

Διαχείριση Δεδομένων Μεγάλης Κλίμακας

Εξαμηνιαία Εργασία

Αλέξιος Μηλιώνης, 03400226

Παναγιώτης Χρονόπουλος, 03400240

Επιστήμη Δεδομένων και Μηχανική Μάθηση

Περιεχόμενα

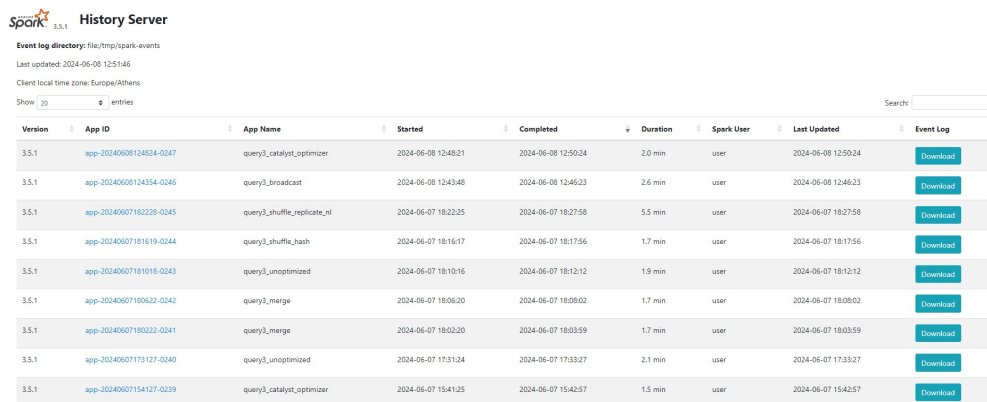
Ζητούμενο 1	2
Ζητούμενο 2	3
Ζητούμενο 3	4
Ζητούμενο 4	6
Ζητούμενο 5	7
Ζητούμενο 6	10
Ζητούμενο 7	12
Παράρτημα	13

Ζητούμενο 1

Ακολουθώντας τις δοθείσες οδηγίες εγκαταστήσαμε το Apache Spark και το Hadoop Distributed File System (HDFS). Το περιβάλλον είναι πλήρως κατανεμημένο με 2 κόμβους (Εικόνα 1) και οι web εφαρμογές των HDFS και Spark Job History Server είναι προσβάσιμες (Εικόνες 2 και 3).

```
user@master:~$ jps
1268 Worker
29190 HistoryServer
679 SecondaryNameNode
29016 Jps
32744 NameNode
1165 Master
431 DataNode
user@master:~$ |
```

Εικόνα 1: Το κατανεμημένο περιβάλλον



The screenshot shows the Spark Job History Server interface. At the top, it displays the Spark logo and version 3.5.1, followed by 'History Server'. Below this, it shows the event log directory as 'file:/tmp/spark-events', the last update time as '2024-06-08 12:51:46', and the client local time zone as 'Europe/Athens'. There is a search bar and a 'Show 20 entries' dropdown. The main part of the interface is a table with columns: Version, App ID, App Name, Started, Completed, Duration, Spark User, Last Updated, and Event Log. The table contains 9 rows of job history entries, each with a 'Download' button in the Event Log column.

Version	App ID	App Name	Started	Completed	Duration	Spark User	Last Updated	Event Log
3.5.1	app-20240608124024-0247	query3_catalyst_optimizer	2024-06-08 12:40:21	2024-06-08 12:50:24	2.0 min	user	2024-06-08 12:50:24	Download
3.5.1	app-20240608124354-0246	query3_broadcast	2024-06-08 12:43:48	2024-06-08 12:46:23	2.6 min	user	2024-06-08 12:46:23	Download
3.5.1	app-20240607182228-0245	query3_shuffle_replicate_nl	2024-06-07 18:22:25	2024-06-07 18:27:58	5.5 min	user	2024-06-07 18:27:58	Download
3.5.1	app-20240607181619-0244	query3_shuffle_hash	2024-06-07 18:16:17	2024-06-07 18:17:56	1.7 min	user	2024-06-07 18:17:56	Download
3.5.1	app-20240607181010-0243	query3_unoptimized	2024-06-07 18:10:16	2024-06-07 18:12:12	1.9 min	user	2024-06-07 18:12:12	Download
3.5.1	app-20240607180622-0242	query3_merge	2024-06-07 18:06:20	2024-06-07 18:08:02	1.7 min	user	2024-06-07 18:08:02	Download
3.5.1	app-20240607180222-0241	query3_merge	2024-06-07 18:02:20	2024-06-07 18:03:59	1.7 min	user	2024-06-07 18:03:59	Download
3.5.1	app-20240607173127-0240	query3_unoptimized	2024-06-07 17:31:24	2024-06-07 17:33:27	2.1 min	user	2024-06-07 17:33:27	Download
3.5.1	app-20240607154127-0239	query3_catalyst_optimizer	2024-06-07 15:41:25	2024-06-07 15:42:57	1.5 min	user	2024-06-07 15:42:57	Download

Εικόνα 2: Spark Job History Server

Hadoop

Overview

Datanodes

Datanode Volume Failures

Snapshot

Startup Progress

Utilities

Overview 'master:9000' (active)

Started:	Thu Mar 28 20:27:56 +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-a92ac512-b264-4d88-8507-bed4474671b5
Block Pool ID:	BP-333587580-192.168.0.1-1711650434962

Summary

Security is off.

Safemode is off.

21 files and directories, 26 blocks (26 replicated blocks, 0 erasure coded block groups) = 47 total filesystem object(s).

Heap Memory used 67.78 MB of 96.5 MB Heap Memory. Max Heap Memory is 878.5 MB.

Non Heap Memory used 78.75 MB of 80.71 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	58.8 GB
Configured Remote Capacity:	0 B
DFS Used:	1.7 GB (2.89%)
Non DFS Used:	11.68 GB
DFS Remaining:	42.4 GB (72.1%)
Block Pool Used:	1.7 GB (2.89%)
DataNodes usages% (Min/Median/Max/stdDev):	2.89% / 2.89% / 2.89% / 0.00%

Εικόνα 3: Hadoop UI

Ζητούμενο 2

Δημιουργήθηκε ένα directory στο HDFS όπου τοποθετήθηκαν τα σύνολα δεδομένων της εργασίας (Εικόνα 4). Χρησιμοποιήθηκαν οι εξής εντολές:

- **hadoop fs -mkdir /datasets:** δημιουργεί το φάκελο datasets στο HDFS
- **wget <url>:** Πραγματοποιεί λήψη του csv από το δοθέν url
- **hadoop fs -put <csv path>/datasets/<csv path>:** Τοποθετεί το δοθέν csv αρχείο στο φάκελο datasets στο HDFS

Υλοποιήθηκε επίσης το **convert_to_parquet.py** το οποίο μετατρέπει τα csv για το βασικό σύνολο δεδομένων, δηλαδή τις καταγραφές εγκλημάτων στο Los Angeles, σε αρχεία parquet.

Να σημειωθεί πως το βασικό σύνολο δεδομένων και συγκεκριμένα το αρχείο **Crime Data from 2020 to Present** ενημερώνεται σε τακτά χρονικά διαστήματα, επομένως τα αποτελέσματα των queries θα διαφέρουν ανάλογα με ποια έκδοση του αρχείου έχει κανείς.

```

user@master:~$ hadoop fs -ls /datasets
Found 6 items
-rw-r--r-- 2 user supergroup 12859 2024-05-25 18:22 /datasets/LA_income_2015.csv
-rw-r--r-- 2 user supergroup 537198636 2024-05-26 20:12 /datasets/crime_data.csv
-rw-r--r-- 2 user supergroup 241051722 2024-05-25 18:22 /datasets/crime_data2.csv
drwxr-xr-x ~ user supergroup 0 2024-05-26 20:14 /datasets/merged_crime_data
-rw-r--r-- 2 user supergroup 1380 2024-05-01 17:11 /datasets/police_stations.csv
-rw-r--r-- 2 user supergroup 897062 2024-05-25 18:22 /datasets/revgecoding.csv
user@master:~$

```

Εικόνα 4: Το σύστημα αρχείων

Ζητούμενο 3

Στον Πίνακα 1 παρατίθενται οι χρόνοι εκτέλεσης για την υλοποίηση του Query 1 με DataFrame και SQL APIs με εισόδους csv αρχεία και parquet αρχεία.

Αναμενόμενα το parquet format οδηγεί σε γρηγορότερους χρόνους εκτέλεσης, γεγονός που οφείλεται στο μικρότερο μέγεθος των parquet αρχείων εξαιτίας του built-in compression τους. Ακόμη, το parquet format αποθηκεύει τα δεδομένα κατά στήλη, το οποίο συμβάλλει σε γρηγορότερους χρόνους εκτέλεσης των χυεριες, καθώς αποφεύγει να διαβάσει στήλες με μη χρήσιμα, για το query, δεδομένα. Αντίθετα το csv format πρέπει να διαβάσει όλες τις στήλες της κάθε γραμμής, ακόμα και αν χρειάζεται μόνο ένα μικρό υποσύνολο αυτών.

Για την επιλογή Spark DataFrame ή sparkSQL, οι χρόνοι απόδοσης είναι παραπλήσιοι, το οποίο είναι και θεωρητικά αναμενόμενο, καθώς χρησιμοποιούν τις τεχνικές optimization που παρέχονται από τον catalyst optimizer του Spark, αλλά και το ίδιο physical execution engine.

Στον Πίνακα 2 παρουσιάζονται τα αποτελέσματα του Query 1.

	DataFrame	SQL
CSV	1 min	59 s
Parquet	35 s	36 s

Πίνακας 1: Οι χρόνοι εκτέλεσης του Query 1

year	month	crime_total	ranking
2010	01	19520	1
2010	03	18131	2
2010	07	17857	3
2011	01	18141	1
2011	07	17283	2
2011	10	17034	3
2012	01	17954	1
2012	08	17661	2
2012	05	17502	3
2013	08	17441	1
2013	01	16828	2
2013	07	16645	3
2014	10	17331	1
2014	07	17258	2
2014	12	17198	3
2015	10	19221	1
2015	08	19011	2
2015	07	18709	3
2016	10	19660	1
2016	08	19496	2
2016	07	19450	3
2017	10	20437	1
2017	07	20199	2
2017	01	19849	3
2018	05	19976	1
2018	07	19879	2
2018	08	19765	3
2019	07	19126	1
2019	08	18987	2
2019	03	18865	3
2020	01	18542	1
2020	02	17272	2
2020	05	17219	3
2021	10	19326	1
2021	07	18672	2
2021	08	18387	3
2022	05	20450	1
2022	10	20313	2
2022	06	20255	3
2023	10	20029	1
2023	08	20024	2
2023	01	19902	3
2024	01	18762	1
2024	02	17214	2
2024	03	16009	3

Πίνακας 2: Τα αποτελέσματα του Query 1

Ζητούμενο 4

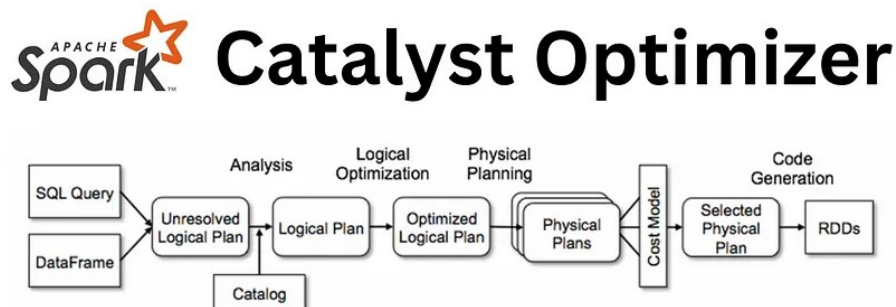
Υλοποιήθηκε το Query 2 με χρήση DataFrame και RDD APIs. Κατά την ανάγνωση των αρχείων και μετατροπή σε rdd objects παρατηρήθηκε πως ορισμένες γραμμές είχαν περισσότερα αντικείμενα από το πλήθος των στηλών και στο τέλος είχαμε διαφορετικά αποτελέσματα σε σχέση με το DataFrame. Πηγή του προβλήματος ήταν ορισμένα στοιχεία στηλών τα οποία περιείχαν το κόμμα (,) (Εικόνα 5). Για να λυθεί αξιοποιήσαμε τη βιβλιοθήκη csv για κατάλληλη ανάγνωση των αρχείων.

100	Central	111	2	745 VANDALISM - MISDEAMEANOR (\$399 OR UNDER)	0329 1402	31 F	O	122 VEH
101	Central	157	1	230 ASSAULT WITH DEADLY WEAPON, AGGRAVATED ASSAULT	416	24 M	H	710 OTH
102	Central	123	2	745 VANDALISM - MISDEAMEANOR (\$399 OR UNDER)	329	46 M	B	108 PARK
103	Central	161	1	510 VEHICLE - STOLEN		31 M	W	101 STRE
104	Central	153	1	230 ASSAULT WITH DEADLY WEAPON, AGGRAVATED ASSAULT	416	41 M	H	101 STRE
105	Central	135	1	210 ROBBERY	302	61 M	O	406 OTH
106	Central	133	1	230 ASSAULT WITH DEADLY WEAPON, AGGRAVATED ASSAULT	0416 0429	30 M	H	102 SIDE
107	Central	111	2	740 VANDALISM - FELONY (\$400 & OVER, ALL CHURCH VANDALISMS)	0329 1251	23 F	H	122 VEH
108	Central	139	2	930 CRIMINAL THREATS - NO WEAPON DISPLAYED	421	27 M	W	729 SPEC
109	Central	155	2	946 OTHER MISCELLANEOUS CRIME	914	0 M	W	101 STRE
110	Central	161	1	341 THEFT-GRAND (\$950.01 & OVER)EXCPT,GUNS,FOWL,LIVESTK,PROD 0344 1402		41 M	W	203 OTH

Εικόνα 5: Στοιχείο που περιλαμβάνει κόμμα

Τα αποτελέσματα της εκτέλεσης παρατίθενται στον Πίνακα 3.

Οι χρόνοι εκτέλεσης παρατίθενται στον Πίνακα 4. Οι χρόνοι εκτέλεσης είναι παραπλήσιοι, ωστόσο στο χρόνο υλοποίησης με DataFrame συμπεριλαμβάνεται χρόνος για τον καθορισμό των logical και physical plans από τον Catalyst Optimizer. Αυτό συνεπάγεται ότι συγκρίνοντας τους καθαρούς χρόνους εκτέλεσης το DataFrame αποτελεί την ταχύτερη λύση. Το γεγονός αυτό συμφωνεί με τη θεωρία η οποία αναφέρει ότι ο Catalyst Optimizer εντοπίζει το βέλτιστο τρόπο εκτέλεσης των μετασχηματισμών των δεδομένων, μετατρέπει το DataFrame σε lower level RDD, και εκτελεί τις πράξεις (Εικόνα 6).



Εικόνα 6: Λειτουργία του Catalyst Optimizer (πηγή)

Time of Day	Crime Total
Night	243374
Evening	191792
Afternoon	151564
Morning	126611

Πίνακας 3: Τα αποτελέσματα του Query 2

API	Χρόνος εκτέλεσης
DataFrame	36 s
RDD	37 s

Πίνακας 4: Οι χρόνοι εκτέλεσης του Query 2

Ζητούμενο 5

Στο κώδικα χρησιμοποιήθηκαν δύο joins. Το πρώτο συνέδεσε τις εγγραφές των εγκλημάτων με τους ταχυδρομικούς κώδικες των περιοχών όπου διαπράχθηκαν και το δεύτερο το αποτέλεσμα του πρώτου με τα ετήσια εισοδήματα ανά νοικοκυριό. Τα αποτελέσματα παρατίθενται στους Πίνακες 5 και 6.

Η στρατηγική που επιλέχθηκε από τον Catalyst Optimizer και στις δύο περιπτώσεις είναι το **Broadcast Join** (Εικόνες 7 και 8). Πρόκειται για τη βέλτιστη στρατηγική όταν ο ένας πίνακας είναι μικρότερος από ένα κατώφλι (default τιμή τα 10MB). Κατά το Broadcast ο μικρός πίνακας, ο οποίος χωράει ολόκληρος στη μνήμη κάθε κόμβου, αποθηκεύεται αρχικά στο master και έπειτα κάθε slave λαμβάνει ένα αντίγραφο για να εκτελέσει τοπικά το join, μειώνοντας την ανάγκη για μεταφορά δεδομένων.

Έπειτα, πειραματιστήκαμε με τις υπόλοιπες 3 υλοποιήσεις που προσφέρονται:

- **Shuffle Hash:** Τα δεδομένα αρχικά διαμοιράζονται σε partitions έτσι ώστε δεδομένα που περιλαμβάνουν το ίδιο join key να βρίσκονται στον ίδιο partition. Στη συνέχεια τα partitions διαμοιράζονται στους κόμβους. Σε κάθε κόμβο δημιουργείται ένα hash table για το μικρότερο πίνακα που παίζει το ρόλο του ευρετηρίου ώστε να πραγματοποιηθεί το join. Χρησιμοποιείται όταν και οι δύο πίνακες είναι μεγάλοι και δεν μπορούμε να χρησιμοποιήσουμε το Broadcast. Πρέπει τα partitions και τα hash tables να χωρούν στη μνήμη για να εκτελεστεί επιτυχώς.
- **Merge:** Τα δεδομένα πρώτα ταξινομούνται με βάση το join key (πρέπει να μπορεί να γίνει sort) και έπειτα διαδοχικά προσπελούνται και συνδέονται. Λόγω της ταξινόμησης δεν απαιτείται μία πλήρης προσπέλαση για κάθε εγγραφή. Μπορεί να χρησιμοποιηθεί και για join μεγάλων πινάκων. Εάν δεν υπάρχει αρκετή μνήμη, τα δεδομένα εγγράφονται στο δίσκο, εξασφαλίζοντας τη περάτωση της διαδικασίας σε αντίθεση με το Shuffle Hash.

- **Shuffle Replicate NL:** Τα δεδομένα του μικρού dataset αντιγράφονται σε κάθε worker ενώ του μεγάλου διαμοιράζονται με shuffle. Μετά, σε κάθε κόμβο εκτελείται ένα Nested-Loop Join ανάμεσα στο μικρό dataset και στο partition. Είναι κατάλληλη στρατηγική όταν έχουμε μια άνιση κατανομή των join keys (skewed data). Λόγω του Nested-Loop, είναι υπολογιστικά κοστοβόρα.

Η στρατηγική που επιλέχθηκε από τον Catalyst Optimizer είναι πράγματι η βέλτιστη στη περίπτωση μας (Πίνακας 7) καθώς σε κάθε περίπτωση ενώνουμε με πολύ μικρά σε μέγεθος σύνολα δεδομένων. Αυτό επιβεβαιώνεται και από τους χρόνους εκτέλεσης (Πίνακας 7).

victim descent	total victims
W	338
O	106
H	52
X	27
B	16
A	16

Πίνακας 5: Τα αποτελέσματα του Query 3 για τις top 3 περιοχές με βάση το εισόδημα

victim descent	total victims
H	1531
B	1092
W	703
O	393
A	103
X	64
K	9
J	3
I	3
C	2
F	1

Πίνακας 6: Τα αποτελέσματα του Query 3 για τις bottom 3 περιοχές με βάση το εισόδημα

```

=====
Explain for first join:
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- Project [LAT#43, LON#44, DATE OCC#129, Vict Descent#30, ZIPcode#145]
   +- BroadcastHashJoin [LAT#43, LON#44], [LAT#90, LON#91], Inner, BuildRight, false
      :- Project [date_format(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false), yyyy, Some(Europe/Athens)) AS DATE OCC#129, Vict Descent#30, LAT#43, LON#44]
         : +- Filter (((date_format(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false), yyyy, Some(Europe/Athens)) = 2015) AND atleastnonnulls(1, Vict Descent#30)) AND isnotnull(LAT#43)) AND isnotnull(LON#44))
            : +- FileScan csv [DATE OCC#19,Vict Descent#30,LAT#43,LON#44] Batched: false, DataFilters: [(date_format(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false), yyyy, Some(Europe/Athens)) = 2015) AND atleastnonnulls(1, Vict Descent#30)) AND isnotnull(LAT#43)) AND isnotnull(LON#44)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:9000/datasets/crime_data.csv], PartitionFilters: [], PushedFilters: [IsNotNull(LAT), IsNotNull(LON)], ReadSchema: struct<DATE OCC:string,Vict Descent:string,LAT:string,LON:string>
            +- BroadcastExchange HashedRelationBroadcastMode(List(input[0, string, true], input[1, string, true]),false), [plan_id=99]
            +- SortAggregate(key=[LAT#90, LON#91], functions=[first(ZIPcode#92, false)])
            +- Sort [LAT#90 ASC NULLS FIRST, LON#91 ASC NULLS FIRST], false, 0
            +- Exchange hashpartitioning(LAT#90, LON#91, 200), ENSURE_REQUIREMENTS, [plan_id=95]
            +- SortAggregate(key=[LAT#90, LON#91], functions=[partial_first(ZIPcode#92, false)])
            +- Sort [LAT#90 ASC NULLS FIRST, LON#91 ASC NULLS FIRST], false, 0
            +- Filter (isnotnull(LAT#90) AND isnotnull(LON#91))
            +- FileScan csv [LAT#90,LON#91,ZIPcode#92] Batched: false, DataFilters: [isnotnull(LAT#90), isnotnull(LON#91)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:9000/datasets/revgecoding.csv], PartitionFilters: [], PushedFilters: [IsNotNull(LAT), IsNotNull(LON)], ReadSchema: struct<LAT:string,LON:string,ZIPcode:string>

```

Εικόνα 7: Physical plan για το join εγκλημάτων και ταχυδρομικών κωδικών

```

Explain for second join:
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- Project [Zip Code#150, LAT#43, LON#44, DATE OCC#129, Vict Descent#30, Community#114, Estimated Median Income#115]
   +- BroadcastHashJoin [Zip Code#150], [Zip Code#113], Inner, BuildRight, false
      :- Project [LAT#43, LON#44, DATE OCC#129, Vict Descent#30, ZIPcode#165 AS Zip Code#150]
         : +- BroadcastHashJoin [LAT#43, LON#44], [LAT#90, LON#91], Inner, BuildRight, false
            : :- Project [date_format(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false), yyyy, Some(Europe/Athens)) AS DATE OCC#129, Vict Descent#30, LAT#43, LON#44]
               : +- Filter (((date_format(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false), yyyy, Some(Europe/Athens)) = 2015) AND atleastnonnulls(1, Vict Descent#30)) AND isnotnull(LAT#43)) AND isnotnull(LON#44))
                  : +- FileScan csv [DATE OCC#19,Vict Descent#30,LAT#43,LON#44] Batched: false, DataFilters: [(date_format(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false), yyyy, Some(Europe/Athens)) = 2015) AND atleastnonnulls(1, Vict Descent#30)) AND isnotnull(LAT#43)) AND isnotnull(LON#44)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:9000/datasets/crime_data.csv], PartitionFilters: [], PushedFilters: [IsNotNull(LAT), IsNotNull(LON)], ReadSchema: struct<DATE OCC:string,Vict Descent:string,LAT:string,LON:string>
                  +- BroadcastExchange HashedRelationBroadcastMode(List(input[0, string, true], input[1, string, true]),false), [plan_id=168]
                  +- Filter isnotnull(ZIPcode#165)
                  +- SortAggregate(key=[LAT#90, LON#91], functions=[first(ZIPcode#92, false)])
                  +- Sort [LAT#90 ASC NULLS FIRST, LON#91 ASC NULLS FIRST], false, 0
                  +- Exchange hashpartitioning(LAT#90, LON#91, 200), ENSURE_REQUIREMENTS, [plan_id=163]
                  +- SortAggregate(key=[LAT#90, LON#91], functions=[partial_first(ZIPcode#92, false)])
                  +- Sort [LAT#90 ASC NULLS FIRST, LON#91 ASC NULLS FIRST], false, 0
                  +- Filter (isnotnull(LAT#90) AND isnotnull(LON#91))
                  +- FileScan csv [LAT#90,LON#91,ZIPcode#92] Batched: false, DataFilters: [isnotnull(LAT#90), isnotnull(LON#91)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:9000/datasets/revgecoding.csv], PartitionFilters: [], PushedFilters: [IsNotNull(LAT), IsNotNull(LON)], ReadSchema: struct<LAT:string,LON:string,ZIPcode:string>
                  +- BroadcastExchange HashedRelationBroadcastMode(List(input[0, string, false]),false), [plan_id=172]
                  +- Filter isnotnull(Zip Code#113)
                  +- FileScan csv [Zip Code#113,Community#114,Estimated Median Income#115] Batched: false, DataFilters: [isnotnull(Zip Code#113)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:9000/datasets/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Zip Code)], ReadSchema: struct<Zip Code:string,Community:string,Estimated Median Income:string>

```

Εικόνα 8: Physical plan για το join του συνολικού dataframe και των εισοδημάτων

Στρατηγική Join	Χρόνος εκτέλεσης
Broadcast (*)	1.5 min
Merge	1.7 min
Shuffle Hash	1.7 min
Shuffle Replicate NL	43 min

Πίνακας 7: Οι χρόνοι εκτέλεσης του Query 3

* Η στρατηγική που επιλέχθηκε από τον Catalyst Optimizer

Ζητούμενο 6

Υλοποιήθηκαν οι αλγόριθμοι Broadcast Join και Repartition Join.

Για το Broadcast Join (Απόκομμα Κώδικα 1) ακολουθήθηκε η εξής διαδικασία:

- Κατασκευάζουμε key-value pairs για τους δύο πίνακες, με key το join key και value την εγγραφή
- Κάνουμε broadcast τον μικρό πίνακα στους workers
- Αναζητούμε για κοινό κλειδί με χρήση του broadcast object ως look-up table
- Εάν έχουμε match συνδυάζονται οι εγγραφές, αλλιώς επιστρέφεται None
- Αφαιρούμε τα None

Για το Repartition Join (Απόκομμα Κώδικα 2) ακολουθήθηκε η εξής διαδικασία:

- Κατασκευάζουμε key-value pairs για τους δύο πίνακες, με key το join key και value την εγγραφή χωρίς το κλειδί και με ένα tag (string) για να παραμείνει η πληροφορία της προέλευσης της εγγραφής
- Ενώνουμε τα RDDs
- Κάνουμε groupByKey ώστε να ομαδοποιηθούν τα δεδομένα σε partitions με βάση το κλειδί
- Για κάθε partition ανακατασκευάζουμε τα δύο RDDs με βάση το tag και συνδυάζουμε τις εγγραφές

Το αποτέλεσμα των εκτελέσεων παρατίθεται στο Πίνακα 8.

```
small_dataset = police_stations.keyBy(lambda t: int(t[0]))
broadcast_small = sc.broadcast(small_dataset.collectAsMap())
large_dataset = crime_data.filter(lambda x: x[0] is not None) \
    .keyBy(lambda t: int(t[0]))

def combine_records(row_large_dataset):
    key = row_large_dataset[0]
    matching_value = broadcast_small.value.get(key, None)
    if matching_value:
        return row_large_dataset[1] + matching_value[1:]
    else:
        return None

joined_rdd = large_dataset.map(combine_records) \
    .filter(lambda x: x is not None)
```

Απόκομμα κώδικα 1: Υλοποίηση του Broadcast Join

```
crime_data_tagged = crime_data \
    .map(lambda x: (x[0], ('crime',x[1],x[2],x[3])) )
police_stations_tagged = police_stations \
    .map(lambda x: (x[0], ('police_station',x[1],x[2],x[3])))
union_rdd = crime_data_tagged.union(police_stations_tagged)

def combine_records(dataset_tagged):
    key = dataset_tagged[0]
    val = dataset_tagged[1]

    crime_list, police_stations_list = [], []
    for v in val:
        if v[0] == 'crime':
            crime_list.append(v[1:])
        elif v[0] == 'police_station':
            police_stations_list.append(v[1:])
    to_return = []
    for cr in crime_list:
        for ps in police_stations_list:
            to_return.append((key,) + cr + ps)
    return to_return

joined_rdd = union_rdd.groupByKey() \
    .flatMap(combine_records) \
    .filter(lambda x: x is not None)
```

Απόκομμα κώδικα 2: Υλοποίηση του Repartition Join

Ζητούμενο 7

Υλοποιήθηκε το Query 4 με χρήση DataFrame. Τα αποτελέσματα παρατίθενται στο Πίνακα 8.

DIVISION	average_distance	incidents total
77TH STREET	2.689	17019
SOUTHEAST	2.106	12942
NEWTON	2.017	9846
SOUTHWEST	2.702	8912
HOLLENBECK	2.649	6202
HARBOR	4.075	5621
RAMPART	1.575	5115
MISSION	4.708	4504
OLYMPIC	1.821	4424
NORTHEAST	3.907	3920
FOOTHILL	3.803	3774
HOLLYWOOD	1.461	3641
CENTRAL	1.138	3614
WILSHIRE	2.313	3525
NORTH HOLLYWOOD	2.716	3466
WEST VALLEY	3.532	2902
VAN NUYS	2.220	2733
PACIFIC	3.729	2708
DEVONSHIRE	4.012	2471
TOPANGA	3.481	2283
WEST LOS ANGELES	4.249	1541

Πίνακας 8: Αποτελέσματα του Query 4

Παράρτημα

GitHub repository