

Προχωρημένα Θέματα Βάσεων Δεδομένων

Εξαμηνιαία εργασία

*

Λουκας Αγγελος el19877

Μαντζαφινης Αλεξανδρος el18057

Github Repository ([here](#))

- [Ερώτημα 1](#)
- [Ερώτημα 2](#)
- [Ερώτημα 3](#)
- [Ερώτημα 4](#)
- [Ερώτημα 5](#)
- [Ερώτημα 6](#)
- [Ερώτημα 7](#)

Ερώτημα 1

Ακολουθώντας τον οδηγό εγκατάστασης δημιουργήσαμε ένα πλήρως κατανεμημένο περιβάλλον, με δύο κόμβους(master,worker), με την βοήθεια του okeanos-knossos, ο οποίος μας παρείχε με τους απαραίτητους πόρους. Παρακάτω είναι οι web εφαρμογές των HDFS, YARN και Spark History Server:

HDFS με 2 live nodes:

Overview 'okeanos-master:54310' (✓active)

Started:	Thu Jan 11 10:43:37 +0200 2024
Version:	3.3.6, r1be78238728da9266a4f88195058f08fd012bf9c
Compiled:	Sun Jun 18 11:22:00 +0300 2023 by ubuntu from (HEAD detached at release-3.3.6-RC1)
Cluster ID:	CID-7d0e6997-0c53-4bdc-b406-06fc4d1470e4
Block Pool ID:	BP-319276962-192.168.0.2-1704962608300

Summary

Security is off.

Safemode is off.

1 files and directories, 0 blocks (0 replicated blocks, 0 erasure coded block groups) = 1 total filesystem object(s).

Heap Memory used 76.08 MB of 194 MB Heap Memory. Max Heap Memory is 1.94 GB.

Non Heap Memory used 49.47 MB of 53.13 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	58.78 GB
Configured Remote Capacity:	0 B
DFS Used:	48 KB (0%)
Non DFS Used:	13.82 GB
DFS Remaining:	41.93 GB (71.34%)
Block Pool Used:	48 KB (0%)
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	2 (Decommissioned: 0, In Maintenance: 0)
Dead Nodes	0 (Decommissioned: 0, In Maintenance: 0)
Decommissioning Nodes	0
Entering Maintenance Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion (including replicas)	0
Block Deletion Start Time	Thu Jan 11 10:43:37 +0200 2024
Last Checkpoint Time	Thu Jan 11 10:43:28 +0200 2024
Enabled Erasure Coding Policies	RS-6-3-1024k

Ομοίως YARN:

 **Nodes of the cluster**

Logged in as: dr.who

Cluster Metrics	Apps Submitted		Apps Pending		Apps Running		Apps Completed		Containers Running		Used Resources		Total Resources		Reserved Resources		Physical Mem Used %		Physical Vcores Used %	
0	0	0	0	0	0	0	0	0	0	<memory:0 B, vCores:0>	<memory:12 GB, vCores:16>	<memory:0 B, vCores:0>	<memory:12 GB, vCores:16>	23	0	0	0	0	0	
Cluster Nodes Metrics	Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes													
2	0	0	0	0	0	0	0													
Scheduler Metrics	Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority	Scheduler Busy %														
Capacity Scheduler	[memory-mb (unit=M), vcores]	<memory:128, vCores:1>	<memory:6144, vCores:4>	0	0	0														
Show 20 ▾ entries	Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-report	Containers	Allocation Tags	Mem Used	Mem Avail	Phys Mem Used %	Vcores Used	Vcores Avail	Phys Vcores Used %	Version	Search:				
	default-rack	RUNNING	okeanos-master:44047	okeanos-master:8042	Thu Jan 11 10:44:13 +0200 2024	0	0 B	6 GB	29	0	8	3	3.3.6							
	default-rack	RUNNING	okeanos-worker:45635	okeanos-worker:8042	Thu Jan 11 10:44:14 +0200 2024	0	0 B	6 GB	16	0	8	0	3.3.6							

Showing 1 to 2 of 2 entries

First Previous Next Last

Kαι το Spark History Server:

 **History Server**

Event log directory: <http://okeanos-master:54310/spark.eventLog>

Last updated: 2024-01-11 10:53:16

Client local time zone: Europe/Athens

Version	App ID	App Name	Started	Spark User	Last Updated	Event Log
3.5.0	application_1704962660509_0001	PySparkShell	2024-01-11 10:46:49	user	2024-01-11 10:53:16	Download

Showing 1 to 1 of 1 entries

[Back to completed applications](#)

Ερώτημα 2

Στο συγκεκριμένο ερώτημα, στόχος μας ήταν να δημιουργήσουμε ένα DataFrame από το αρχικό σύνολο δεδομένων. Ο πρωταρχικός στόχος ήταν να προσαρμόσουμε τους τύπους δεδομένων ορισμένων στηλών σύμφωνα με τις προδιαγραφές που δόθηκαν. Τα αρχικά ονόματα των στηλών διατηρήθηκαν καθ' όλη τη διάρκεια αυτής της διαδικασίας.

Ο συνολικός αριθμός γραμμών στο σύνολο δεδομένων είναι **2973193**.

Παρακάτω φαίνεται ο κώδικας του αρχείου **question2.py** (Εικόνα 1)

```
1  from pyspark.sql import SparkSession
2  from pyspark.sql.functions import to_timestamp
3
4  # Create a Spark session
5  spark = SparkSession.builder.appName("DATAFRAME").getOrCreate()
6
7  # Define the date-time format
8  datetime_format = "MM/dd/yyyy hh:mm:ss a"
9
10 # HDFS paths to the data files
11 file_path1 = 'hdfs:///user/data/crime-data-from-2010-to-2019.csv'
12 file_path2 = 'hdfs:///user/data/crime-data-from-2020-to-present.csv'
13
14 # Read the data into DataFrames with inferred schema
15 df1 = spark.read.format("csv").option("header", "true")\
16     .option("inferSchema", "true").load(file_path1)
17 df2 = spark.read.format("csv").option("header", "true")\
18     .option("inferSchema", "true").load(file_path2)
19
20 # Convert the date-time strings to TimestampType and then to DateType
21 df1 = df1.withColumn("Date Rptd", to_timestamp("Date Rptd", datetime_format) \
22     .cast("date"))\
23     .withColumn("DATE OCC", to_timestamp("DATE OCC", datetime_format) \
24     .cast("date"))
25
26 df2 = df2.withColumn("Date Rptd", to_timestamp("Date Rptd", datetime_format) \
27     .cast("date"))\
28     .withColumn("DATE OCC", to_timestamp("DATE OCC", datetime_format) \
29     .cast("date"))
30
31 # Union the two DataFrames
32 df = df1.union(df2)
33
34 row_count = df.count()
35 print(f"Number of rows in the DataFrame: {row_count}")
36
37 for column, dtype in df.dtypes:
38     print(f"{column}, {dtype}")
39
40 spark.stop()
41
```

Εικόνα 1

Ο κώδικας ξεκινά με τη δημιουργία μιας συνεδρίας Spark, η οποία αποτελεί το σημείο εισόδου για την αλληλεπίδραση με τη λειτουργικότητα της Spark. Στη συνέχεια ορίζει τη μορφή ημερομηνίας-ώρας που

χρησιμοποιείται στα αρχεία δεδομένων και καθορίζει τις διευθύνσεις προς τα δύο αρχεία δεδομένων HDFS που περιέχουν δεδομένα εγκλημάτων από διαφορετικές χρονικές περιόδους. Μετά την ανάγνωση των δεδομένων, εφαρμόζει μετασχηματισμούς για τη μετατροπή των συμβολοσειρών ημερομηνίας-χρόνου στις στήλες "Date Rptd" και "DATE OCC". Τέλος, ο κώδικας εκτελεί μια πράξη ένωσης στα δύο DataFrames για να τα συνδυάσει σε ένα ενιαίο DataFrame (df).

Έξοδος DataFrame setup

```
Number of rows in the DataFrame: 2973193
DR_NO, int
Date Rptd, date
DATE OCC, date
TIME OCC, int
AREA , int
AREA NAME, string
Rpt Dist No, int
Part 1-2, int
Crm Cd, int
Crm Cd Desc, string
Mocodes, string
Vict Age, int
Vict Sex, string
Vict Descent, string
Premis Cd, int
Premis Desc, string
Weapon Used Cd, int
Weapon Desc, string
Status, string
Status Desc, string
Crm Cd 1, int
Crm Cd 2, int
Crm Cd 3, int
Crm Cd 4, int
LOCATION, string
Cross Street, string
LAT, double
LON, double
```

Ερώτημα 3

Για την υλοποίηση όλων των υπολοίπων ερωτημάτων δημιουργήσαμε την συνάρτηση **session_setup** (Εικόνα 2) που βρίσκεται στο αρχείο **dataframe_setup.py**. Η συνάρτηση αυτή παίρνει δύο μεταβλητές, τον αριθμό των executors που θέλουμε και το όνομα του Session μας. Ο κώδικας αρχικοποιεί το Session και διαβάζει τα αρχεία εισόδου μας που χρησιμοποιούνται σε όλα τα queries των υπόλοιπων ερωτημάτων.



```
1  #----- dataframe_setup.py -----
2
3  from pyspark.sql import SparkSession
4  from pyspark.sql.functions import to_timestamp
5
6  def session_setup(executors,MyappName):
7      # Create a Spark session
8      spark = SparkSession.builder \
9          .appName(MyappName) \
10         .config("spark.executor.instances", str(executors)) \
11         .getOrCreate()
12
13     # Define the date-time format
14     datetime_format = "MM/dd/yyyy hh:mm:ss a"
15
16     file_path1 = 'hdfs:///user/data/crime-data-from-2010-to-2019.csv'
17     file_path2 = 'hdfs:///user/data/crime-data-from-2020-to-present.csv'
18
19     df1 = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load(file_path1)
20     df2 = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load(file_path2)
21
22     # Convert the date-time strings to TimestampType and then to DateType
23     df1 = df1.withColumn("Date Rptd", to_timestamp("Date Rptd", datetime_format).cast("date"))\
24         .withColumn("DATE OCC", to_timestamp("DATE OCC", datetime_format).cast("date"))
25     df2 = df2.withColumn("Date Rptd", to_timestamp("Date Rptd", datetime_format).cast("date"))\
26         .withColumn("DATE OCC", to_timestamp("DATE OCC", datetime_format).cast("date"))
27
28     return (df1.union(df2),spark)
```

