its partial sum and the number
    of data points related to the particular sum (value);
    this is the output of Combine Process (combiner.py)
Process: Reads Input from HDFS (STDIN) and calculates for each cluster,
    the total sum of all the partial sums of its data points.
    Then for each cluster, the mean value of its data points is computed;
```

```
    that's the cluster's new centroid.
Output: The new centroids of the clusters.

"""

__author__ = "Zoe Kotti, Chryssa Nampouri"

import sys
import ast

current_cluster = None
current_partial_sum = []
cluster = None

# Read Input from HDFS (STDIN - standard input)
for line in sys.stdin:
    cluster, partial = line.strip().split('\t', 1)
    partial_sum = ",".join(partial.split(",", 2)[:2]).replace("(", "")
    num_points = partial.split(",", 2)[2].replace(")", "")
    num_points = int(num_points)
    # Convert String representation of list into actual list
    # (e.g. '[-5.8, 3.6]' --> [-5.8, 3.6])
    partial_sum = ast.literal_eval(partial_sum)

    # Input is sorted by cluster (key) and IF-switch is based on this logic
    if current_cluster == cluster:
        current_partial_sum[0] += partial_sum[0]
        current_partial_sum[1] += partial_sum[1]
        current_num_points += num_points
    else:
        if current_cluster:
            # Calculate new centroid coordinates, rounded to first digit
            xCentroid = round(current_partial_sum[0]/current_num_points, 1)
            yCentroid = round(current_partial_sum[1]/current_num_points, 1)
            new_centroid = [xCentroid, yCentroid]
            # Write new centroid to STDOUT
            print('%s\t' % (new_centroid))
        # Initialize/Update variables
        current_partial_sum = partial_sum
        current_num_points = num_points
        current_cluster = cluster

```

