LARGE SCALE DATA MANAGEMENT COURSE

Lamprakis Panagiotis (03003142)

PhD Candidate – AILS

Contents

Introduction

In scope of the Large-Scale Data Management course, I have set up a mini cluster of two nodes using VirtualBox[1]. I have created a master and a worker node and installed Spark and Hadoop on them. The images (1 & 2) below verify the successful installation of Spart and Hadoop on both instances. Also, in the images below, it's noticeable that passwordless ssh between master & worker has been established.

The code for this homework can be found in [PanosLam/DSML-NTUA-Big-Data]() GitHub repo.

```
osboxes@master:~$ hadoop version
Hadoop 3.3.6
Source code repository https://github.com/apache/hadoop.git -r 1be78238728da9266a4f88195058f08fd012bf9c
Compiled by ubuntu on 2023-06-18T08:22Z
Compiled on platform linux-x86_64
Compiled with protoc 3.7.1
From source with checksum 5652179ad55f76cb287d9c633bb53bbd
This command was run using /home/osboxes/hadoop-3.3.6/share/hadoop/common/hadoop-common-3.3.6.jar
osboxes@master:~$ spark-submit --version
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.1
      /_/

Using Scala version 2.12.18, OpenJDK 64-Bit Server VM, 1.8.0_402
Branch HEAD
Compiled by user heartsavior on 2024-02-15T11:24:58Z
Revision fd86f85e181fc2dc0f50a096855acf83a6cc5d9c
Url https://github.com/apache/spark
Type --help for more information.
osboxes@master:~$ ssh-slave
Welcome to Ubuntu 22.04 LTS (GNU/Linux 5.15.0-25-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sun Jun  2 05:25:44 PM UTC 2024

  System load:    1.47265625          Processes:               132
  Usage of /home: 1.4% of 249.65GB    Users logged in:         1
  Memory usage:   10%                 IPv4 address for enp0s3: 192.168.1.20
  Swap usage:     0%

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

199 updates can be applied immediately.
104 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable


Last login: Sun Jun  2 17:23:55 2024 from 192.168.1.236
osboxes@worker:~$ hadoop version
Hadoop 3.3.6
Source code repository https://github.com/apache/hadoop.git -r 1be78238728da9266a4f88195058f08fd012bf9c
Compiled by ubuntu on 2023-06-18T08:22Z
Compiled on platform linux-x86_64
Compiled with protoc 3.7.1
From source with checksum 5652179ad55f76cb287d9c633bb53bbd
This command was run using /home/osboxes/hadoop-3.3.6/share/hadoop/common/hadoop-common-3.3.6.jar
osboxes@worker:~$
```

Image 1 Hadoop & Spark versions in Master Node

Master's private IP is *192.168.1.236* (verified via *ifconfig* command), while worker's private IP is *192.168.1.20*.

```
This command was run using /home/osboxes/hadoop-3.3.6/share/hadoop/common/hadoop-common-3.3.6.jar
osboxes@master:~$ spark-submit --version
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.1
      /_/

Using Scala version 2.12.18, OpenJDK 64-Bit Server VM, 1.8.0_402
Branch HEAD
Compiled by user heartsavior on 2024-02-15T11:24:58Z
Revision fd86f85e181fc2dc0f50a096855acf83a6cc5d9c
Url https://github.com/apache/spark
Type --help for more information.
osboxes@master:~$ ssh-slave
Welcome to Ubuntu 22.04 LTS (GNU/Linux 5.15.0-25-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sun Jun  2 05:25:44 PM UTC 2024

  System load:    1.47265625        Processes:               132
  Usage of /home: 1.4% of 249.65GB  Users logged in:         1
  Memory usage:   10%               IPv4 address for enp0s3: 192.168.1.20
  Swap usage:     0%

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

199 updates can be applied immediately.
104 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable


Last login: Sun Jun  2 17:23:55 2024 from 192.168.1.236
osboxes@worker:~$ hadoop version
Hadoop 3.3.6
Source code repository https://github.com/apache/hadoop.git -r 1be78238728da9266a4f88195058f08fd012bf9c
Compiled by ubuntu on 2023-06-18T08:22Z
Compiled on platform linux-x86_64
Compiled with protoc 3.7.1
From source with checksum 5652179ad55f76cb287d9c633bb53bbd
This command was run using /home/osboxes/hadoop-3.3.6/share/hadoop/common/hadoop-common-3.3.6.jar
osboxes@worker:~$ spark-submit --version
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.1
      /_/

Using Scala version 2.12.18, OpenJDK 64-Bit Server VM, 1.8.0_402
Branch HEAD
Compiled by user heartsavior on 2024-02-15T11:24:58Z
Revision fd86f85e181fc2dc0f50a096855acf83a6cc5d9c
Url https://github.com/apache/spark
Type --help for more information.
osboxes@worker:~$
```

Image 2 Hadoop & Spark versions in Worker Node

Hadoop web interface can be accessed via: http://192.168.1.236:9870/dfshealth.html#tab-datanode hyperlink [2].

