

Project #1: MapReduce/Hadoop

Implementation Report

Big Data Management Systems (Associate Prof. Damianos Chatziantoniou)
Theodoros Plessas, 8160192 (t8160192@aueb.gr)

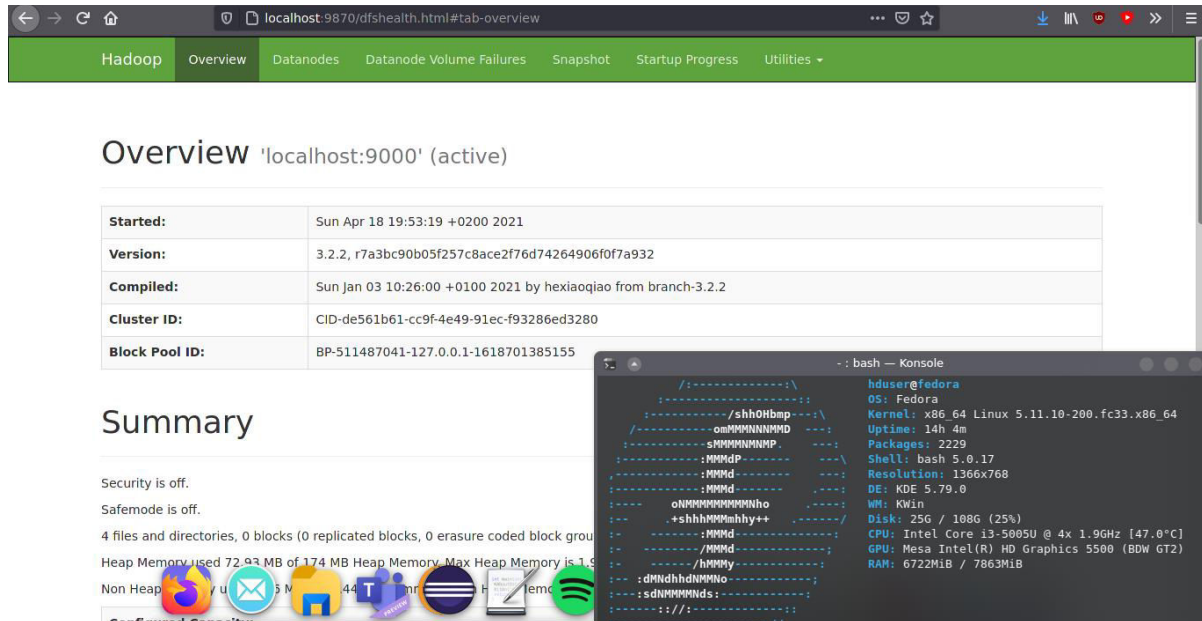
1. Introduction

In this report I explain the process of carrying out the given assignment in a concise manner, including the steps followed, the code written, and the problems faced along the way. Proof of work is provided for any part of the assignment that might not be obvious from the attached files.

2. Installation process

[Hadoop 3.2.2](#) (the latest stable version) was installed on a personal machine running Fedora 33, set up to run in pseudo-distributed mode. A secondary user “hduser” as well as a user group were created in order to leave my usual working environment untouched, though I did have to disable IPv6 system-wide.

Despite the recommendation to download a different distribution such as Cloudera the installation process was straightforward, using a [tutorial](#) and the relevant [documentation](#). The only problems faced were due to the tutorial being written for an older version of Hadoop, which were overcome quickly by learning the new paths for the relevant configuration files and binaries.



The screenshot shows the Hadoop Overview page for 'localhost:9000' (active). The page includes a table with the following information:

Property	Value
Started:	Sun Apr 18 19:53:19 +0200 2021
Version:	3.2.2, r7a3bc90b05f257c8ace2f76d74264906f0f7a932
Compiled:	Sun Jan 03 10:26:00 +0100 2021 by hexiaogiao from branch-3.2.2
Cluster ID:	CID-de561b61-cc9f-4e49-91ec-f93286ed3280
Block Pool ID:	BP-511487041-127.0.0.1-1618701385155

Below the table is a 'Summary' section with the following text:

Security is off.
Safemode is off.
4 files and directories, 0 blocks (0 replicated blocks, 0 erasure coded block groups)
Heap Memory used 72.93 MB of 174 MB Heap Memory. Max Heap Memory is 1.8 GB
Non-Heap Memory used 44.0 MB of 144.0 MB Non-Heap Memory. Max Non-Heap Memory is 1.8 GB
Configured Capacity: 1.8 TB

Overlaid on the bottom right is a terminal window showing system information for 'hduser@fedora':

```
OS: Fedora
Kernel: x86_64 Linux 5.11.10-200.fc33.x86_64
Uptime: 14h 4m
Packages: 2229
Shell: bash 5.0.17
Resolution: 1366x768
DE: KDE 5.79.0
WM: KWin
Disk: 25G / 108G (25%)
CPU: Intel Core i3-5005U @ 4x 1.9GHz [47.0°C]
GPU: Mesa Intel(R) HD Graphics 5500 (BDW GT2)
RAM: 6722MiB / 7863MiB
```

3. Dataset generation

The utility “datagen.py” was written for this part of the assignment.

datagen.py: k-means clustering dataset generator
USAGE: ./datagen.py <centers_file> (-v)

Required arguments:
<centers_file>: File containing an arbitrary amount of 2D data points to be used as centers.

Optional arguments:
-v: Display visualisation of data points at finish time.

The centers $[(-5, 13), (20, 7), (6, -1)]$ were chosen arbitrarily and can be found in the “centers_og.txt” file.

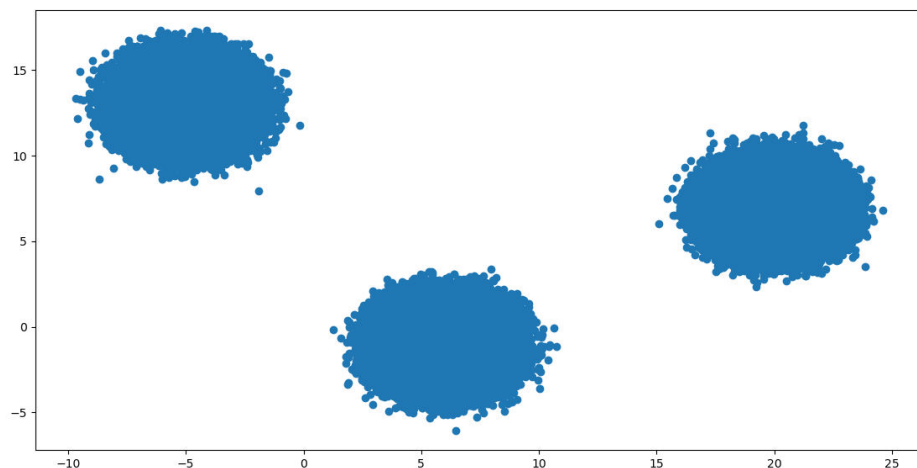
datagen.py outputs to STDOUT. Therefore, for further usage of its results, its output was redirected to a text file as such:

```
$ ./datagen.py centers_og.txt -v >> dataset.txt
```

At first the [make_blobs](#) function of the scikit-learn library was employed, though it did not satisfy the requirement for a right skewed distribution of the distances of the data points to be generated. Due to this, the final version of the utility makes use of the [skewnorm](#) function of the SciPy library.

```
for center in centers:
    # Generate 800k right skewed distances (a = 5), twice the amount of
    # points per center
    distances = skewnorm.rvs(5, size = 800000)
    for i in range(0, 800000, 2):
        # Use random distances to place points around centers
        x = center[0] + [-1,1][random.randrange(2)] * distances[i]
        y = center[1] + [-1,1][random.randrange(2)] * distances[i + 1]
        points.append((x, y))
```

In order to achieve the required result skewnorm (with a > 0 for the needed distribution) produces double the number of distances of the number of points assigned to each center (hard-coded to 400 thousand, as it leads to a nice, round, perfectly divisible sum of points). Then, each distance is randomly assigned a positive or negative value in order to non-deterministically spread out the points all around the center.



Indeed, the visualization produced using matplotlib shows that the intended result was achieved: It should also be mentioned that a small attempt was made to generalize the utility of datagen.py; it is able to accept

an arbitrary number of centers to generate points for.

4. Moving the files to HDFS

In order to run the MapReduce job presented in the next section the file “dataset.txt” needed to be moved to HDFS.

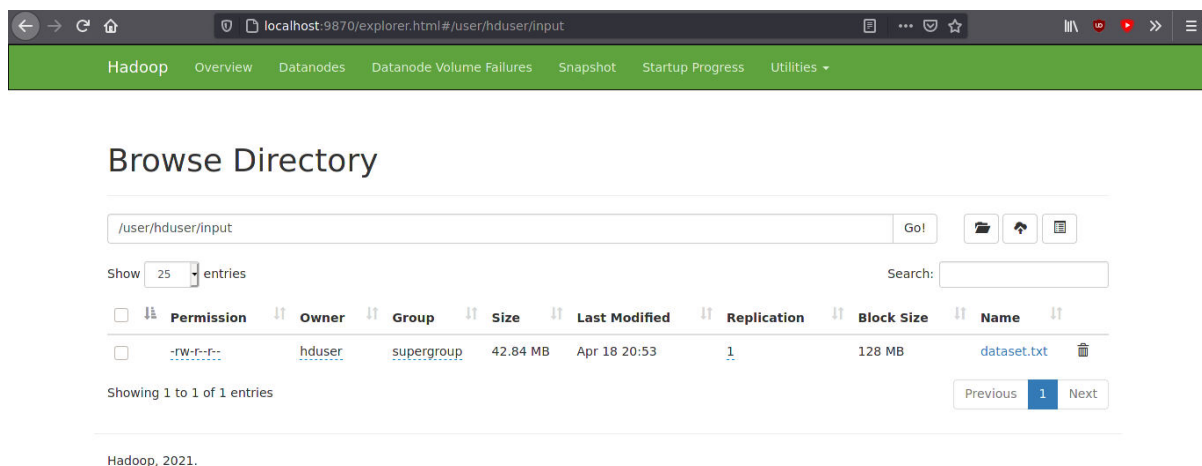
First, a user directory was created:

```
$ bin/hdfs dfs -mkdir /user
$ bin/hdfs dfs -mkdir /user/hduser
```

After that, an input directory was created and the file was moved to it:

```
$ bin/hdfs dfs -mkdir input
$ bin/hdfs dfs -put ~/hadoop/dataset.txt input
```

The successful result of the above operations can be seen in the NameNode’s web interface:



5. The MapReduce job

The Hadoop environment might be characterized as “Java-first”. Despite that, as well as my relative inexperience with the language, I chose to write the MapReduce job in Python, as there were more online resources for the k-means algorithm for it. Both the mapper and the reducer were written, like datagen, to work with an arbitrary number of centers and data points in a two-dimensional space.

During the writing of the MapReduce job Michael Noll’s tutorial “[Writing An Hadoop MapReduce Program In Python](#)” proved indispensable.

The “mapper.py” program implements the “Map” part of the paradigm. In order to run successfully it expects the dataset file in its STDIN, as well as the centers file as an argument. The mapping operation happens in these lines of code:

```
# Calculate distance from every center to select nearest one
euclidean_center = 0
center_distance = 999999999
for center in centers:
    distance = euclidean_distance(x, y, center[0], center[1])
    if center_distance > distance:
```

```

    euclidean_center = centers.index(center)
    center_distance = distance

# Reducer takes it from here
print(str(x) + ',' + str(y) + ',' + str(euclidean_center))

```

An unreasonably high distance is typed in as default. During runtime it updates, at first to match the real distance from center 0 and then, if applicable, to match the distance from the nearest center. The result is printed in STDOUT to be pipelined into the reducer.

The “reducer.py” program implements the “Reduce” part of the paradigm. It takes the result of the mapper in its STDIN and prints the recalculated coordinates of the centers as its output. These two programs together constitute, in essence, a singular iteration of the k-means algorithm.

```

# Storage space for map results
# Format: [<center_id>, <x_sum>, <y_sum>, <n_observations>], ...]
mapresults = []

...

# If the center exists in mapresults[] add line data
# If not create entry
flag = 0
for cluster in mapresults:
    if cluster[0] == center:
        cluster[1] += x
        cluster[2] += y
        cluster[3] += 1
        flag = 1
if flag == 0:
    mapresults.append([center, x, y, 1])

for cluster in mapresults:
    print(str(cluster[1]/cluster[3]) + ',' + str(cluster[2]/cluster[3]))

```

The mapresults[] list stores nested lists for each cluster, containing their numbers, sum of x and y coordinates of attached points and the number of points attached to them, as provided by the mapping operation. The reducer iterates on each line received from the mapper, tries to find the relevant cluster in mapresults[] and either adds the data to it or appends a new nested list for it. Finally, the coordinates of the new centers are printed to STDOUT, them being the mean of all the coordinate values for each center.

Before running the job on Hadoop the programs were tested locally to ensure that issues were not present. Their output has been redirected to the two text files found in the “testresults” directory.

In order to run the MapReduce job on my configuration I used the following shell command from Hadoop’s installation directory.

```

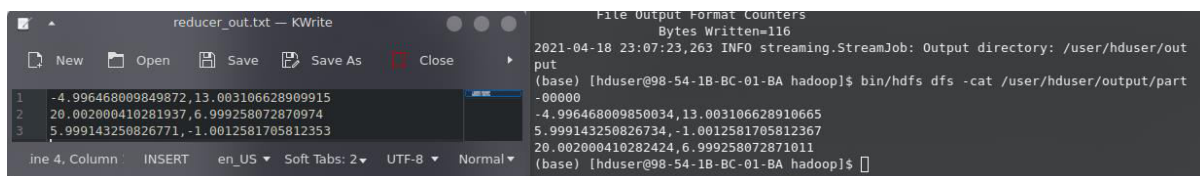
$ bin/hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.2.2.jar \
-file /home/hduser/hadoop/centers_og.txt \
-file /home/hduser/hadoop/mapper.py \
-mapper 'mapper.py centers_og.txt' \
-file /home/hduser/hadoop/reducer.py \
-reducer reducer.py \
-input /user/hduser/input/dataset.txt \
-output /user/hduser/output

```

The file paths may need to be altered for testing in another configuration. In order for Python to run on Hadoop the Streaming API is employed.

The result was successful, as can be ascertained from the screenshots below. There is a small difference between the coordinates produced during testing and the Hadoop job, which is completely expected and due to floating-point error.

```
2021-04-18 23:06:31,901 INFO mapreduce.Job: Running job: job_1618777985158_0008
2021-04-18 23:06:43,302 INFO mapreduce.Job: Job job_1618777985158_0008 running in uber mode : false
2021-04-18 23:06:43,307 INFO mapreduce.Job: map 0% reduce 0%
2021-04-18 23:07:06,803 INFO mapreduce.Job: map 33% reduce 0%
2021-04-18 23:07:07,818 INFO mapreduce.Job: map 67% reduce 0%
2021-04-18 23:07:08,844 INFO mapreduce.Job: map 100% reduce 0%
2021-04-18 23:07:22,001 INFO mapreduce.Job: map 100% reduce 100%
2021-04-18 23:07:23,025 INFO mapreduce.Job: Job job_1618777985158_0008 completed successfully
2021-04-18 23:07:23,263 INFO mapreduce.Job: Counters: 54
      File System Counters
        FILE: Number of bytes read=50920157
        FILE: Number of bytes written=102556137
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
```



```
reducer_out.txt -- KWrite
File Output Format Counters
  Bytes Written=116
2021-04-18 23:07:23,263 INFO streaming.StreamJob: Output directory: /user/hduser/output
put
(base) [hduser@98-54-1B-BC-01-BA hadoop]$ bin/hdfs dfs -cat /user/hduser/output/part
-00000
-4.996468009850034,13.003106628910665
5.999143250826734,-1.0012581705812367
20.002000410282424,6.999258072871011
(base) [hduser@98-54-1B-BC-01-BA hadoop]$
```

Finally, the output folder was downloaded from HDFS to my local filesystem using the command below. It is attached along with the rest of the assignment files.

```
$ bin/hadoop fs -get /user/hduser/output ~
```

6. Conclusion

Through this assignment I managed to get a first impression of the capabilities of the Hadoop framework, as well as the MapReduce paradigm. I gained a newfound appreciation for Python and its conciseness compared to Java, as well as its abstractions which made the implementation of this assignment considerably quicker. I was also able to get acquainted with the k-means clustering algorithm, this being my first foray into the world of machine learning.

Due to a fault of my own I was unaware at the time of writing that this assignment was intended to be carried out by a pair of people and, therefore, did it all by myself. Considering the fruitful results, and pending Prof. Chatziantoniou's approval, it is my hope that I will be able to continue as a team of one for the forthcoming assignments as well.