Εικόνα 2

To Query 1 με χρήση του DataFrame (αρχείο question3_DF.py)

```
● ● ●
```

```
1  from pyspark.sql import SparkSession
2  from pyspark.sql.functions import year, month, col, desc, rank
3  from pyspark.sql.window import Window
4  import time
5  from pythonScripts import dataframe_setup
6
7  df,spark=dataframe_setup.session_setup(4,"Query1_DF_implementation")
8
9  # Start the timer
10 start_time = time.time()
11
12  # Group by year and month, and count the crimes
13 crime_counts = df.groupBy(year("DATE OCC").alias("Year"), month("DATE OCC").alias("Month")) \
14  .count() \
15  .withColumnRenamed("count", "Crime Total")
16
17  # Define a window spec partitioned by year and ordered by crime total (descending)
18 windowSpec = Window.partitionBy("Year").orderBy(desc("Crime Total"))
19
20  # Apply the window spec to rank the months within each year
21 crime_counts = crime_counts.withColumn("Rank", rank().over(windowSpec))
22
23  # Filter for top 3 months for each year
24 top_months = crime_counts.filter(col("Rank") <= 3)
25  # Order by year and crime total
26 query1 = top_months.orderBy("Year", desc("Crime Total"))
27
28 # Show the result
29 query1.show(df.count(), truncate=False)
30
31
32  # End the timer and print execution time
33 end_time = time.time()
34 print(f"Execution time with 4 executors: {end_time - start_time} seconds")
35 spark.stop()
```

Εικόνα 3

Για την υλοποίηση του χρησιμοποιήσαμε το DF που φτιάξαμε στο προηγούμενο ερώτημα, το κάναμε GroupBy year και month, μετρήσαμε τα εγκλήματα, και ορίσαμε ένα window που διαχωρίζεται από το year και είναι ordered σε φθίνουσα σειρά με βάση τα εγκλήματα. Τέλος, «συμπτύξαμε» το window και το αλλαγμένο DF μας και το κάναμε rank έτσι ώστε να υπάρχουν μόνο οι 3 μήνες με τα περισσότερα εγκλήματα κάθε χρόνο. Η υλοποίηση έγινε με 4 executors όπως ζητήθηκε και το timer μας μέσα στον κώδικα μας ξεκίνησε την χρονομέτρηση μετά το διάβασμα των δεδομένων και τελείωσε αφού έγινε επιστροφή των αποτελεσμάτων.

Έξοδος με DataFrame

```
+---+---+---+---+
|Year|Month|Crime Total|Rank|
+---+---+---+---+
|2010|1   |15750    |1   |
|2010|3   |15653    |2   |
|2010|4   |15516    |3   |
|2011|1   |18269    |1   |
|2011|7   |18257    |2   |
|2011|8   |18041    |3   |
|2012|1   |22983    |1   |
|2012|8   |22832    |2   |
|2012|5   |22525    |3   |
|2013|8   |9112     |1   |
|2013|7   |8956     |2   |
|2013|1   |8682     |3   |
|2014|7   |12106    |1   |
|2014|8   |11990    |2   |
|2014|12  |11969    |3   |
|2015|8   |13857    |1   |
|2015|10  |13502    |2   |
|2015|9   |13365    |3   |
|2016|8   |14596    |1   |
|2016|7   |14479    |2   |
|2016|12  |14323    |3   |
|2017|18  |33585    |1   |
|2017|7   |33357    |2   |
|2017|8   |33297    |3   |
|2018|11  |15613    |1   |
|2018|10  |15341    |2   |
|2018|12  |15198    |3   |
|2019|7   |19122    |1   |
|2019|8   |18979    |2   |
|2019|3   |18856    |3   |
|2020|1   |6090     |1   |
|2020|2   |5869     |2   |
|2020|5   |5530     |3   |
|2021|7   |34744    |1   |
|2021|8   |33653    |2   |
|2021|10  |33639    |3   |
|2022|7   |16748    |1   |
|2022|8   |16526    |2   |
|2022|5   |15833    |3   |
|2023|7   |37594    |1   |
|2023|8   |37500    |2   |
|2023|10  |37033    |3   |
+---+---+---+---+
Execution time with 4 executors: 11.72415018081665 seconds
```

Εικόνα 4

Μέσω του Spark UI βλέπουμε χρόνο εκτέλεσης

Spark Jobs (?)

User: user

Total Uptime: 52 s

Scheduling Mode: FIFO

Completed Jobs: 9

Εικόνα 5

To Query 1 με χρήση SQL APIs (αρχείο question3_SQL.py)

```
 1  from pyspark.sql import SparkSession
 2  import time
 3  from pythonScripts import dataframe_setup
 4
 5  df,spark=dataframe_setup.session_setup(4,"Query1_SQL_implementation")
 6
 7  # Register the DataFrame as a SQL temporary view
 8  df.createOrReplaceTempView("crime_data")
 9
10 start_time = time.time()
11 # SQL query
12 query1_sql = """
13 SELECT Year, Month, Crime_Total, Rank
14 FROM (
15     SELECT YEAR(`DATE OCC`) AS Year, MONTH(`DATE OCC`) AS Month, COUNT(*) AS Crime_Total,
16            RANK() OVER (PARTITION BY YEAR(`DATE OCC`) ORDER BY COUNT(*) DESC) AS Rank    FROM crime_data
17     GROUP BY YEAR(`DATE OCC`), MONTH(`DATE OCC`)
18 )
19 WHERE Rank <= 3
20 ORDER BY Year, Crime_Total DESC
21 """
22 # Execute the SQL query
23 query1_result = spark.sql(query1_sql)
24
25 # Show the result
26 query1_result.show(query1_result.count(), truncate=False)
27
28
29 # End the timer and print execution time
30 end_time = time.time()
31 print(f"Execution time with 4 executors: {end_time - start_time} seconds")
32 spark.stop()
```

Εικόνα 6

Για την υλοποίηση αυτή μετατρέψαμε το DataFrame μας σε ένα SQL temporary view και πραγματοποιήσαμε τον κώδικα με SQL ακολουθώντας την ίδια λογική. Ο χρονομετρητής μας εδώ δεν λαμβάνει υπόψιν τον χρόνο μετατροπής του df σε Temporary View (παρόλο που η μετατροπή είναι αρκετά γρήγορη) καθώς θέλουμε να συγκρίνουμε τους χρόνους εκτέλεσης των Dataframe και SQL.

Έξοδος με SQL API

Year	Month	Crime_Total	Rank
2010	1	15750	1
2010	3	15653	2
2010	4	15516	3
2011	1	18269	1
2011	7	18257	2
2011	8	18041	3
2012	1	22983	1
2012	8	22832	2
2012	5	22525	3
2013	8	9112	1
2013	7	8956	2
2013	1	8682	3
2014	7	12106	1
2014	8	11990	2
2014	12	11969	3
2015	8	13857	1
2015	10	13502	2
2015	9	13365	3
2016	8	14596	1
2016	7	14479	2
2016	12	14323	3
2017	10	33585	1
2017	7	33357	2
2017	8	33297	3
2018	11	15613	1
2018	10	15341	2
2018	12	15198	3
2019	7	19122	1
2019	8	18979	2
2019	3	18856	3
2020	1	6090	1
2020	2	5869	2
2020	5	5530	3
2021	7	34744	1
2021	8	33653	2
2021	10	33639	3
2022	7	16748	1
2022	8	16526	2
2022	5	15833	3
2023	7	37594	1
2023	8	37500	2
2023	10	37033	3

Execution time with 4 executors: 16.277575254440308 seconds

Εικόνα 7

Μέσω του Spark UI βλέπουμε χρόνο εκτέλεσης

Spark Jobs [\(?\)](#)

User: user

Total Uptime: 1.0 min

Scheduling Mode: FIFO

Completed Jobs: 11

Εικόνα 8

Συμπεράσματα

Η χρήση του βελτιστοποιητή Catalyst και η δημιουργία κώδικα από το Spark SQL μπορεί να προσφέρει σημαντικές βελτιώσεις στην απόδοση σε σχέση με το API DataFrame, ιδίως για πολύπλοκα SQL ερωτήματα. Ωστόσο, το API DataFrame μπορεί να είναι πιο ευέλικτο και ευκολότερο στη χρήση για ορισμένες περιπτώσεις χρήσης, ιδίως για εκείνες που αφορούν απλούς μετασχηματισμούς δεδομένων ή επεξεργασία δομημένων πηγών δεδομένων. Γενικά όμως δεν υπάρχει σημαντική διαφορά όσον αφορά τις επιδόσεις, επειδή τόσο το DataFrame API όσο και το Spark SQL API είναι abstractions πάνω στο RDD (Resilient Distributed Dataset). Όπως παρατηρούμε από τα αποτελέσματα μας οι χρόνοι των δύο συστημάτων μας απέχουν ελάχιστα με το Dataframe API να έχει ελάχιστα καλύτερες επιδόσεις.

Ερώτημα 4

Για το ερώτημα 4, η εργασία περιελάμβανε την ταξινόμηση των εγγραφών εγκλημάτων σε δρόμους (STREET) με βάση την ώρα της ημέρας χρησιμοποιώντας τόσο το DataFrame/SQL όσο και το RDD API στο Apache Spark. Τα χρονικά τμήματα που εξετάστηκαν για την ταξινόμηση κατηγοριοποιήθηκαν ως εξής:

- Πρωί: 5.00πμ – 11.59πμ
- Απόγευμα: 12.00μμ – 4.59μμ
- Βράδυ: 5.00μμ – 8.59μμ
- Νύχτα: 9.00μμ – 4.59πμ

To Query 2 με χρήση του DataFrame (αρχείο question4_DF.py)

```
● ● ●
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import udf, count, desc, col
3 from pyspark.sql.types import StringType
4 from pythonScripts import dataframe_setup
5 import time
6 # Create a Spark session
7 df, spark = dataframe_setup.session_setup(4, "Query2_Df_implementation")
8
9 start_time = time.time()
10
11 def query2Group(_temp):
12     if 500 <= _temp <= 1159:
13         return 'Πρωί'
14     elif 1200 <= _temp <= 1659:
15         return 'Απόγευμα'
16     elif 1700 <= _temp <= 2059:
17         return 'Βράδυ'
18     elif 2100 <= _temp or _temp <= 459:
19         return 'Νύχτα'
20     else:
21         return 'Wrong time stamp'
22
23 # Register the function as a UDF
24 query2Group_udf = udf(query2Group, StringType())
25
26 street_df = df.filter(col("Premis Desc") == "STREET") \
27     .withColumn("TIME_OCC_GROUP", query2Group_udf("TIME OCC"))
28
29 query2 = (
30     street_df
31     .groupBy("TIME_OCC_GROUP")
32     .agg(count("*").alias("count"))
33     .orderBy(desc("count"))
34 )
35
36 # Show the result
37 query2.show()
38
39 end_time = time.time()
40 print(f"Execution time with 4 executors: {end_time - start_time} seconds")
41
42 spark.stop()
```

Για την υλοποίηση αυτού του Query δημιουργήσαμε την συνάρτηση query2Group η οποία χωρίζει τις ώρες της ημέρας σε 4 διαστήματα σύμφωνα με την εκφώνηση και τα δεδομένα που έχει μέσα το Dataframe μας. Στην συνέχεια φιλτράρουμε το Dataframe στο column Premis Desc, κρατάμε μόνο όσα έγιναν στο STREET και προσθέτουμε σε αυτά ένα νέο column, το TIME_OCC_GROUP που καλεί την συνάρτηση query2Group ώστε κάθε έγκλημα να αντιστοιχιστεί σε μία κατηγορία. Μετά κάνουμε GroupBy TIME_OCC_GROUP μετρώντας τον αριθμό των εγκλημάτων που έγιναν σε κάθε κατηγορία και OrderBy τον αριθμό των εγκλημάτων σε φθίνουσα σειρά.

Έξοδος DataFrame

TIME_OCC_GROUP	count
Νυχτα	240393
Βράδυ	188522
Απόγευμα	149713
Πρωί	125500

Execution time with 4 executors: 15.128343343734741 seconds

Εικόνα 10

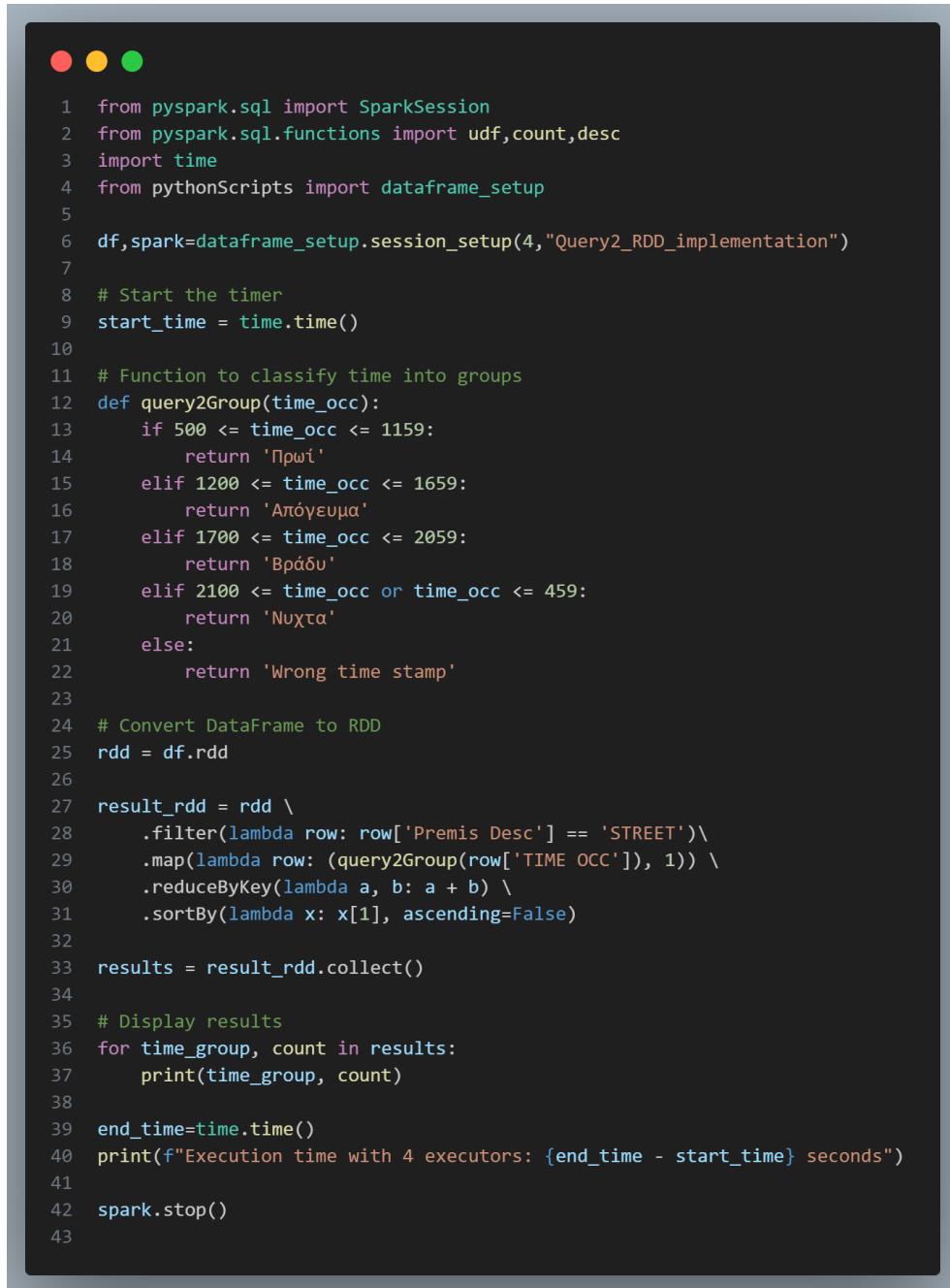
Μέσω του Spark UI βλέπουμε χρόνο εκτέλεσης

Spark Jobs [\(?\)](#)

User: user
Total Uptime: 54 s
Scheduling Mode: FIFO
Completed Jobs: 9

Εικόνα 11

To Query 2 με χρήση του RDD (αρχείο question4_RDD.py)

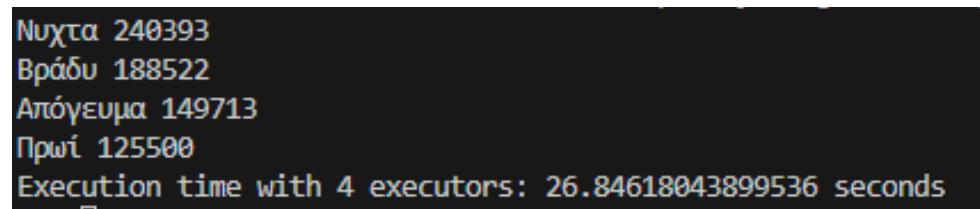


```
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import udf, count, desc
3 import time
4 from pythonScripts import dataframe_setup
5
6 df, spark = dataframe_setup.session_setup(4, "Query2_RDD_implementation")
7
8 # Start the timer
9 start_time = time.time()
10
11 # Function to classify time into groups
12 def query2Group(time_occ):
13     if 500 <= time_occ <= 1159:
14         return 'Πρωί'
15     elif 1200 <= time_occ <= 1659:
16         return 'Απόγευμα'
17     elif 1700 <= time_occ <= 2059:
18         return 'Βράδυ'
19     elif 2100 <= time_occ or time_occ <= 459:
20         return 'Νυχτα'
21     else:
22         return 'Wrong time stamp'
23
24 # Convert DataFrame to RDD
25 rdd = df.rdd
26
27 result_rdd = rdd \
28     .filter(lambda row: row['Premis Desc'] == 'STREET') \
29     .map(lambda row: (query2Group(row['TIME OCC']), 1)) \
30     .reduceByKey(lambda a, b: a + b) \
31     .sortBy(lambda x: x[1], ascending=False)
32
33 results = result_rdd.collect()
34
35 # Display results
36 for time_group, count in results:
37     print(time_group, count)
38
39 end_time = time.time()
40 print(f"Execution time with 4 executors: {end_time - start_time} seconds")
41
42 spark.stop()
43
```

Εικόνα 12

Για αυτό το μέρος του ερωτήματος μετατρέψαμε το Dataframe μας σε RDD και ακολουθήσαμε την ίδια λογική με πριν για την υλοποίηση του κώδικά μας.