Spark web UI for Master node can be accessed via: http://192.168.1.236:8080/ hyperlink, and via http://192.168.1.20:8081/ for the Worker Node.

In order to verify the code, I have setup Spark locally in my PC and run the experiments over a small but representative amount of data. The final results come by executing the code in the cluster.

Creating a folder on HDFS.

```
osboxes@master:~$ hadoop fs -mkdir /files
osboxes@master:~$ hadoop fs -ls /
Found 1 items
drwxr-xr-x   - osboxes supergroup          0 2024-06-02 20:51 /files
osboxes@master:~$ hadoop fs -chmod 777 /files
osboxes@master:~$ hadoop fs -ls /
Found 1 items
drwxrwxrwx   - osboxes supergroup          0 2024-06-02 20:51 /files
osboxes@master:~$
```

Copying the files from local (PC) into the Cloud (VirtualBox – Master instance)

```
panoslam@LAPTOP-56BVTUE4:[redacted]/Διαχειριση-Δεδομενων-Μεγαλης-Κλιμακας/Εργασια$ scp query-4-data.csv osboxes@192.168.1.236:/home
/osboxes/local-datasets
osboxes@192.168.1.236's password:
query-4-data.csv                                   100% 1387     1.1MB/s   00:00
panoslam@LAPTOP-56BVTUE4:[redacted]/Διαχειριση-Δεδομενων-Μεγαλης-Κλιμακας/Εργασια$
```

In case the DataNode is not up and running we need to perform the following steps:

1. Run *stop-dfs.sh*
2. Run *mv /home/osboxes/hdfsdata /home/osboxes/hdfsdata_backup* (remove DataNode directories and backup them)
3. Run *start-dfs.sh*

Make a directory in HDFS and upload a file from Master local file system into HDFS.

```
osboxes@master:~/local-datasets$ hadoop fs -mkdir /files
osboxes@master:~/local-datasets$ hadoop fs -put query-4-data.csv /files/
osboxes@master:~/local-datasets$ hadoop fs -ls /files/
Found 1 items
-rw-r--r--   2 osboxes supergroup       1387 2024-06-02 21:17 /files/query-4-data.csv
osboxes@master:~/local-datasets$
```

The image below shows that all necessary datasets have been uploaded from local PC to Master FS.

```
panoslam@LAPTOP-56BVTUE4:[redacted]/Διαχειριση-Δεδομενων-Μεγαλης-Κλιμακας/Εργασια$
panoslam@LAPTOP-56BVTUE4:[redacted]/Διαχειριση-Δεδομενων-Μεγαλης-Κλιμακας/Εργασια$
panoslam@LAPTOP-56BVTUE4:[redacted]/Διαχειριση-Δεδομενων-Μεγαλης-Κλιμακας/Εργασια$ scp query-4-data.csv osboxes@192.168.1.236:/home
/osboxes/local-datasets
osboxes@192.168.1.236's password:
query-4-data.csv                                   100% 1387     1.1MB/s   00:00
panoslam@LAPTOP-56BVTUE4:[redacted]/Διαχειριση-Δεδομενων-Μεγαλης-Κλιμακας/Εργασια$
panoslam@LAPTOP-56BVTUE4:[redacted]/Διαχειριση-Δεδομενων-Μεγαλης-Κλιμακας/Εργασια$
panoslam@LAPTOP-56BVTUE4:[redacted]/Διαχειριση-Δεδομενων-Μεγαλης-Κλιμακας/Εργασια$ scp Crime_Data_from_2010_to_2019.csv osboxes@192
.168.1.236:/home/osboxes/local-datasets
osboxes@192.168.1.236's password:
Crime_Data_from_2010_to_2019.csv                   100%  512MB  38.6MB/s   00:13
panoslam@LAPTOP-56BVTUE4:[redacted]/Διαχειριση-Δεδομενων-Μεγαλης-Κλιμακας/Εργασια$ scp Crime_Data_from_2020_to_Present.csv osboxes@
192.168.1.236:/home/osboxes/local-datasets
osboxes@192.168.1.236's password:
Crime_Data_from_2020_to_Present.csv                100%  230MB  43.0MB/s   00:05
panoslam@LAPTOP-56BVTUE4:[redacted]/Διαχειριση-Δεδομενων-Μεγαλης-Κλιμακας/Εργασια$ cd median-household-income-2015/
panoslam@LAPTOP-56BVTUE4:[redacted]/Διαχειριση-Δεδομενων-Μεγαλης-Κλιμακας/Εργασια/median-household-income-2015$ scp revgecoding.csv
 osboxes@192.168.1.236:/home/osboxes/local-datasets
osboxes@192.168.1.236's password:
revgecoding.csv                                    100%  876KB  43.2MB/s   00:00
panoslam@LAPTOP-56BVTUE4:[redacted]/Διαχειριση-Δεδομενων-Μεγαλης-Κλιμακας/Εργασια/median-household-income-2015$ cd income/
panoslam@LAPTOP-56BVTUE4:[redacted]/Διαχειριση-Δεδομενων-Μεγαλης-Κλιμακας/Εργασια/median-household-income-2015/income$ scp LA_incom
e_2015.csv osboxes@192.168.1.236:/home/osboxes/local-datasets
osboxes@192.168.1.236's password:
LA_income_2015.csv                                 100%   13KB   8.8MB/s   00:00
panoslam@LAPTOP-56BVTUE4:[redacted]/Διαχειριση-Δεδομενων-Μεγαλης-Κλιμακας/Εργασια/median-household-income-2015/income$
```