```
54 # Make sure last record is also written to STDOUT
55 if current_cluster == cluster:
56     xCentroid = round(current_partial_sum[0]/current_num_points, 1)
57     yCentroid = round(current_partial_sum[1]/current_num_points, 1)
58     new_centroid = [xCentroid, yCentroid]
59     print('%s\t' % (new_centroid))
```

4 MapReduce Runner for K-Means

We implemented the running process of *MapReduce*, as described in 3, by developing another Python file named *kMeansRunner.py*. kMeansRunner.py is executed for as long as the centroids of the clusters keep changing; if the centroids do not change in two sequential iterations, then the process completes, and the final centroids are the last that occurred from this process.

Before the user runs kMeansRunner.py, he must first create a directory named *KMeansProject* under the directory where all Python files are stored. Otherwise, he needs to modify all paths that include the particular directory according to his preferences. In this directory, the output of the *MapReduce* process (i.e. the centroids produced in an iteration of K-Means) is copied locally temporarily from **HDFS**. The particular file is used for the comparison of the new and the last centroids.

While executing the *MapReduce* process, we also keep a file with all centroids produced during all iterations of K-Means algorithm. At the end, this is our output; the last line of this file includes the **final** centroids.

All input and output files of the MapReduce process can also been seen through the localhost of the browser in a more user-friendly way.

The code of **KMeansRunner** follows below:

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

"""
kMeansRunner.py: Implements the K-Means algorithm using
the Map-Combine-Reduce process.

"""

__author__ = "Zoe Kotti, Chryssa Nampouri"

import ast
import random
import subprocess # Requires Python 3

CENTROIDS_FILE = "old-centroids.csv"
INPUT_FILE = "data-points.csv"
# Create a subdirectory "KMeansProject" inside the current directory
# and save the output according to its name in HDFS
```

```
19 OUTPUT_FILE = "KMeansOutput/part-00000"
20 CENTROIDS = "all-centroids.csv"
21
22 class KMeansRunner(object):
23     """
24     Implements methods needed to support the K-Means algorithm
25     and the Map-Combine-Reduce process.
26     """
27
28     @staticmethod
29     def RetrieveDataPoints(file):
30         """Retrieves the data points from a file.
31         :param file: A file with the data points
32         :return: A list with the data points
33         """
34         with open(file, "r") as data:
35             data = data.readlines()
36             dataList = []
37             for d in data:
38                 d = d.strip().split(",")
39                 d = [float(d[0]), float(d[1])]
40                 dataList.append(d)
41         return dataList
42
43     @staticmethod
44     def AddCentroids(centroids):
45         """Adds the current centroids to a file that contains all centroids
46         from all the iterations of the K-Means algorithm.
47         :param centroids: The new centroids resulted from an iteration
48             of the K-Means algorithm
49         """
50         with open(CENTROIDS, "a") as file:
51             for centroid in centroids:
52                 file.write("%s\n" % str([centroid]).strip('[]'))
53
54     def RetrieveCentroids(self, file):
55         """Retrieves the current centroids from a file.
56         :param self: An instance of the class KMeansRunner
57         :param file: The file that contains only the current centroids
58         :return: The current centroids in a list
59         """
60         with open(file, "r") as centroidsFile:
```

```
61              centroidsFile = centroidsFile.readlines()
62              centroids = []
63              for centroid in centroidsFile:
64                  centroid = ast.literal_eval(centroid)
65                  centroids.append(centroid)
66          return centroids
67
68      @staticmethod
69      def CheckCentroids(oldCentroids, newCentroids):
70          """Checks whether the cluster centroids of the new iteration of the
71          K-Means algorithm differ from those resulted from the last one.
72          :param oldCentroids: The last centroids
73          :param newCentroids: The new centroids
74          :return: A boolean value; True if the centroids have changed
75              and False if not
76          """
77          match = False
78          if sorted(oldCentroids) == sorted(newCentroids):
79              match = True
80          return match
81
82      @staticmethod
83      def WriteCentroids(centroids):
84          """Updates the file that contains the current centroids with
85          the new ones, only in case they differ from the current.
86          :param centroids: The new centroids
87          """
88          with open(CENTROIDS_FILE, "w+") as file:
89              for centroid in centroids:
90                  file.write("%s\n" % str([centroid]).strip('[]'))
91
92  if __name__ == "__main__":
93
94      instanceKMeans = KMeansRunner()
95      # Retrieve the initial data points
96      dataPointsList = instanceKMeans.RetrieveDataPoints(INPUT_FILE)
97      # Generate the initial centroids of the clusters randomly
98      initialCentroids = random.sample(dataPointsList, k=3)
99      instanceKMeans.WriteCentroids(initialCentroids)
100     instanceKMeans.AddCentroids(initialCentroids)
101
102     match = False
```

```
103     # Run until centroids do not change for two sequential iterations
104     while (match == False):
105         # Connect with HDFS and run Map-Combine-Reduce process
106         # through Hadoop Streaming
107         completed = subprocess.run(["/home/hadoop/hadoop-2.8.5/bin/hadoop",
108         "jar",
109         "/home/hadoop/hadoop-2.8.5/share/hadoop/tools/lib/hadoop-streaming-2.8.5.jar",
110         "-file", "mapper.py", "-mapper","mapper.py",
111         "-file","combiner.py" ,"-combiner" ,"combiner.py",
112         "-file", "reducer.py", "-reducer", "reducer.py",
113         "-file", "old-centroids.csv",
114         "-input", "/kmeans/data-points.csv",
115         "-output","kmeans_output/output"])
116         # Copy files from HDFS locally to KMeansProject subdirectory
117         # Attention: The user must first create this directory
118         # under the directory where all Python files are located
119         output = subprocess.run(["/home/hadoop/hadoop-2.8.5/bin/hadoop",
120         "fs", "-get", "/user/hadoop/kmeans_output/output/part-00000",
121         "KMeansProject/"])
122 
123         oldCentroids = instanceKMeans.RetrieveCentroids(CENTROIDS_FILE)
124         newCentroids = instanceKMeans.RetrieveCentroids(OUTPUT_FILE)
125         oldCentroids= [list(centroid) for centroid in oldCentroids]
126         instanceKMeans.AddCentroids(newCentroids)
127 
128         match = instanceKMeans.CheckCentroids(oldCentroids, newCentroids)
129         # Centroids have changed during two sequential iterations
130         if match == False:
131             # Update centroids file with the new ones
132             instanceKMeans.WriteCentroids(newCentroids)
133             # Remove the output from both HDFS and local directory
134             # and re-create it in next iteration
135             remove_previous_output_hdfs = subprocess.run(["hdfs",
136             "dfs", "-rm", "-r",
137             "/user/hadoop/kmeans_output/output"])
138             remove_previous_output_local = subprocess.run(["rm",
139             "-r", "KMeansOutput/part-00000"])
140         # Centroids have NOT changed during two sequential iterations
141         else:
142             remove_previous_output_local = subprocess.run(["rm",
143             "-r", "KMeansOutput/part-00000"])
144             # Print the final coordinates of the cluster centroids
```

```
145             print()
146             print("The final coordinates of the cluster centroids are:")
147             print("----------------------------------------")
148             for i in range(len(newCentroids)):
149                 print("Cluster " + str(i) + ": " + str(newCentroids[i]))
```