Έξοδος API RDD



```
Νυχτα 240393
Βράδυ 188522
Απόγευμα 149713
Πρωί 125500
Execution time with 4 executors: 26.84618043899536 seconds
```

Εικόνα 13

Μέσω του Spark UI βλέπουμε χρόνο εκτέλεσης

Spark Jobs [\(?\)](#)

User: user

Total Uptime: 1.2 min

Scheduling Mode: FIFO

Completed Jobs: 7

Εικόνα 14

Συμπεράσματα

To DataFrame/SQL API υπερέβη σημαντικά το RDD API όσον αφορά την απόδοση του συστήματος για το ερώτημα. Η υλοποίηση DataFrame/SQL χρειάστηκε περίπου 15 δευτερόλεπτα, ενώ η υλοποίηση RDD περίπου 27 δευτερόλεπτα. Αυτή η σημαντική διαφορά μπορεί να αποδοθεί στις υψηλού επιπέδου αφαιρέσεις και βελτιστοποιήσεις που παρέχει το API DataFrame, επιτρέποντας την αποδοτικότερη εκτέλεση ερωτημάτων. Τα RDDs είναι μια θεμελιώδης δομή δεδομένων στο Apache Spark. Πρόκειται για αμετάβλητες, κατανεμημένες συλλογές αντικειμένων, οι οποίες είναι ανθεκτικές σε σφάλματα και μπορούν να λειτουργούν παράλληλα. Τα RDDs δεν έχουν από μόνα τους optimization engine, και αυτό πρέπει να γραφτεί από τον προγραμματιστή. Αντίθετα, τα Dataframes έχουν έναν catalyst optimizer από μόνα τους, οπότε στην περίπτωση μας είναι λογικό η απόδοση των Dataframes είναι αρκετά καλύτερη από αυτήν των RDDs.

Ερώτημα 5

To Query 3 με χρήση του DataFrame (αρχείο question5.py)

```
● ● ●
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import to_timestamp, year, col, desc, broadcast
3 import time
4 from pythonScripts import dataframe_setup
5
6 file_path1='hdfs:///user/data/LA_income_2015.csv'
7 file_path2='hdfs:///user/data/revgeocoding.csv'
8
9
10 for i in range(2,5):
11     df,spark=dataframe_setup.session_setup(i,"Query_3_" +str(i)+ "_executors_implementation")
12
13     df1 = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load(file_path1)
14     df2 = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load(file_path2)
15
16     start_time=time.time()
17
18     df2 = df2.withColumn('ZIPcode', col('ZIPcode').substr(1, 5))
19
20     # Filter year 2015
21     crime_data_2015 = df.filter(year("DATE OCC") == 2015) \
22         .filter(col("Vict Descent").isNotNull())
23
24     crime_with_zip = crime_data_2015.join(df2, (crime_data_2015["LAT"] == df2["LAT"]) & (crime_data_2015["LON"] == df2["LON"]))
25
26     crime_with_income = crime_with_zip.join(df1.withColumnRenamed("Zip Code", "ZIPcode"), "ZIPcode")
27
28     # Identify Top and Bottom 3 ZIP Codes by Income
29     top_zip_codes = df1.orderBy(desc("Estimated Median Income")).select("Zip Code").limit(3)
30     bottom_zip_codes = df1.orderBy("Estimated Median Income").select("Zip Code").limit(3)
31     selected_zip_codes = top_zip_codes.union(bottom_zip_codes).distinct()
32
33     # Convert selected_zip_codes DataFrame to a list of ZIP code values
34     zip_code_list = [row['Zip Code'] for row in selected_zip_codes.collect()]
35
36     # Analyze Crime Data by Descent
37     selected_crimes = crime_with_income.filter(col("ZIPcode").isin(zip_code_list))
38     query3 = selected_crimes.groupBy("Vict Descent").count().orderBy(desc("count"))
39
40     query3.show()
41     # End the timer and print execution time
42     end_time = time.time()
43
44     print("Execution time with ",str(i)," executors:", end_time - start_time, "seconds")
45
46     spark.stop()
```

Ευκόνα 15

Στο ερώτημα αυτό πραγματοποιούμε ένα for loop στο οποίο κάνουμε και τις 3 υλοποιήσεις, με 2,3 και 4 executors. Από το DF μας κρατάμε μόνο το 2015 και βγάζουμε εκτός τα Null Victim Descent. Μετά κάνουμε Join το DF με το revgeocoding. Στη συνέχεια βρίσκουμε τα top 3 και bottom 3 Estimated Median Incomes και επιστρέφουμε την περιοχή. Τέλος κάνουμε GroupBy Victim Descent και OrderBy των αριθμό εγκλημάτων σε φθίνουσα σειρά.

Έξοδος με 2 executors

Vict	Descent	count
W		623
O		161
H		154
X		102
B		62
A		27
F		2
J		1

```
Execution time with 2 executors: 14.188245058059692 seconds
```

Εικόνα 16

Έξοδος με 3 executors

Vict	Descent	count
W		623
O		161
H		154
X		102
B		62
A		27
F		2
J		1

```
Execution time with 3 executors: 11.633086681365967 seconds
```

Εικόνα 17

Έξοδος με 4 executors

Vict	Descent	count
W		623
O		161
H		154
X		102
B		62
A		27
F		2
J		1

```
Execution time with 4 executors: 11.69702935218811 seconds
```

Εικόνα 18

Μέσω του Spark UI βλέπουμε και χρόνο εκτέλεσης

3.5.0	application_1704962660509_0214	Query_3_4_executors_implementation	2024-01-12 21:52:17	2024-01-12 21:53:09	52 s
3.5.0	application_1704962660509_0213	Query_3_3_executors_implementation	2024-01-12 21:51:24	2024-01-12 21:52:16	52 s
3.5.0	application_1704962660509_0212	Query_3_2_executors_implementation	2024-01-12 21:50:28	2024-01-12 21:51:24	55 s

Εικόνα 19

Συμπεράσματα

Εύκολα παρατηρούμε πως η υλοποίηση με 2 executors είναι η πιο αργή. Όσο αναφορά τα 3 και 4 executors τα αποτελέσματα δεν είναι ξεκάθαρα καθώς μετά από αρκετές δοκιμές φαίνεται κάποιες φορές η υλοποίηση με 3 executors να είναι πιο γρήγορη, ενώ άλλες φορές το αντίθετο. Αυτό μας οδηγεί στο συμπέρασμα ότι η διαφορά μεταξύ 3 και 4 executors δεν προσφέρει reliable improvements. Αυτό μπορεί να συμβαίνει για διάφορους λόγους, όπως ότι στο συγκεκριμένο query το parallelization πάνω από 3 executors δεν έχει σημαντική χρήση γιατί πρέπει να γίνουν πολλά πράγματα sequentially ή ίσως αυτό το εξτρά parallelization να δημιουργεί χρονικές καθυστερήσεις μέσω της επιπλέον επικοινωνίας που δημιουργήσαμε στο σύστημα μας.

Επομένως καταλήγουμε πως οι 3 executors θα ήταν η καλύτερη λύση στο query μας καθώς είναι σαφώς αποτελεσματικότεροι από τους 2, και η προσθήκη 4ου executor δεν βελτιώνει την επίδοση του συστήματος μας.

Παρατηρώντας πιο αναλυτικά το Spark UI βλέπουμε πως στην εκτέλεση των 4 executors αρκετός χρόνος χάνεται στο initiation των executors ενώ η εκτέλεση των υπόλοιπων διαδικασιών πραγματοποιείται πιο γρήγορα από την αντίστοιχη εκτέλεση των 3 executors. Αυτός είναι ένας παράγοντας που οι τελικοί χρόνοι εκτέλεσης είναι τόσο κοντά.

Ερώτημα 6

To Query 4α με το αστυνομικό τμήμα που ανέλαβε την έρευνα για το περιστατικό (αρχείο question6_1a.py)

```
1 def haversine_distance(lat1, lon1, lat2, lon2):
2     # Radius of the Earth in kilometers
3     R = 6371.0
4     # Convert latitude and longitude from degrees to radians
5     lat1, lon1, lat2, lon2 = map(radians, [lat1, lon1, lat2, lon2])
6     # Calculate differences in coordinates
7     dlat = lat2 - lat1
8     dlon = lon2 - lon1
9     # Haversine formula
10    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
11    c = 2 * atan2(sqrt(a), sqrt(1 - a))
12    distance = R * c
13    return distance
14
15 df,spark=spark.read.format("csv")\ 
16 .option("header", "true").option("inferSchema", "true").load(file_path1)
17 revgecoding = spark.read.format("csv")\
18 .option("header", "true").option("inferSchema", "true").load(file_path2)
19
20 null_island_rows = df.filter((col("LAT") != 0.0) & (col("LON") != 0.0))
21
22 # Remove non gun crimes
23 df_upper = null_island_rows.where(
24     (null_island_rows["Weapon Used Cd"]>=100) &
25     (null_island_rows["Weapon Used Cd"]<200))
26
27 # Connect stations to crimes X= LON and Y= LAT
28 crimes_connected_stations=df_upper.join(police_stations, police_stations["PREC"] == df_upper["AREA "])
29
30 # Register the function as a UDF
31 custom_udf = udf(haversine_distance, FloatType())
32 # Register the function as a UDF
33 custom_udf = udf(haversine_distance, FloatType())
34
35 # Add a new column to the DataFrame using the UDF
36 crimes_connected_stations_with_distance = crimes_connected_stations\
37     .withColumn("Distance from crime", custom_udf("LAT", "LON", "Y","X"))
38
39 query4_1a = crimes_connected_stations_with_distance.groupBy(year("DATE OCC").alias("Year")) \
40     .agg(
41         format_number(avg(col("Distance from crime")), 4).alias("average_distance (km)"),
42         count(col("Distance from crime")).alias("crime_count")
43     )\
44     .orderBy("Year")
45 query4_1a.show()
46
47 end_time=time.time()
48 print(f"Execution time with 4 executors: {end_time - start_time} seconds")
49 spark.stop()
```

Ευκόνα 20

Ο κώδικας ξεκινά με την ανάγνωση δεδομένων από δύο αρχεία CSV: LAPD και δεδομένα αντίστροφης γεωκωδικοποίησης. Ακολουθεί ένα βήμα καθαρισμού των δεδομένων, όπου οι γραμμές με γεωγραφικό πλάτος και γεωγραφικό μήκος και οι δύο ίσες με μηδέν, φιλτράρονται από το κύριο σύνολο δεδομένων. Στη συνέχεια, αποκλείονται τα εγκλήματα χωρίς όπλο με φιλτράρισμα των γραμμών όπου ο κωδικός του όπλου που χρησιμοποιήθηκε βρίσκεται μεταξύ 100 και 199. Στη συνέχεια, ο κώδικας εκτελεί ένα join operation, συνδέοντας τα δεδομένα εγκλημάτων με τις πληροφορίες του αστυνομικού τμήματος. Συγκεκριμένα, ενώνει το DataFrame που περιέχει τα εγκλήματα με όπλα (df_upper) με το DataFrame των αστυνομικών σταθμών (police_stations) με βάση τις στήλες "PREC" και "AREA". Για τον υπολογισμό της απόστασης μεταξύ των τοποθεσιών εγκλημάτων και των αντίστοιχων αστυνομικών σταθμών, ορίζεται μια συνάρτηση απόστασης Haversine. Η συνάρτηση αυτή καταχωρίζεται ως συνάρτηση ορισμένη από τον χρήστη (UDF) και εφαρμόζεται για τη δημιουργία μιας νέας στήλης, "Απόσταση από το έγκλημα", στο DataFrame. Η ανάλυση συνεχίζεται με την ομαδοποίηση του DataFrame με βάση το έτος τέλεσης του εγκλήματος. Στη συνέχεια χρησιμοποιούνται συναρτήσεις συνάθροισης για τον υπολογισμό της μέσης απόστασης και την καταμέτρηση του αριθμού των εγκλημάτων για κάθε έτος. Τα τελικά αποτελέσματα ταξινομούνται ανά έτος. Τέλος, εμφανίζονται τα αποτελέσματα της ανάλυσης, παρουσιάζοντας τη μέση απόσταση και τον αριθμό των εγκλημάτων για κάθε έτος.

Έξοδος

Year	average_distance (km)	crime_count
2010	2.7451	6762
2011	2.7193	7797
2012	2.9096	8600
2013	2.6862	2981
2014	2.7319	3411
2015	2.6262	4557
2016	2.6808	5389
2017	2.7188	13592
2018	2.6765	5776
2019	2.7399	7129
2020	2.4122	2492
2021	2.6631	18240
2022	2.4725	7660
2023	2.6677	17572

Execution time with 4 executors: 10.39709186553955 seconds

Εικόνα 21

Μέσω του Spark UI βλέπουμε και χρόνο εκτέλεσης

Spark Jobs (?)

User: user
Total Uptime: 1.1 min
Scheduling Mode: FIFO
Completed Jobs: 11

Εικόνα 22

To Query 4α από το πλησιέστερο σε αυτά αστυνομικό τμήμα (αρχείο question6_2a.py)

```
● ● ●
1 def haversine_distance(lat1, lon1, lat2, lon2):
2     # Radius of the Earth in kilometers
3     R = 6371.0
4     # Convert latitude and longitude from degrees to radians
5     lat1, lon1, lat2, lon2 = map(radians, [lat1, lon1, lat2, lon2])
6     # Calculate differences in coordinates
7     dlat = lat2 - lat1
8     dlon = lon2 - lon1
9     # Haversine formula
10    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
11    c = 2 * atan2(sqrt(a), sqrt(1 - a))
12    distance = R * c
13    return distance
14
15 def smallest_haversine_distance(lat1, lon1):
16     smallest = float('inf')
17     # Access the broadcasted stations list
18     for station in broadcast_stations.value:
19         lat2, lon2 = station
20         _temp = haversine_distance(lat1, lon1, lat2, lon2)
21         if _temp < smallest:
22             smallest = _temp
23     return smallest
24
25 df,spark=datasframe_setup.session_setup(4,"Query4_2a_implementation")
26
27 start_time=time.time()
28
29 null_island_rows = df.filter((col("LAT") != 0.0) & (col("LON") != 0.0))
30
31 file_path1='hdfs:///user/data/LAPD_Police_Stations.csv'
32 file_path2='hdfs:///user/data/revgecoding.csv'
33
34 # Read the data into DataFrames with inferred schema
35 police_stations = spark.read.format("csv")\
36     .option("header", "true").option("inferSchema", "true").load(file_path1)
37 revgecoding = spark.read.format("csv")\
38     .option("header", "true").option("inferSchema", "true").load(file_path2)
39
40 # Convert the police stations DataFrame to a list of tuples
41 stations_list = [(row['Y'], row['X']) for row in police_stations.collect()]
42
43 broadcast_stations = spark.sparkContext.broadcast(stations_list)
44
45 df_upper = null_island_rows\
46     .where((null_island_rows["Weapon Used Cd"] >= 100) & (null_island_rows["Weapon Used Cd"] < 200))
47 custom_udf = udf(smallest_haversine_distance, FloatType())
48
49 # Add a new column to the DataFrame using the UDF
50 crimes_connected_closest_station = df_upper.withColumn("Distance from crime", custom_udf("LAT", "LON"))
51
52 query4_2a = crimes_connected_closest_station.groupBy(year("DATE OCC").alias("Year")) \
53     .agg(
54         format_number(avg(col("Distance from crime")), 4).alias("average_distance (km)"),
55         count(col("Distance from crime")).alias("crime_count")
56     )\
57     .orderBy("Year")
58
59 query4_2a.show()
60
61 end_time=time.time()
62 print(f"Execution time with 4 executors: {end_time - start_time} seconds")
63
64 broadcast_stations.unpersist()
65 spark.stop()
```

Για την υλοποίηση αυτού του Query κατασκευάσαμε την συνάρτηση smallest_haversine_distance, η οποία κάνει iterate τα broadcasted stations και βρίσκει την το station με την μικρότερη απόσταση από τα δοσμένα στην συνάρτηση lat,lon. Αρχικά φιλτράρουμε έξω όσες περιοχές είναι στο null island, δηλαδή έχουν lat==0 & lon==0. Στη συνέχεια φτιάχνουμε το stations_list που είναι μια λίστα με tuples που περιέχει τα coordinates κάθε station και την κάνουμε broadcast. Επιλέξαμε να κάνουμε broadcast αυτή την λίστα καθώς χωρίς αυτό όταν κάναμε execution του query είχαμε προβλήματα στο worker node, το οποίο έλεγε πως δεν έχει τα station διαθέσιμα. Έπειτα, κρατάμε μόνο τα εγκλήματα που έγιναν με πυροβόλο όπλο και κατασκευάζουμε ένα νέο udf με την συνάρτηση smallest_haversine_distance. Με αυτό το udf βάζουμε μια νέα στήλη πλέον στο dataframme μας που περιέχει για κάθε έγκλημα το κοντινότερο police station. Τέλος κάνουμε GroupBy το year, βρίσκουμε το avg distance για κάθε χρόνο των εγκλημάτων από το πιο κοντινό police station, και κάνουμε OrderBy year σε αύξουσα σειρά.

Έξοδος

Year	average_distance (km)	crime_count
2010	2.3861	6762
2011	2.4182	7797
2012	2.5443	8600
2013	2.4393	2981
2014	2.4301	3411
2015	2.4118	4557
2016	2.4625	5389
2017	2.3609	13592
2018	2.3913	5776
2019	2.4302	7129
2020	2.2036	2492
2021	2.3442	18240
2022	2.1689	7660
2023	2.3721	17572

Execution time with 4 executors: 10.980151891708374 seconds

Εικόνα 24

Μέσω του Spark UI βλέπουμε και χρόνο εκτέλεσης

Spark Jobs (?)

User: user

Total Uptime: 1.0 min

Scheduling Mode: FIFO

Completed Jobs: 11

Εικόνα 25

To Query 4β για τα εγκλήματα που ανατέθηκαν στο αστυνομικό τμήμα (αρχείο question6_1b.py)

```
● ● ●
1 def haversine_distance(lat1, lon1, lat2, lon2):
2     # Radius of the Earth in kilometers
3     R = 6371.0
4
5     # Convert latitude and longitude from degrees to radians
6     lat1, lon1, lat2, lon2 = map(radians, [lat1, lon1, lat2, lon2])
7
8     # Calculate differences in coordinates
9     dlat = lat2 - lat1
10    dlon = lon2 - lon1
11
12    # Haversine formula
13    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
14    c = 2 * atan2(sqrt(a), sqrt(1 - a))
15
16    distance = R * c
17
18    return distance
19
20 df,spark=dataframe_setup.session_setup(4,"Query4_1b_implementation")
21
22 start_time=time.time()
23
24 file_path1='hdfs:///user/data/LAPD_Police_Stations.csv'
25 file_path2='hdfs:///user/data/revgecoding.csv'
26
27 # Read the data into DataFrames with inferred schema
28 police_stations = spark.read.format("csv")\
29     .option("header", "true").option("inferSchema", "true").load(file_path1)
30 revgecoding = spark.read.format("csv")\
31     .option("header", "true").option("inferSchema", "true").load(file_path2)
32
33 null_island_rows = df.filter((col("LAT") != 0.0) & (col("LON") != 0.0))
34
35 null_island_rows = null_island_rows.where(
36     (null_island_rows["Weapon Used Cd"]>=100) &
37     (null_island_rows["Weapon Used Cd"]<200))
38
39 police_and_crimes=null_island_rows.join(police_stations, police_stations["PREC"] == null_island_rows["AREA"])
40
41 # Register the function as a UDF
42 custom_udf = udf(haversine_distance, FloatType())
43
44 # Add a new column to the DataFrame using the UDF
45 police_and_crimes_distance = police_and_crimes.withColumn("Distance from station", custom_udf("LAT", "LON", "Y","X"))
46
47 query4_1b = police_and_crimes_distance.groupBy("DIVISION") \
48     .agg(
49         format_number(avg(col("Distance from station")), 4).alias("average_distance (km)"),
50         count(col("Distance from station")).alias("crime_count")
51     )\
52     .orderBy(desc("crime_count"))
53
54 query4_1b.show(query4_1b.count(), truncate=False)
55
56 end_time=time.time()
57 print(f"Execution time with 4 executors: {end_time - start_time} seconds")
58 spark.stop()
```

Εικόνα 26

Ο κώδικας δημιουργεί ένα PySpark DataFrame (df) και ένα Spark Session καλώντας τη συνάρτηση **session_setup** από το αρχείο **dataframe_setup**. Οι σειρές με μηδενικές συντεταγμένες (γεωγραφικό πλάτος και γεωγραφικό μήκος ίσα με μηδέν) και εγκλήματα χωρίς την χρήση πυροβόλων όπλων φιλτράρονται από το κύριο σύνολο δεδομένων (df). Ένα νέο DataFrame, **police_and_crimes**, σχηματίζεται με την ένωση των

φιλτραρισμένων δεδομένων εγκληματικότητας με πληροφορίες αστυνομικών σταθμών με βάση τις στήλες "PREC" και "AREA" που ταιριάζουν. Η συνάρτηση Haversine distance καταχωρίζεται ως User Defined Function (UDF) και μια νέα στήλη, "Distance from station", προστίθεται στο DataFrame για την αποθήκευση των αποστάσεων μεταξύ των τοποθεσιών εγκλημάτων και των αντίστοιχων αστυνομικών τμημάτων. Στη συνέχεια, ομαδοποιείται το DataFrame με βάση το "DIVISION" και εφαρμόζονται συναρτήσεις συνάθροισης για τον υπολογισμό της μέσης απόστασης και του αριθμού των εγκλημάτων για κάθε αστυνομικό τμήμα. Τα αποτελέσματα ταξινομούνται κατά φθίνουσα σειρά με βάση τον αριθμό των εγκλημάτων. Τέλος, εμφανίζονται τα αποτελέσματα του ερωτήματος, τα οποία δείχνουν τη μέση απόσταση και τον αριθμό εγκλημάτων για κάθε αστυνομικό τμήμα.

Έξοδος

DIVISION	average_distance (km)	crime_count
77TH STREET	2.6381	17383
SOUTHEAST	2.1058	13691
NEWTON	2.0282	9884
SOUTHWEST	2.6981	8625
HOLLENBECK	2.6519	6100
HARBOR	4.0673	5798
RAMPART	1.5780	4990
NORTHEAST	3.8642	4320
OLYMPIC	1.8226	4251
HOLLYWOOD	1.4418	4158
MISSION	4.7176	4003
FOOTHILL	3.7742	3536
WILSHIRE	2.3982	3519
NORTH HOLLYWOOD	2.6781	3507
CENTRAL	1.1359	3469
WEST VALLEY	3.4397	3194
PACIFIC	3.7506	2677
VAN NUYS	2.2214	2670
DEVONSHIRE	3.9807	2557
TOPANGA	3.4464	2072
WEST LOS ANGELES	4.1971	1554

Execution time with 4 executors: 22.05766224861145 seconds

Εικόνα 27

Μέσω του Spark UI βλέπουμε και χρόνο εκτέλεσης

Spark Jobs (?)

User: user

Total Uptime: 1.5 min

Scheduling Mode: FIFO

Completed Jobs: 15

Εικόνα 28

To Query 4β για τα εγκλήματα με αστυνομικά τμήματα που ήταν πλησιέστερα σε αυτά (αρχείο question6_2b.py)

```
 1 def smallest_haversine_distance_return_code(lat1, lon1):
 2     smallest = float('inf')
 3     return_prec=0
 4     # Access the broadcasted stations list
 5     for station in broadcast_stations_code.value:
 6         lat2, lon2 ,prec_code = station
 7         _temp = haversine_distance(lat1, lon1, lat2, lon2)
 8         if _temp < smallest:
 9             return_prec=prec_code
10             smallest = _temp
11
12     return int(return_prec)
13
14 df,spark=dataframe_setup.session_setup(4,"Query4_2b_implementation")
15
16 start_time=time.time()
17
18 null_island_rows = df.filter((col("LAT") != 0.0) & (col("LON") != 0.0))
19
20 null_island_rows = null_island_rows.where(
21     (null_island_rows[ "Weapon Used Cd"]>=100) &
22     (null_island_rows[ "Weapon Used Cd"]<200))
23
24 file_path1='hdfs:///user/data/LAPD_Police_Stations.csv'
25 file_path2='hdfs:///user/data/revgecoding.csv'
26
27 # Read the data into DataFrames with inferred schema
28 police_stations = spark.read.format("csv")\
29     .option("header", "true").option("inferSchema", "true").load(file_path1)
30 revgecoding = spark.read.format("csv")\
31     .option("header", "true").option("inferSchema", "true").load(file_path2)
32
33 # Convert the police stations DataFrame to a list of tuples
34 stations_list = [(row['Y'], row['X'],row['PREC']) for row in police_stations.collect()]
35
36 # Broadcast the list
37 broadcast_stations_code = spark.sparkContext.broadcast(stations_list)
38
39 custom_udf = udf(smallest_haversine_distance_return_code, IntegerType())
40
41 # Add a new column to the DataFrame using the UDF
42 crimes_with_closest_prec = null_island_rows.withColumn("Prec code", custom_udf("LAT", "LON"))
43
44 custom_udf = udf(haversine_distance, FloatType())
45
46 crimes_with_closest_prec_distance=crimes_with_closest_prec\
47     .join(police_stations,crimes_with_closest_prec[ "Prec code"]==police_stations[ "PREC"])\\
48     .withColumn("Distance from prec",custom_udf("LAT","LON","Y","X"))
49
50 query4_2b = crimes_with_closest_prec_distance.groupBy("DIVISION") \
51     .agg(
52         format_number(avg(col("Distance from prec")), 4).alias("average_distance (km)"),
53         count(col("Distance from prec")).alias("crime_count")
54     )\
55     .orderBy(desc("crime_count"))
56
57 query4_2b.show(query4_2b.count(), truncate=False)
58
59 end_time=time.time()
60 print(f"Execution time with 4 executors: {end_time - start_time} seconds")
61
62 broadcast_stations_code.unpersist()
63 spark.stop()
```

Για την υλοποίηση αυτού του Query κατασκευάσαμε την συνάρτηση smallest_haversine_distance_return_code, που μοιάζει με την συνάρτηση που χρησιμοποιήσαμε στο 6_2α, με την διαφορά ότι τώρα επιστρέφουμε τον κωδικό του αστυνομικού τμήματος που έχει την μικρότερη απόσταση από το έγκλημα. Όπως και πριν, φιλτράρουμε έξω όσες περιοχές είναι στο null island και δεν χρησιμοποιήθηκαν πυροβόλα όπλα, και φτιάχνουμε το station_list πάλι με broadcast. Τώρα κατασκευάζουμε νέο udf με την συνάρτηση smallest_haversine_distance_return_code και βάζουμε μία καινούργια στήλη στο Dataframe μας που σε κάθε έγκλημα αντιστοιχίζει το prec_code του κοντινότερου police station. Αυτό το κάνουμε για να γίνεται αμέσως το join στο επόμενο βήμα μεταξύ του Dataframe και του police stations πάνω στο PREC. Τέλος, κάνουμε GroupBy Division, βρίσκουμε το average distance των εγκλημάτων που έγιναν πιο κοντά σε κάθε station, κάνουμε count τον αριθμό των εγκλημάτων, και OrderBy τον αριθμό των εγκλημάτων.

Έξοδος

DIVISION	average_distance (km)	crime_count
77TH STREET	1.7020	14449
SOUTHEAST	2.1973	12543
SOUTHWEST	2.2978	11176
NEWTON	1.5753	7255
WILSHIRE	2.4637	6263
HOLLENBECK	2.6459	6169
HOLLYWOOD	1.9703	6115
HARBOR	3.8788	5666
OLYMPIC	1.6591	5331
RAMPART	1.4227	4752
VAN NUYS	2.9574	4579
FOOTHILL	3.5570	4219
CENTRAL	1.0272	3707
NORTH HOLLYWOOD	2.7099	3526
NORTHEAST	3.6931	3438
WEST VALLEY	2.7767	3098
MISSION	3.7931	2556
PACIFIC	3.7209	2547
TOPANGA	3.0008	2176
DEVONSHIRE	3.0521	1350
WEST LOS ANGELES	2.6701	1043

Execution time with 4 executors: 21.52437663078308 seconds

Εικόνα 30

Μέσω του Spark UI βλέπουμε και χρόνο εκτέλεσης

Spark Jobs (?)

User: user

Total Uptime: 1.6 min

Scheduling Mode: FIFO

Completed Jobs: 16

Εικόνα 31

Ερώτημα 7

To Query 3 με χρήση του DataFrame (αρχείο question7_query3.py)

```
● ● ●

1  from pyspark.sql import SparkSession
2  from pyspark.sql.functions import to_timestamp, year, col, desc, broadcast
3  import time
4  from pythonScripts import dataframe_setup
5
6  file_path1='hdfs:///user/data/LA_income_2015.csv'
7  file_path2='hdfs:///user/data/revgeocoding.csv'
8
9  join_hints = ["broadcast", "MERGE", "SHUFFLE_HASH", "SHUFFLE_REPLICATE_NL"]
10
11 for i in range(2,5):
12     for hint in join_hints:
13         df,spark=dataframe_setup.session_setup(i,"Query3_"+str(i)+"_executors_"+hint)
14
15         df1 = spark.read.format("csv").option("header", "true")\
16             .option("inferSchema", "true").load(file_path1)
17         df2 = spark.read.format("csv").option("header", "true")\
18             .option("inferSchema", "true").load(file_path2)
19
20         start_time=time.time()
21
22         # Filter year 2015
23         crime_data_2015 = df.filter(year("DATE OCC") == 2015) \
24             .filter(col("Vict Descent").isNotNull())
25
26         print(f"\nApplying join hint {hint} for first join:")
27         # First Join
28         if hint == "broadcast":
29             crime_with_zip = crime_data_2015.join(broadcast(df2), ["LAT", "LON"])
30         else:
31             crime_with_zip = crime_data_2015.join(df2.hint(hint), ["LAT", "LON"])
32         crime_with_zip.explain()
33
34         print(f"\nApplying join hint {hint} for second join:")
35         # Second Join
36         if hint == "broadcast":
37             crime_with_income = crime_with_zip\
38                 .join(broadcast(df1.withColumnRenamed("Zip Code", "ZIPcode")), "ZIPcode")
39         else:
40             crime_with_income = crime_with_zip\
41                 .join(df1.withColumnRenamed("Zip Code", "ZIPcode").hint(hint), "ZIPcode")
42         crime_with_income.explain()
43
44         # Identify Top and Bottom 3 ZIP Codes by Income
45         top_zip_codes = df1.orderBy(desc("Estimated Median Income")).select("Zip Code").limit(3)
46         bottom_zip_codes = df1.orderBy("Estimated Median Income").select("Zip Code").limit(3)
47         selected_zip_codes = top_zip_codes.union(bottom_zip_codes).distinct()
48
49         # Convert selected_zip_codes DataFrame to a list of ZIP code values
50         zip_code_list = [row['Zip Code'] for row in selected_zip_codes.collect()]
51
52         # Analyze Crime Data by Descent
53         selected_crimes = crime_with_income.filter(col("ZIPcode").isin(zip_code_list))
54         query3 = selected_crimes.groupBy("Vict Descent").count().orderBy(desc("count"))
55
56         query3.show()
57         # End the timer and print execution time
58         end_time = time.time()
59
60         print("Execution time with ",str(i)," executors:", end_time - start_time, "seconds")
61
62         spark.stop()
```

Στο αρχείο αυτό καλούμε το query 3 για 3 διαφορετικούς αριθμούς executors (2,3,4) και για κάθε ένα από αυτά τα περάσματα τρέχουμε τον κώδικα 4 φορές για την κάθε διαφορετική τιμή του hint (broadcast , merge, shuffle hash, shuffle replicate nl) που θα πάρει το join των πινάκων μας. Θα δημιουργηθούνε δηλαδή $3*4=12$ Sessions στα οποία θα τρέχουμε διαφορετικούς συνδυασμούς των αριθμών executors και hints.

Μέσω του Spark UI βλέπουμε και χρόνο εκτέλεσης

3.5.0	application_1704962660509_0297	Query3_2_executors_SHUFFLE_REPLICATE_NL	2024-01-13 21:37:01	2024-01-13 21:37:58	57 s
3.5.0	application_1704962660509_0296	Query3_2_executors_SHUFFLE_HASH	2024-01-13 21:36:04	2024-01-13 21:37:01	57 s
3.5.0	application_1704962660509_0295	Query3_2_executors_MERGE	2024-01-13 21:35:11	2024-01-13 21:36:04	53 s
3.5.0	application_1704962660509_0294	Query3_2_executors_broadcast	2024-01-13 21:34:16	2024-01-13 21:35:10	55 s
3.5.0	application_1704962660509_0301	Query3_3_executors_SHUFFLE_REPLICATE_NL	2024-01-13 21:40:37	2024-01-13 21:41:31	54 s
3.5.0	application_1704962660509_0300	Query3_3_executors_SHUFFLE_HASH	2024-01-13 21:39:47	2024-01-13 21:40:37	50 s
3.5.0	application_1704962660509_0299	Query3_3_executors_MERGE	2024-01-13 21:38:52	2024-01-13 21:39:45	54 s
3.5.0	application_1704962660509_0298	Query3_3_executors_broadcast	2024-01-13 21:37:59	2024-01-13 21:38:51	52 s
3.5.0	application_1704962660509_0305	Query3_4_executors_SHUFFLE_REPLICATE_NL	2024-01-13 21:44:13	2024-01-13 21:45:09	57 s
3.5.0	application_1704962660509_0304	Query3_4_executors_SHUFFLE_HASH	2024-01-13 21:43:20	2024-01-13 21:44:12	52 s
3.5.0	application_1704962660509_0303	Query3_4_executors_MERGE	2024-01-13 21:42:26	2024-01-13 21:43:20	54 s
3.5.0	application_1704962660509_0302	Query3_4_executors_broadcast	2024-01-13 21:41:32	2024-01-13 21:42:25	53 s

To Query 4α με το αστυνομικό τμήμα που ανέλαβε την έρευνα για το περιστατικό (αρχείο question7_query4_1a.py)



```
1 join_hints = ["broadcast", "MERGE", "SHUFFLE_HASH", "SHUFFLE_REPLICATE_NL"]
2
3 for hint in join_hints:
4     df,spark=_dataframe_setup.session_setup(4,"Query4_1a_"+hint)
5
6     start_time=time.time()
7
8     #paths for test working in colab
9
10    file_path1='hdfs:///user/data/LAPD_Police_Stations.csv'
11    file_path2='hdfs:///user/data/revgecoding.csv'
12
13    # Read the data into DataFrames with inferred schema
14    police_stations = spark.read.format("csv")\
15        .option("header", "true").option("inferSchema", "true").load(file_path1)
16    revgecoding = spark.read.format("csv")\
17        .option("header", "true").option("inferSchema", "true").load(file_path2)
18
19    null_island_rows = df.filter((col("LAT") != 0.0) & (col("LON") != 0.0))
20
21    # Remove non gun crimes
22    df_upper = null_island_rows\
23        .where((null_island_rows["Weapon Used Cd"]>=100) & (null_island_rows["Weapon Used Cd"]<200))
24
25    # Register the function as a UDF
26    custom_udf = udf(haversine_distance, FloatType())
27
28    start_time = time.time()
29
30    # Apply join with different hints
31    if hint == "broadcast":
32        crimes_connected_stations = df_upper\
33            .join(broadcast(police_stations), police_stations["PREC"] == df_upper["AREA"])
34    else:
35        crimes_connected_stations = df_upper\
36            .join(police_stations.hint(hint), police_stations["PREC"] == df_upper["AREA"])
37
38    crimes_connected_stations.explain()
39
40    # Add a new column to the DataFrame using the UDF
41    crimes_connected_stations_with_distance = crimes_connected_stations\
42        .withColumn("Distance from crime", custom_udf("LAT", "LON", "Y","X"))
43
44    query4_1a = crimes_connected_stations_with_distance.groupBy(year("DATE OCC").alias("Year")) \
45        .agg(
46            format_number(avg(col("Distance from crime")), 4).alias("average_distance (km)"),
47            count(col("Distance from crime")).alias("crime_count")
48        )\
49        .orderBy("Year")
50    query4_1a.show()
51
52    end_time=time.time()
53    print(f"Execution time with {hint}: {end_time - start_time} seconds")
54    spark.stop()
```

Μέσω του Spark UI βλέπουμε και χρόνο εκτέλεσης

Version	App ID	App Name	Started	Completed	Duration	
3.5.0	application_1704962660509_0322	Query4_1a_SHUFFLE_REPLICATE_NL	2024-01-13 22:45:14	2024-01-13 22:46:08	54 s	
3.5.0	application_1704962660509_0321	Query4_1a_SHUFFLE_HASH	2024-01-13 22:44:21	2024-01-13 22:45:13	52 s	
3.5.0	application_1704962660509_0320	Query4_1a_MERGE	2024-01-13 22:43:29	2024-01-13 22:44:20	51 s	
3.5.0	application_1704962660509_0319	Query4_1a_broadcast	2024-01-13 22:42:36	2024-01-13 22:43:28	52 s	

Εικόνα 35

To Query 4α από το πλησιέστερο σε αυτά αστυνομικό τμήμα

Στο query αυτό, επιλέξαμε να μην χρησιμοποιήσουμε join operations μεταξύ DataFrames, επιλέγοντας μια εναλλακτική προσέγγιση για τη εύρεση των δεδομένων. Αντί να εκτελούμε παραδοσιακές ενώσεις, αξιοποιήσαμε το broadcast variable για την αποτελεσματική διανομή μικρότερων DataFrames σε κάθε εκτελεστή Spark, ελαχιστοποιώντας τη μετακίνηση δεδομένων σε όλο το cluster. Αξιοποιώντας την υπόδειξη broadcast, ενισχύσαμε την απόδοση του υπολογισμού, καθώς αποδείχθηκε πιο αποδοτική για τη συγκεκριμένη περίπτωση χρήσης. Αυτή η στρατηγική είναι ιδιαίτερα επωφελής όταν έχουμε να κάνουμε με μικρότερους πίνακες αναζήτησης που μπορούν να χωρέσουν στη μνήμη κάθε εκτελεστή. Αποφεύγοντας τα join operations, μετριάσαμε την πιθανή επιβάρυνση που σχετίζεται με την ανακατανομή μεγάλων ποσοτήτων δεδομένων σε όλο το cluster.

To Query 4β για τα εγκλήματα που ανατέθηκαν στο αστυνομικό τμήμα (αρχείο question7_query4_1b.py)

```
 1  file_path1='hdfs:///user/data/LAPD_Police_Stations.csv'
 2  file_path2='hdfs:///user/data/revgecoding.csv'
 3
 4  join_hints = ["broadcast", "MERGE", "SHUFFLE_HASH", "SHUFFLE_REPLICATE_NL"]
 5  for hint in join_hints:
 6
 7      df,spark=dataframe_setup.session_setup(4,"Query4_1b_"+hint)
 8
 9      # Read the data into DataFrames with inferred schema
10      police_stations = spark.read.format("csv")\
11          .option("header", "true").option("inferSchema", "true").load(file_path1)
12      revgecoding = spark.read.format("csv")\
13          .option("header", "true").option("inferSchema", "true").load(file_path2)
14
15      null_island_rows = df.filter((col("LAT") != 0.0) & (col("LONG") != 0.0))
16
17      null_island_rows = null_island_rows.where(
18          (null_island_rows["Weapon Used Cd"]>=100) &
19          (null_island_rows["Weapon Used Cd"]<200))
20      # Register the function as a UDF
21      custom_udf = udf(haversine_distance, FloatType())
22
23
24      start_time = time.time()
25      print(f"\nApplying join hint {hint}:")
26      # Apply join with different hints
27      if hint == "broadcast":
28          police_and_crimes = null_island_rows\
29              .join(broadcast(police_stations),
30                  police_stations["PREC"] == null_island_rows["AREA"])
31      else:
32          police_and_crimes = null_island_rows\
33              .join(police_stations.hint(hint),
34                  police_stations["PREC"] == null_island_rows["AREA"])
35
36      police_and_crimes.explain()
37
38      # Add a new column to the DataFrame using the UDF
39      police_and_crimes_distance = police_and_crimes\
40          .withColumn("Distance from station", custom_udf("LAT", "LONG", "Y", "X"))
41
42      query4_1b = police_and_crimes_distance.groupBy("DIVISION") \
43          .agg(
44              format_number(avg(col("Distance from station")), 4)\ \
45                  .alias("average_distance (km)"),
46              count(col("Distance from station")).alias("crime_count"))
47          ) \
48          .orderBy(desc("crime_count"))
49
50      query4_1b.show()
51
52      end_time=time.time()
53      print(f"Execution time with 4 executors: {end_time - start_time} seconds")
54      spark.stop()
```

Μέσω του Spark UI βλέπουμε και χρόνο εκτέλεσης

Version	App ID	App Name	Started	Completed	Duration
3.5.0	application_1704962660509_0509	Query4_1b_SHUFFLE_REPLICATE_NL	2024-01-16 11:02:12	2024-01-16 11:03:12	1.0 min
3.5.0	application_1704962660509_0508	Query4_1b_SHUFFLE_HASH	2024-01-16 11:01:14	2024-01-16 11:02:11	56 s
3.5.0	application_1704962660509_0507	Query4_1b_MERGE	2024-01-16 11:00:08	2024-01-16 11:01:13	1.1 min
3.5.0	application_1704962660509_0506	Query4_1b_broadcast	2024-01-16 10:59:03	2024-01-16 11:00:06	1.0 min

Εικόνα 37

To Query 4β για τα εγκλήματα με αστυνομικά τμήματα που ήταν πλησιέστερα σε αυτά (αρχείο question7_query4_2b.py)

```
● ● ●

1  join_hints = ["broadcast", "MERGE", "SHUFFLE_HASH", "SHUFFLE_REPLICATE_NL"]
2  for hint in join_hints:
3      df,spark=datasource_setup.session_setup(4,"Query4_2b_"+hint)
4
5      file_path1='hdfs:///user/data/LAPD_Police_Stations.csv'
6      file_path2='hdfs:///user/data/revgeocoding.csv'
7
8          # Read the data into DataFrames with inferred schema
9          police_stations = spark.read.format("csv")\
10              .option("header", "true").option("inferSchema", "true").load(file_path1)
11          revgeocoding = spark.read.format("csv")\
12              .option("header", "true").option("inferSchema", "true").load(file_path2)
13          start_time=time.time()
14
15          null_island_rows = df.filter((col("LAT") != 0.0) & (col("LON") != 0.0))
16
17          null_island_rows = null_island_rows.where(
18              (null_island_rows["Weapon Used Cd"]>100) &
19              (null_island_rows["Weapon Used Cd"]<200))
20
21          # Convert the police stations DataFrame to a list of tuples
22          stations_list = [(row['Y'], row['X'],row['PREC']) for row in police_stations.collect()]
23
24          # Broadcast the list
25          broadcast_stations_code = spark.sparkContext.broadcast(stations_list)
26
27          custom_udf = udf(smallest_haversine_distance_return_code, IntegerType())
28
29          # Add a new column to the DataFrame using the UDF
30          crimes_with_closest_prec = null_island_rows.withColumn("Prec code", custom_udf("LAT", "LON"))
31
32          custom_udf = udf(haversine_distance, FloatType())
33          print(f"Applying join hint {hint}")
34          if hint == "broadcast":
35              crimes_with_closest_prec_distance=crimes_with_closest_prec\
36                  .join(broadcast(police_stations),
37                      crimes_with_closest_prec["Prec code"]==police_stations["PREC"])
38          else:
39              crimes_with_closest_prec_distance = crimes_with_closest_prec\
40                  .join(police_stations.hint(hint),
41                      crimes_with_closest_prec["Prec code"] == police_stations["PREC"])
42          crimes_with_closest_prec_distance.explain()
43          crimes_with_closest_prec_distance = crimes_with_closest_prec_distance\
44              .withColumn("Distance from prec", custom_udf("LAT", "LON", "Y", "X"))
45          query4_2b = crimes_with_closest_prec_distance.groupBy("DIVISION") \
46              .agg(
47                  format_number(avg(col("Distance from prec")), 4).alias("average_distance (km)"),
48                  count(col("Distance from prec")).alias("crime_count")
49              )\
50              .orderBy(desc("crime_count"))
51
52          query4_2b.show(query4_2b.count(), truncate=False)
53
54          end_time=time.time()
55          print(f"Execution time with 4 executors: {end_time - start_time} seconds")
56
57          broadcast_stations_code.unpersist()
58          spark.stop()
```

Μέσω του Spark UI βλέπουμε και χρόνο εκτέλεσης

Version	App ID	App Name	Started	Completed	Duration
3.5.0	application_1704962660509_0496	Query4_2b_SHUFFLE_REPLICATE_NL	2024-01-16 10:39:31	2024-01-16 10:40:45	1.2 min
3.5.0	application_1704962660509_0495	Query4_2b_SHUFFLE_HASH	2024-01-16 10:38:07	2024-01-16 10:39:28	1.4 min
3.5.0	application_1704962660509_0494	Query4_2b_MERGE	2024-01-16 10:36:53	2024-01-16 10:38:06	1.2 min
3.5.0	application_1704962660509_0493	Query4_2b_broadcast	2024-01-16 10:35:37	2024-01-16 10:36:52	1.3 min

Εικόνα 39

Broadcast join

- Χρησιμοποιείται όταν ένα από τα σύνολα δεδομένων είναι αρκετά μικρό ώστε να χωράει στη μνήμη κάθε κόμβου εργασίας.
- Ιδανικό για την ένωση ενός μεγάλου συνόλου δεδομένων με ένα μικρό σύνολο δεδομένων.
- Μειώνει την ανάγκη για ανακάτεμα των δεδομένων στους κόμβους, οδηγώντας σε ταχύτερη εκτέλεση.

Merge join

- Αποτελεσματικό για μεγάλα σύνολα δεδομένων όπου και οι δύο πλευρές του join είναι πολύ μεγάλες για να χωρέσουν στη μνήμη.
- Απαιτεί ανακάτεμα των δεδομένων, αλλά είναι αποδοτικό όσον αφορά τη χρήση μνήμης.
- Καλή προεπιλεγμένη επιλογή για μεγάλες, ανακατεμένες ενώσεις σε σύνολα δεδομένων που δεν μπορούν να αξιοποιήσουν την broadcast join.

Shuffle Hash Join

- Χρήσιμο όταν και τα δύο σύνολα δεδομένων είναι μεγάλα αλλά αρκετά μικρά ώστε να δημιουργηθεί ένα hash table για τη μία πλευρά του join στη μνήμη.
- Μπορεί να είναι ταχύτερη από την merge join για ορισμένα μεγέθη δεδομένων (συνήθως για μεγέθη δεδομένων που κυμαίνονται από 10MB έως μερικά GB).
- Δεν χρησιμοποιείται ή δεν συνιστάται τόσο ευρέως όσο η sort merge join λόγω των περιορισμών μνήμης.

Shuffle Replicate NL Join

- Χρησιμοποιείται για καρτεσιανές συνδέσεις όπου κάθε γραμμή ενός συνόλου δεδομένων συνδέεται με κάθε γραμμή ενός άλλου συνόλου δεδομένων.
- Πολύ απαιτητική σε πόρους και συνήθως δεν συνιστάται, εκτός αν απαιτείται ρητά για καρτεσιανά προϊόντα.

Συμπεράσματα για query 3

Από τα αποτελέσματα του query 3 ([Εικόνα 33](#)) παρατηρούμε πως το shuffle replicate nl έχει την χειρότερη απόδοση όπως και αναμέναμε. Το merge join είχε καλή επίδοση για 2 executors αλλά καθώς ανέβαινε ο αριθμός των executors δεν βελτιωνόταν ο χρόνος εκτέλεσης του κώδικα. Το broadcast ήταν το πιο consistent από τα 4 παρουσιάζοντας καλές αποδόσεις για κάθε εκτέλεση. Τέλος το shuffle hash ενώ για 2 executors δεν είχε καλή απόδοση, στους 3 και 4 executors αποδείχθηκε να έχει την καλύτερη απόδοση.

Στην περίπτωσή μας θα διαλέγαμε το broadcast ως hint των join operations μας καθώς είναι η πιο consistent επιλογή για το μέγεθος των datasets που χρησιμοποιούμε.

Συμπεράσματα για query 4

Στα συμπεράσματα για το Query 4, παρατηρήθηκε ότι οι χρόνοι εκτέλεσης των ερωτημάτων ήταν συγκρίσιμοι, αλλά η στρατηγική join με τη χρήση του broadcast ήταν πιο αποτελεσματική και σταθερή σε σχέση με άλλες πιθανές μεθόδους. Παράλληλα, η αξιοποίηση της στρατηγικής join με merge ή shuffle hash join μπορεί να είναι πιο αποδοτική για μεγαλύτερα datasets. Ωστόσο, στην περίπτωση του Query 4, όπου τα σύνολα δεδομένων είναι σχετικά μικρά, οποιοσδήποτε τρόπος join και αν χρησιμοποιήσουμε θα παρέχει παρόμοια αποτελέσματα όσο αναφορά την επίδοση. Συνεπώς, η επιλογή του broadcast join συνάδει με τη θεωρητική βέλτιστη πρακτική για τέτοιου μεγέθους datasets, παρέχοντας ικανοποιητική εκτέλεση και αποτελεσματικότητα.