The following image shows the datasets in Master FS.

```
osboxes@master:~/local-datasets$ ll
total 760920
drwxrwxrwx  2 osboxes osboxes      4096 Jun  2 21:21 ./
drwxr-x--- 11 osboxes osboxes      4096 Jun  2 21:10 ../
-rwxrwxr-x  1 osboxes osboxes 537190637 Jun  2 21:19 Crime_Data_from_2010_to_2019.csv*
-rwxrwxr-x  1 osboxes osboxes 241051722 Jun  2 21:20 Crime_Data_from_2020_to_Present.csv*
-rwxrwxr-x  1 osboxes osboxes     12859 Jun  2 21:21 LA_income_2015.csv*
-rwxrwxr-x  1 osboxes osboxes      1387 Jun  2 21:01 query-4-data.csv*
-rwxrwxr-x  1 osboxes osboxes    897062 Jun  2 21:20 revgecoding.csv*
osboxes@master:~/local-datasets$
```

Finally, the image below verifies that datasets have been uploaded successfully to HDFS.

```
osboxes@master:~/local-datasets$ hadoop fs -put Crime_Data_from_2010_to_2019.csv /files/
osboxes@master:~/local-datasets$ hadoop fs -put Crime_Data_from_2020_to_Present.csv /files/
osboxes@master:~/local-datasets$ hadoop fs -put LA_income_2015.csv /files/
osboxes@master:~/local-datasets$ hadoop fs -put revgecoding.csv /files/
osboxes@master:~/local-datasets$ hadoop fs -ls /files/
Found 5 items
-rw-r--r--   2 osboxes supergroup  537190637 2024-06-02 21:31 /files/Crime_Data_from_2010_to_2019.csv
-rw-r--r--   2 osboxes supergroup  241051722 2024-06-02 21:31 /files/Crime_Data_from_2020_to_Present.csv
-rw-r--r--   2 osboxes supergroup      12859 2024-06-02 21:31 /files/LA_income_2015.csv
-rw-r--r--   2 osboxes supergroup       1387 2024-06-02 21:17 /files/query-4-data.csv
-rw-r--r--   2 osboxes supergroup     897062 2024-06-02 21:31 /files/revgecoding.csv
osboxes@master:~/local-datasets$
```

Verify HDFS uri as created in python code to be used for saving parquet files in HDFS.

```
osboxes@master:~/local-datasets$ hdfs dfs -ls hdfs://192.168.1.236:9000/files
Found 5 items
-rw-r--r--   2 osboxes supergroup  537190637 2024-06-02 21:31 hdfs://192.168.1.236:9000/files/Crime_Data_from_2010_to_2019.csv
-rw-r--r--   2 osboxes supergroup  241051722 2024-06-02 21:31 hdfs://192.168.1.236:9000/files/Crime_Data_from_2020_to_Present.csv
-rw-r--r--   2 osboxes supergroup      12859 2024-06-02 21:31 hdfs://192.168.1.236:9000/files/LA_income_2015.csv
-rw-r--r--   2 osboxes supergroup       1387 2024-06-02 21:17 hdfs://192.168.1.236:9000/files/query-4-data.csv
-rw-r--r--   2 osboxes supergroup     897062 2024-06-02 21:31 hdfs://192.168.1.236:9000/files/revgecoding.csv
```

Work

Query 1

We implement the first query in using both SQL API & DataFrame and we read the data from the CSV file and the parquet file. The table below shows the total execution times for all possible combinations.

	CSV	PARQUET
SQL API	55.259 sec	26.472 sec
DATAFRAME	54.378 sec	25.521 sec

We observe that in general, using the parquet format results in 52-53% execution time cut compared to the CSV cases. This happens for the following reasons:

1. Parquet file formats tend to give better read times because of their columnar storage and their included schema information as opposed to CSV files which are row-based storage and need schema inference. This means that when Spark needs to read specific columns, it’s faster for parquet cases compared to csv, in which Spark needs to read whole rows.
2. Parquet files are compressed, leading to smaller data sizes being transmitted over the network compared to the csv case. This gives an improvement on the network I/O latency.

3. Computations held over parquet data are more efficient, since batches of data can be processed in parallel (vectorized processing).

It is evident that a parquet file format ends up with more benefits.

CRIME_YEAR	CRIME_MONTH	count	counter
2010	3	17595	1
2010	7	17520	2
2010	5	17338	3
2011	8	17139	1
2011	5	17050	2
2011	3	16951	3
2012	8	17696	1
2012	10	17477	2
2012	5	17391	3
2013	8	17329	1
2013	7	16714	2
2013	5	16671	3
2014	7	17456	1
2014	10	17300	2
2014	12	17076	3
2015	8	19134	1
2015	10	19065	2
2015	7	18755	3
2016	8	19834	1
2016	10	19678	2
2016	7	19343	3
2017	10	20436	1
2017	8	20127	2
2017	7	20034	3
2018	5	20277	1
2018	7	19998	2
2018	10	19851	3
2019	7	19349	1
2019	8	19094	2
2019	3	18967	3
2020	1	18512	1
2020	2	17443	2
2020	7	17257	3
2021	10	19191	1
2021	7	18954	2
2021	11	18666	3
2022	5	20784	1
2022	8	20585	2
2022	6	20418	3
2023	10	20350	1
2023	8	20345	2
2023	1	20257	3
2024	1	20015	1
2024	2	18009	2
2024	3	17186	3

Image 3 DataFrame results

year	month	crime_total	ranking
2010	3	17595	1
2010	7	17520	2
2010	5	17338	3
2011	8	17139	1
2011	5	17050	2
2011	3	16951	3
2012	8	17696	1
2012	10	17477	2
2012	5	17391	3
2013	8	17329	1
2013	7	16714	2
2013	5	16671	3
2014	7	17456	1
2014	10	17300	2
2014	12	17076	3
2015	8	19134	1
2015	10	19065	2
2015	7	18755	3
2016	8	19834	1
2016	10	19678	2
2016	7	19343	3
2017	10	20436	1
2017	8	20127	2
2017	7	20034	3
2018	5	20277	1
2018	7	19998	2
2018	10	19851	3
2019	7	19349	1
2019	8	19094	2
2019	3	18967	3
2020	1	18512	1
2020	2	17443	2
2020	7	17257	3
2021	10	19191	1
2021	7	18954	2
2021	11	18666	3
2022	5	20784	1
2022	8	20585	2
2022	6	20418	3
2023	10	20350	1
2023	8	20345	2
2023	1	20257	3
2024	1	20015	1
2024	2	18009	2
2024	3	17186	3

Image 4 SQL API results

Query 2

```
+------------------------+------+
|time_of_day             |count |
+------------------------+------+
|midnight (21:00-04:59)  |243374|
|night (17:00-20:59)     |191792|
|afternoon (12:00-16:59)|151564|
|morning (05:00-11:59)   |126611|
+------------------------+------+
```

Image 5 DataFrame results

```
('midnight (21:00-04:59)', 243374)
('night (17:00-20:59)', 191792)
('afternoon (12:00-16:59)', 151564)
('morning (05:00-11:59)', 126611)
```

Image 6 RDD results

	RDD	DATAFRAME
EXECUTION TIME (SEC)	74.997	45.779

It is easy to observe that using DataFrames over RDDs we achieve an improvement of almost 40% (reduction) in execution time. This is because RDDs offer a low-level API to interact with the data and it doesn't come with optimizations, compared to DataFrames which use the Spark Catalyst optimizer. Moreover, RDDs store the data in Java objects, which can be inefficient, while DataFrames store the data in binary format, which reduces serialization/deserialization costs, thus improving the execution time.

Query 3

Due to limited time, I was not able to experiment with the different join implementations offered by Spark API. However, I will attempt to guess the best join implementation based on the nature of the different Spark implementations and the dataset under analysis.

The **broadcast** join broadcasts the small dataset to the worker node(s) and the join is performed locally on each node. This join type is effective when merging a small with a large dataset.

The **merge** join sorts both data sets by key and then merges them. This join type is efficient for large datasets that are already sorted or can be sorted efficiently.

The **shuffle hash** join shuffles the data to the worker node(s) and performs the join on the shuffled data. This join type is a good option when the join keys fit into memory and the dataset is evenly distributed.

The **shuffle and replicate NL** join takes each row of one dataset and compares it to every row in the other dataset to perform the join.

Given the above, I believe that **broadcast** join would be the most appropriate one, since we do have a small data set (produced by merging revgeocoding over the zip codes) which is joined with a large one (2015 crime dataset). Also, the **shuffle hash** join would work too, since the keys can fit into worker node memory.

I would expect **Merge** & **shuffle and replicate NL** joins to end up with worse execution times compared to the other two strategies.

Victim_Descent_Detailed	Total Victims
Hispanic/Latin/Me...	1526
Black	1098
White	700
Other	390
Other Asian	101
Unknown	65
Hawaiian	32
American Indian/A...	25
Korean	12
Chinese	6
Japanese	5
Pacific Islander	5
Asian Indian	4
Filipino	3
Guamanian	3
Vietnamese	3
Laotian	1
Samoan	1

Image 7 DataFrame results

Query 4

No time to implement the broadcast & repartition join algorithms. However, below there are the results by leveraging the join strategy selected by Catalyst Optimizer for both DataFrame and RDD implementations.

DIVISION	incidents total	AVERAGE_DISTANCE
77TH STREET	17019	2.688
SOUTHEAST	12942	2.105
NEWTON	9846	2.019
SOUTHWEST	8912	2.7
HOLLENBECK	6202	2.652
HARBOR	5621	4.082
RAMPART	5115	1.575
MISSION	4504	4.715
OLYMPIC	4424	1.822
NORTHEAST	3920	3.905
FOOTHILL	3774	3.803
HOLLYWOOD	3641	1.46
CENTRAL	3614	1.138
WILSHIRE	3525	2.314
NORTH HOLLYWOOD	3466	2.719
WEST VALLEY	2902	3.53
VAN NUYS	2733	2.222
PACIFIC	2708	3.729
DEVONSHIRE	2471	4.01
TOPANGA	2283	3.486
WEST LOS ANGELES	1541	4.243

Image 8 DataFrame results

division	average_distance	incidents_total
77TH STREET	45747.717000000106	17019
SOUTHEAST	27247.22999999984	12942
NEWTON	19877.977000000083	9846
SOUTHWEST	24058.91200000001	8912
HOLLENBECK	16449.319000000047	6202
HARBOR	22943.546000000042	5621
RAMPART	8058.242999999956	5115
MISSION	21238.392000000113	4504
OLYMPIC	8059.666000000035	4424
NORTHEAST	15305.701999999948	3920
FOOTHILL	14353.190000000053	3774
HOLLYWOOD	5315.0450000000255	3641
CENTRAL	4113.179000000007	3614
WILSHIRE	8156.621999999998	3525
NORTH HOLLYWOOD	9423.159000000007	3466
WEST VALLEY	10243.582999999982	2902
VAN NUYS	6071.758000000035	2733
PACIFIC	10097.389000000021	2708
DEVONSHIRE	9909.019999999997	2471
TOPANGA	7958.548000000008	2283
WEST LOS ANGELES	6539.0819999999985	1541

Image 9 RDD results

What differs between the results of the two implementations here is the numbers in the average distance (not rounded for the case of RDD). I must have done a mistake and I may have confused the longitude and latitude coordinations for the RDD case. However, due to time limitations I'm not in position of fixing it right now.

Appendix

[1] Using the instructions provided in ***01_lab1-virtualbox*** & ***0_Preparatory_lab_virtual-box***.

[2] I have faced lots of problems by setting up the cluster in VirtualBox. The private IPs of main and worker nodes were constantly changing everytime I was powering off the machines. Additionally, out of nowhere, Hadoop starting prompting me an error about mismatched cluster ids.