5 Execution & Results of MapReduce for K-Means

The execution of MapReduce process requires Python 3. The process is executed through *kMeansRunner.py* and the user simply runs the following command:

```
$ python3 kMeansRunner.py
```

Results:

```
hadoop@chryssa-Inspiron-3558:~/PycharmProjects/untitled/test$ python3 kMeansRunner.py
19/04/07 04:47:59 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [mapper.py, combiner.py, reducer.py, old-centroids.csv, /tmp/hadoop-unjar1931174221918597313/] [] /tmp/streamjob869958189607048
9866.jar tmpDir=null
19/04/07 04:48:02 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
19/04/07 04:48:02 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
19/04/07 04:48:07 INFO mapred.FileInputFormat: Total input files to process : 1
19/04/07 04:48:08 INFO mapreduce.JobSubmitter: number of splits:2
19/04/07 04:48:09 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1554592935425_0012
19/04/07 04:48:09 INFO impl.YarnClientImpl: Submitted application application_1554592935425_0012
19/04/07 04:48:09 INFO mapreduce.Job: The url to track the job: http://chryssa-Inspiron-3558:8088/proxy/application_1554592935425_0012/
19/04/07 04:48:09 INFO mapreduce.Job: Running job: job_1554592935425_0012
19/04/07 04:48:17 INFO mapreduce.Job: Job job_1554592935425_0012 running in uber mode : false
19/04/07 04:48:17 INFO mapreduce.Job:  map 0% reduce 0%
19/04/07 04:48:39 INFO mapreduce.Job:  map 1% reduce 0%
19/04/07 04:48:49 INFO mapreduce.Job:  map 52% reduce 0%
19/04/07 04:48:56 INFO mapreduce.Job:  map 67% reduce 0%
19/04/07 04:49:23 INFO mapreduce.Job:  map 83% reduce 0%
19/04/07 04:49:25 INFO mapreduce.Job:  map 100% reduce 0%
19/04/07 04:49:47 INFO mapreduce.Job:  map 100% reduce 100%
19/04/07 04:49:49 INFO mapreduce.Job: Job job_1554592935425_0012 completed successfully
19/04/07 04:49:50 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=334
                FILE: Number of bytes written=488833
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=17988583
                HDFS: Number of bytes written=58
                HDFS: Number of read operations=9
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=2
                Launched reduce tasks=1
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=126375
```

```
    File System Counters
            FILE: Number of bytes read=330
            FILE: Number of bytes written=488819
            FILE: Number of read operations=0
            FILE: Number of large read operations=0
            FILE: Number of write operations=0
            HDFS: Number of bytes read=17988583
            HDFS: Number of bytes written=64
            HDFS: Number of read operations=9
            HDFS: Number of large read operations=0
            HDFS: Number of write operations=2
    Job Counters
            Launched map tasks=2
            Launched reduce tasks=1
            Data-local map tasks=2
            Total time spent by all maps in occupied slots (ms)=170692
            Total time spent by all reduces in occupied slots (ms)=39335
            Total time spent by all map tasks (ms)=170692
            Total time spent by all reduce tasks (ms)=39335
            Total vcore-milliseconds taken by all map tasks=170692
            Total vcore-milliseconds taken by all reduce tasks=39335
            Total megabyte-milliseconds taken by all map tasks=174788608
            Total megabyte-milliseconds taken by all reduce tasks=40279040
    Map-Reduce Framework
            Map input records=1200000
            Map output records=1200000
            Map output bytes=23984295
            Map output materialized bytes=336
            Input split bytes=192
            Combine input records=1200000
            Combine output records=6
            Reduce input groups=3
            Reduce shuffle bytes=336
            Reduce input records=6
            Reduce output records=3
            Spilled Records=12
            Shuffled Maps =2
```

```
19/04/07 04:51:08 INFO streaming.StreamJob: Output directory: kmeans_output/output

The final coordinates of the cluster centroids are:
----------------------------------------
Cluster 0: [-100000.0, -100000.0]
Cluster 1: [1.0, 1.0]
Cluster 2: [100000.0, 100000.0]
hadoop@chryssa-Inspiron-3558:~/PycharmProjects/untitled/test$
```

The output of the MapReduce process is also stored on Hadoop under the name *part-00000*: