

Διαχείριση Δεδομένων Μεγάλης Κλίμακας

Εξαμηνιαία Εργασία

Υποψήφιος Διδάκτωρ:

Ανδριώτης Νικόλαος: 03003244

nandriotis@mail.ntua.gr



ΔΠΜΣ Επιστήμη Δεδομένων και Μηχανική Μάθηση
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο

Ιούνιος 2025

Πίνακας Περιεχομένων

1	Εισαγωγή	3
2	Πρώτο Ζητούμενο	3
3	Δεύτερο Ζητούμενο	3
4	Τρίτο Ζητούμενο	4
4.1	Τρόποι υλοποίησης	4
4.2	Ιδέα και εκτέλεση της υλοποίησης του Query 1	4
4.3	Χρόνοι εκτέλεσης και σχολιασμοί για το Query 1	5
5	Τέταρτο Ζητούμενο	6
5.1	Υλοποίηση του Query 2 με SQL API	6
5.2	Υλοποίηση του Query 2 με DataFrame API	7
5.3	Υλοποίηση του Query 2 με RDD API	7
5.4	Χρόνοι εκτέλεσης και συμπεράσματα για το Query 2	7
6	Πέμπτο Ζητούμενο	8
6.1	Υλοποίηση του Query 3 με Dataframe API	8
6.2	Υλοποίηση του Query 3 με RDD API	9
6.3	Χρόνοι εκτέλεσης και συμπεράσματα για το Query 3	10
7	Έκτο Ζητούμενο	10
7.1	Horizontal scaling	11
7.2	Vertical scaling	11
8	Έβδομο ζητούμενο	12
8.1	Στρατηγική του Catalyst Optimizer για το Q3	12
8.2	Στρατηγική του Catalyst Optimizer για το Q4	12

Λίστα Σχημάτων

1	Files in HDFS directory	4
2	Αποτελέσματα για το Query 1	5
3	Spark-Submit Settings	5
4	Q2 results	7
5	Q3 results	9
6	Q4 results	11
7	Q3 - PQP - Physical Query Plan	12
8	Q4 - PQP - 1st join	13
9	Q4 - PQP - 2nd join	13

1 Εισαγωγή

Στην παρούσα εργασία με την βοήθεια των κατανεμημένων συστημάτων Apache Hadoop και Apache spark θα κάνουμε ανάλυση δεδομένων μεγάλου όγκου. Τα δεδομένα μας βρίσκονται ήδη ανεβασμένα στο κατανεμημένο σύστημα αρχείων HDFS και είναι τα παρακάτω:

Σύνολο Δεδομένων	HDFS URI
Los Angeles Crime Data (2010-2019)	<code>hdfs://hdfs-namenode:9000/user/root/data/LA_Crime_Data_2010_2019.csv</code>
Los Angeles Crime Data (2020-)	<code>hdfs://hdfs-namenode:9000/user/root/data/LA_Crime_Data_2020_2025.csv</code>
LA Police Stations	<code>hdfs://hdfs-namenode:9000/user/root/data/LA_Police_Stations.csv</code>
Median Household Income by Zip Code	<code>hdfs://hdfs-namenode:9000/user/root/data/LA_income_2015.csv</code>
2010 Census Populations by Zip Code	<code>hdfs://hdfs-namenode:9000/user/root/data/2010_Census_Populations_by_Zip_Code.csv</code>
MO Codes in Numerical Order	<code>hdfs://hdfs-namenode:9000/user/root/data/MO_codes.txt</code>

Table 1: Πίνακας των συνόλων δεδομένων και των αντίστοιχων HDFS URI.

Θα λύσουμε την εκτέλεση τεσσάρων queries τα οποία αφορούν ένα dataset που περιέχει καταγραφές εγκλημάτων για την περιοχή του LA από το 2010 μέχρι σήμερα. Συμπληρωματικά, χρησιμοποιήθηκαν ακόμα τέσσερα datasets τα οποία περιγράφονται στην εργασία. Όλα τα scripts με τον κώδικα βρίσκονται στο αποθετήριο μου στο Github: [Big Data repo](#).

2 Πρώτο Ζητούμενο




Για το πρώτο ζητούμενο συνδέθηκα με την απομακρυσμένη υποδομή kubernetes μέσω του OVPN, και ακολουθήθηκε ο οδηγός του εργαστηρίου για το στήσιμο του Spark Job History Server, μέσω docker και docker compose τοπικά, με αποτέλεσμα να μπορώ να κάνω submit δουλειές και να αντλώ δεδομένα από την απομακρυσμένη υποδομή του HDFS.

3 Δεύτερο Ζητούμενο

Ανεβάζουμε τον κώδικα "csv_to_parquet.py" και εκτελούμε με `spark-submit hdfs://hdfs-namenode:9000/user/nandriotis/csv_to_parquet.py`. Έπειτα, μπορούμε να δούμε ότι τα datasets μας είναι πλέον στο σωστό path και στη μορφή parquet στο παρακάτω screenshot.

/user/nandriotis/data/parquet

Go!



Show

25

entries

Search:

<div><div><div></div><div></div></div></div>	<div><div><div></div><div></div></div></div> Permission	<div><div><div></div><div></div></div></div> Owner	<div><div><div></div><div></div></div></div> Group	<div><div><div></div><div></div></div></div> Size	<div><div><div></div><div></div></div></div> Last Modified	<div><div><div></div><div></div></div></div> Replication	<div><div><div></div><div></div></div></div> Block Size	<div><div><div></div><div></div></div></div> Name	<div><div><div></div><div></div></div></div>
<div><div><div></div><div></div></div></div>	<div><div><div>drwxrwxr-x+</div><div></div></div></div>	<div><div><div>spark</div><div></div></div></div>	<div><div><div>nandriotis</div><div></div></div></div>	<div><div><div>0 B</div><div></div></div></div>	<div><div><div>Jun 21 13:47</div><div></div></div></div>	<div><div><div>0</div><div></div></div></div>	<div><div><div>0 B</div><div></div></div></div>	<div><div><div>2010_Census_Populations_by_Zip_Code.parquet</div><div></div></div></div>	<div><div><div></div><div></div></div></div>
<div><div><div></div><div></div></div></div>	<div><div><div>drwxrwxr-x+</div><div></div></div></div>	<div><div><div>spark</div><div></div></div></div>	<div><div><div>nandriotis</div><div></div></div></div>	<div><div><div>0 B</div><div></div></div></div>	<div><div><div>Jun 21 13:47</div><div></div></div></div>	<div><div><div>0</div><div></div></div></div>	<div><div><div>0 B</div><div></div></div></div>	<div><div><div>LA_Crime_Data_2010_2019.parquet</div><div></div></div></div>	<div><div><div></div><div></div></div></div>
<div><div><div></div><div></div></div></div>	<div><div><div>drwxrwxr-x+</div><div></div></div></div>	<div><div><div>spark</div><div></div></div></div>	<div><div><div>nandriotis</div><div></div></div></div>	<div><div><div>0 B</div><div></div></div></div>	<div><div><div>Jun 21 13:47</div><div></div></div></div>	<div><div><div>0</div><div></div></div></div>	<div><div><div>0 B</div><div></div></div></div>	<div><div><div>LA_Crime_Data_2020_2025.parquet</div><div></div></div></div>	<div><div><div></div><div></div></div></div>
<div><div><div></div><div></div></div></div>	<div><div><div>drwxrwxr-x+</div><div></div></div></div>	<div><div><div>spark</div><div></div></div></div>	<div><div><div>nandriotis</div><div></div></div></div>	<div><div><div>0 B</div><div></div></div></div>	<div><div><div>Jun 21 13:47</div><div></div></div></div>	<div><div><div>0</div><div></div></div></div>	<div><div><div>0 B</div><div></div></div></div>	<div><div><div>LA_Police_Stations.parquet</div><div></div></div></div>	<div><div><div></div><div></div></div></div>
<div><div><div></div><div></div></div></div>	<div><div><div>drwxrwxr-x+</div><div></div></div></div>	<div><div><div>spark</div><div></div></div></div>	<div><div><div>nandriotis</div><div></div></div></div>	<div><div><div>0 B</div><div></div></div></div>	<div><div><div>Jun 21 13:47</div><div></div></div></div>	<div><div><div>0</div><div></div></div></div>	<div><div><div>0 B</div><div></div></div></div>	<div><div><div>LA_income_2015.parquet</div><div></div></div></div>	<div><div><div></div><div></div></div></div>
<div><div><div></div><div></div></div></div>	<div><div><div>drwxrwxr-x+</div><div></div></div></div>	<div><div><div>spark</div><div></div></div></div>	<div><div><div>nandriotis</div><div></div></div></div>	<div><div><div>0 B</div><div></div></div></div>	<div><div><div>Jun 21 13:47</div><div></div></div></div>	<div><div><div>0</div><div></div></div></div>	<div><div><div>0 B</div><div></div></div></div>	<div><div><div>MO_codes.parquet</div><div></div></div></div>	<div><div><div></div><div></div></div></div>

Figure 1: Files in HDFS directory

4 Τρίτο Ζητούμενο

4.1 Τρόποι υλοποίησης

Ζητείται να γίνει το Query 1 χρησιμοποιώντας RDD και DataFrame APIs. Επίσης, για το DataFrame API να κάνουμε χρήση udf ή όχι. Συνδυάζοντας τα ζητούμενα APIs φτιάχνουμε τρία αρχεία κώδικα:

- Με Dataframe API και udf
- Με Dataframe API χωρίς udf
- Με RDD API

4.2 Ιδέα και εκτέλεση της υλοποίησης του Query 1

Η pySpark εφαρμογή αναλύει τα δεδομένα εγκληματικότητας του Λος Άντζελες για να μετρήσει τον αριθμό των επιθέσεων με βαριές κατηγορίες ανά ηλικιακή ομάδα. Αρχικά, διαβάζει δύο ξεχωριστά σύνολα δεδομένων εγκληματικότητας (από το 2010-2019 και από το 2020 και μετά) που είναι αποθηκευμένα σε μορφή Parquet από το HDFS και τα συγχωνεύει σε ένα ενιαίο DataFrame (ή RDD αντίστοιχα). Στη συνέχεια προχωρά στον καθαρισμό των δεδομένων φιλτράροντας τις εγγραφές με ελλείπουσες περιγραφές εγκλημάτων ή ηλικίες θυμάτων. Στη συνέχεια, εστιάζει ειδικά στα εγκλήματα που χαρακτηρίζονται ως «AGGRAVATED ASSAULT», όπου η ηλικία του θύματος είναι μεγαλύτερη του μηδενός. Μετά το φιλτράρισμα, κατηγοριοποιεί τα θύματα σε τέσσερις διαφορετικές ηλικιακές ομάδες: «Kids» (κάτω των 18 ετών), «Young Adults» (18-24 ετών), «Adults» (25-64 ετών) και «Elderly» (άνω των 64 ετών) χρησιμοποιώντας την συνάρτηση when() του Spark. Τέλος, ομαδοποιεί τα δεδομένα ανά αυτές τις νεοδημιουργηθείσες ηλικιακές ομάδες, μετρά τον αριθμό των περιστατικών σε κάθε μία και ταξινομεί τα αποτελέσματα σε φθίνουσα σειρά για να δείξει ποια ηλικιακή ομάδα υφίσταται τις περισσότερες βαριές επιθέσεις. Το αποτέλεσμα είναι το ίδιο και για τα τρία scripts, το οποίο φαίνεται στην Εικόνα 2.

Age Group	Aggravated_Incidents_Count
Adults	122727
Young Adults	34036
Kids	10976
Elderly	6082

Figure 2: Αποτελέσματα για το Query 1

4.3 Χρόνοι εκτέλεσης και σχολιασμοί για το Query 1

Αφότου τρέξουμε τα τρία scripts που αφορούν το Query 1, συγκρίνουμε τους χρόνους. Αυτό που παρατηρήθηκε είναι inconsistent χρόνοι που πιθανότατα έχουν να κάνουν με την κίνηση στην απομακρυσμένη υποδομή. Προσπάθησα με τη χρήση ιδίων τιμών limit και request στις ρυθμίσεις του spark-submit να το αποφύγω αλλά πάλι έβλεπα διακύμανση στις τιμές.

```
spark.master k8s://https://termi7.cslab.ece.ntua.gr:6443
spark.submit.deployMode cluster
spark.hadoop.fs.permissions.umask-mode 000
spark.kubernetes.authenticate.driver.serviceAccountName spark
spark.kubernetes.namespace nandriotis-priv
spark.executor.instances 5
spark.executor.cores 5
spark.kubernetes.executor.request.cores 5
spark.kubernetes.executor.limit.cores 5
spark.executor.memory 1500m
spark.kubernetes.executor.memoryOverhead 500m
spark.driver.cores 1
spark.kubernetes.driver.request.cores 1
spark.kubernetes.driver.limit.cores 1
spark.driver.memory 512m
spark.kubernetes.driver.memoryOverhead 256m
spark.kubernetes.container.image=apache/spark
spark.kubernetes.submission.waitAppCompletion false
spark.eventLog.enabled true
spark.eventLog.dir hdfs://hdfs-namenode:9000/user/nandriotis/logs
spark.history.fs.logDirectory hdfs://hdfs-namenode:9000/user/nandriotis/logs
```

Figure 3: Spark-Submit Settings

Παρόλα αυτά παρακάτω παίρνουμε τα αποτελέσματα. Οι χρόνοι φαίνονται στον παρακάτω πίνακα και έχουν παρθεί από τον history server <http://localhost:18080/>

Dataframe	Dataframe (udf)	RDD
1.1 min	58 sec	1.3 min

Αρχικά, το RDD API είναι πιο low level και δεν επωφελείται το ίδιο από την αυτόματη βελτιστοποίηση όπως το DataFrame API. Με τα RDDs, ο χρήστης έχει περισσότερο έλεγχο αλλά και περισσότερη ευθύνη για τη βελτιστοποίηση του σχεδίου εκτέλεσης. Τα αποτελέσματα με και χωρίς udf είναι πολύ κοντά και το περίεργο είναι ότι με udf έχουμε πιο γρήγορη εκτέλεση. Αυτό δεν θα το περιμέναμε αφού θεωρητικά χρησιμοποιώντας τις native εντολές when() είμαστε αρκετά πιο γρήγοροι.

5 Τέταρτο Ζητούμενο

5.1 Υλοποίηση του Query 2 με SQL API

Αυτό το query βρίσκει τα 3 Αστυνομικά Τμήματα με το υψηλότερο ποσοστό κλεισμένων (περατωμένων) υποθέσεων για κάθε έτος.

Αρχικά, φορτώνει τρία ξεχωριστά σύνολα δεδομένων από το HDFS: δύο για δεδομένα εγκληματικότητας (από διαφορετικές χρονικές περιόδους) και ένα που περιέχει πληροφορίες για τα αστυνομικά τμήματα του Λος Άντζελες. Τα δύο σύνολα δεδομένων για την εγκληματικότητα συγχωνεύονται σε ένα ενιαίο DataFrame. Για να είναι αυτά τα DataFrames επεξεργάσιμα με SQL, καταχωρούνται ως προσωρινά views με ονόματα crimes και stations.

Δόμηση ερωτημάτων με CTEs: Η κύρια λογική ενσωματώνεται σε ένα ενιαίο μεγάλο ερώτημα SQL που χρησιμοποιεί common table expressions (CTEs), οι οποίες εισάγονται με τη ρήτρα WITH. Αυτή η τεχνική καθιστά το ερώτημα ευκολότερο στην ανάγνωση και τη διαχείριση.

Το πρώτο βήμα είναι ο υπολογισμός του συνολικού αριθμού των υποθέσεων και του αριθμού των «κλειστών» υποθέσεων για κάθε PREC (precinct) για κάθε έτος. Το ονομάζουμε case_stats.

Το δεύτερο βήμα χρησιμοποιεί τα δεδομένα από το case_stats. Υπολογίζει το ποσοστό closed_case_rate ως ποσοστό. Χρησιμοποιεί τη συνάρτηση παραθύρου RANK(). Η συνάρτηση αυτή χωρίζει τα δεδομένα ανά έτος και στη συνέχεια, εντός κάθε έτους, αποδίδει μια κατάταξη σε κάθε περιφέρεια με βάση το ποσοστό closed_case_rate (από το υψηλότερο προς το χαμηλότερο). Το ονομάζουμε ranked_stats.

Η τελική δήλωση SELECT αντλεί δεδομένα από το CTE ranked_stats. Συνδέεται με την προβολή σταθμών χρησιμοποιώντας το αναγνωριστικό περιφέρειας για να λάβει το πλήρες όνομα της περιφέρειας (π.χ. «Hollywood»). Φιλτράρει τα αποτελέσματα για να κρατήσει μόνο τις γραμμές όπου η κατάταξη είναι 3 ή λιγότερο. Τέλος, ταξινομεί τα αποτελέσματα ανά έτος και κατάταξη για μια καθαρή, ευανάγνωστη έξοδο.

Τα αποτελέσματα μέχρι το 2016 για εξοικονόμηση χώρου φαίνονται παρακάτω:

year	precinct	closed_case_rate	#
2010	RAMPART	32.84713448949121	1
2010	OLYMPIC	31.515289821999087	2
2010	HARBOR	29.36028339237341	3
2011	OLYMPIC	35.040060090135206	1
2011	RAMPART	32.4964471814306	2
2011	HARBOR	28.51336246316431	3
2012	OLYMPIC	34.328183520599254	1
2012	RAMPART	32.46000463714352	2
2012	HARBOR	29.5064604956577	3
2013	OLYMPIC	33.58217940999398	1
2013	RAMPART	32.118311242022585	2
2013	HARBOR	29.723638951488557	3
2014	SOUTHEAST	66.66666666666666	1
2014	TOPANGA	50.0	2
2014	MISSION	33.33333333333333	3
2015	VAN NUYS	32.27467811158798	1
2015	MISSION	30.467430073211936	2
2015	WEST VALLEY	30.335662733417344	3
2016	VAN NUYS	31.952285558186123	1
2016	WEST VALLEY	30.957771434196278	2
2016	FOOTHILL	29.920520683396955	3

Figure 4: Q2 results

5.2 Υλοποίηση του Query 2 με DataFrame API

Η λογική είναι ακριβώς η ίδια με την SQL υλοποίηση.

5.3 Υλοποίηση του Query 2 με RDD API

Για τα RDDs, η ιδέα είναι όμοια, όμως η εκτέλεση έχει κάποιες διαφορές. Πρέπει ουσιαστικά να φροντίσουμε να κάνουμε broadcast το μικρό dataset με τους σταθμούς. Τα δεδομένα των αστυνομικών τμημάτων είναι πολύ μικρά. Αντί να εκτελέσουμε μια αργή κατανεμημένη «ένωση» αργότερα, χρησιμοποιούμε μια πολύ πιο αποτελεσματική τεχνική. Διαβάζουμε τα δεδομένα του σταθμού, τα αντιστοιχίζουμε σε ζεύγη κλειδιών-τιμών (PrecinctID, DivisionName) και χρησιμοποιούμε την `.collectAsMap()` για να φέρουμε αυτά τα δεδομένα στο κύριο πρόγραμμα οδήγησης ως ένα απλό λεξικό Python. Αυτή είναι μια κρίσιμη βελτιστοποίηση επιδόσεων. Μια broadcast μεταβλητή παίρνει το μικρό λεξικό `stations_map` και στέλνει ένα αντίγραφο μόνο για ανάγνωση σε κάθε worker στη συστάδα. Όταν οι εργασίες μας πρέπει να βρουν το όνομα ενός σταθμού αργότερα, μπορούν να το αναζητήσουν στο τοπικό τους αντίγραφο αμέσως, πράγμα που είναι πολύ πιο γρήγορο από το να το ζητούν επανειλημμένα από τον οδηγό.

5.4 Χρόνοι εκτέλεσης και συμπεράσματα για το Query 2

Στον πίνακα παρακάτω βλέπουμε τους χρόνους εκτέλεσης για τις τρεις υλοποιήσεις μας, τους οποίους βρήκαμε μέσω του history server:

SQL	RDD	Dataframe
1.2 min	1.9 min	1.2 min

Φαίνεται ότι τα SQL και Dataframe API's έχουν ίδιους χρόνους και είναι ταχύτερα από το RDD API. Αυτό εξηγείται από το γεγονός ότι με το SQL/DataFrame API χρησιμοποιείται ο Catalyst Optimizer ο οποίος εφαρμόζει αυτόματα διάφορες τεχνικές βελτιστοποίησης όπως το predicate pushdown, το column pruning και τις βελτιστοποιήσεις λογικού σχεδίου. Αυτές οι βελτιστοποιήσεις οδηγούν σε πιο αποδοτικά σχέδια εκτέλεσης και, κατά συνέπεια, σε ταχύτερη απάντηση των queries. Αντίθετα, το RDD API είναι πιο low level και δεν επωφελείται το ίδιο από την αυτόματη βελτιστοποίηση όπως το SQL/DataFrame API. Με τα RDDs, ο χρήστης έχει περισσότερο έλεγχο αλλά και περισσότερη ευθύνη για τη βελτιστοποίηση του σχεδίου εκτέλεσης.

6 Πέμπτο Ζητούμενο

Για αυτό το ζητούμενο ζητείται και η χρήση δύο διαφορετικών τύπων αρχείων για την εισαγωγή των δεδομένων, .csv και .parquet. Τα parquet αρχεία είναι σχεδιασμένα ώστε να έχουν μεγάλες επιδόσεις αφού μπορούν να διαχειριστούν πολύ καλά μεγάλο όγκο δεδομένων. Γι αυτό τον λόγο, αναμένουμε ότι οι χρόνοι που θα χρειαστούν τα parquet αρχεία θα είναι καλύτεροι από τους χρόνους των csv αρχείων.

6.1 Υλοποίηση του Query 3 με Dataframe API

Ο κύριος στόχος αυτού του query είναι να υπολογίσει το κατά κεφαλήν εισόδημα για κάθε ταχυδρομικό κώδικα στο Λος Άντζελες συνδυάζοντας δύο διαφορετικά σύνολα δεδομένων.

Φορτώνουμε και προετοιμάζουμε τα δεδομένων πληθυσμού: Διαβάζουμε το αρχείο Census Population (csv ή parquet) από το HDFS. Στη συνέχεια, επιλέγουμε μόνο τις τρεις στήλες που χρειάζονται -Zip Code, Total Population και Total Households- και τις μετανομάζουμε σε απλούστερα ονόματα (zip_code, population, households) για ευκολότερη χρήση.

Θα χρειαστεί και καθαρισμός δεδομένων εισοδήματος: Η στήλη εισόδημα αποθηκεύεται ως κείμενο με μορφοποίηση όπως \$123,456. Για να χρησιμοποιηθεί αυτό για υπολογισμούς, το σενάριο εκτελεί ένα βήμα καθαρισμού.

Συνδέουμε τα σύνολα δεδομένων πληθυσμού με τα καθαρισμένα δεδομένα εισοδήματος. Χρησιμοποιούμε ένα inner join στη στήλη zip_code. Αυτό σημαίνει ότι μόνο οι ταχυδρομικοί κώδικες που υπάρχουν τόσο στο αρχείο πληθυσμού όσο και στο αρχείο εισοδήματος θα συμπεριληφθούν στο τελικό αποτέλεσμα.

Τέλος υπολογίζουμε το κατά κεφαλήν εισόδημα: Αυτό είναι το βασικό βήμα υπολογισμού. Δημιουργούμε μια νέα στήλη που ονομάζεται income_per_capita. Προσεγγίζουμε το συνολικό εισόδημα για έναν ταχυδρομικό κώδικα πολλαπλασιάζοντας το median_income επί τον αριθμό των νοικοκυριών. Στη συνέχεια διαιρούμε αυτό το συνολικό εισόδημα με τον πληθυσμό για να προκύψει η κατά κεφαλήν τιμή. Τα αποτελέσματα φαίνονται στο screenshot:

zip_code	income_per_capita
90001	7696.520346699353
90002	6965.130956796752
90003	7271.478737210636
90004	14726.258668382117
90005	12433.328414850987
90006	9915.121348314608
90007	6510.23890518084
90008	15655.097101494108
90010	24266.58
90011	6454.820082393255
90012	10484.048226859146
90013	10838.854230377166
90014	13867.948322626695
90015	11600.931212472347
90016	13001.88776367762
90017	8939.618478626726
90018	10639.93007503549
90019	16866.07440503894
90020	16463.99224985244
90021	5062.286256643888

Figure 5: Q3 results

6.2 Υλοποίηση του Query 3 με RDD API

Αναφέρουμε την κύρια λογική χρησιμοποιώντας μετασχηματισμούς RDD χαμηλού επιπέδου.

Ένωση RDDs: Το σενάριο χρησιμοποιεί τον μετασχηματισμό `.join()` στα δύο RDDs. Επειδή και τα δύο RDD προετοιμάστηκαν με κλειδί το `zip_code`, το Spark κάνει shuffle τα δεδομένα έτσι ώστε όλες οι τιμές για το ίδιο κλειδί να συγκεντρώνονται. Το `join_rdd` που προκύπτει έχει μια nested tuple δομή: `(zip_code, ((population, households), median.income))`.

Υπολογισμός του κατά κεφαλήν εισοδήματος: Εφαρμόζεται ένας μετασχηματισμός `.map()` στο `joined_rdd`. Για κάθε γραμμή (κάθε συνδεδεμένος ταχυδρομικός κώδικας), καλεί την προσαρμοσμένη συνάρτηση της Python `calculate_per_capita`.

Επιστρέφει ένα νέο ζεύγος κλειδιών-τιμών: `(zip_code, calculated_per_capita.income)`.

Ταξινόμηση του RDD: Το τελικό RDD ταξινομείται με βάση τον ταχυδρομικό κώδικα σε αύξουσα σειρά.

6.3 Χρόνοι εκτέλεσης και συμπεράσματα για το Query 3

Dataframe_CSV	Dataframe_Parquet	RDD
1.1 min	60 sec	1.1 min

Επιβεβαιώνουμε ότι το DataFrame API είναι πιο γρήγορο όταν χρησιμοποιούμε Parquet. Επίσης το RDD διαβάζει από Parquet αρχεία αλλά είναι πιο αργό.

7 Έκτο Ζητούμενο

Η λογική έχει ως εξής:

Φορτώνουμε όλα τα απαραίτητα δεδομένα από τέσσερα διαφορετικά αρχεία Parquet: δύο σύνολα δεδομένων εγκλημάτων (τα οποία συγχωνεύονται αμέσως σε ένα), τις τοποθεσίες των αστυνομικών τμημάτων και έναν κατάλογο κωδικών "Modus Operandi" (MO) με τις περιγραφές τους. Απομονώνουμε πρώτα τους συγκεκριμένους τύπους εγκλημάτων που μας ενδιαφέρουν, κωδικούς δηλαδή των οποίων οι περιγραφές περιλαμβάνουν τη λέξη "gun" ή "weapon".

Πριν από οποιαδήποτε ανάλυση, τα κύρια δεδομένα εγκλημάτων καθαρίζονται και προετοιμάζονται. Φιλτράρονται όλα τα περιστατικά εγκλημάτων από τα οποία λείπουν οι συντεταγμένες (LAT, LON) (Null Island) ή στα οποία δεν αναφέρονται κωδικοί MO.

Έπειτα εκτελούμε ένα inner join μεταξύ των προετοιμασμένων δεδομένων εγκλημάτων και των φιλτραρισμένων κωδικών "όπλων" MO. Το αποτέλεσμα είναι ένα νέο DataFrame, crimes_with_weapons, το οποίο περιέχει μόνο τα περιστατικά εγκλημάτων που έχουν κωδικοποιηθεί επίσημα ως εμπλεκόμενα με όπλο. Αυτό το νέο DataFrame των εγκλημάτων που σχετίζονται με όπλα συνδέεται στη συνέχεια με τα δεδομένα police_stations.

Για την απόσταση μεταξύ κάθε τόπου εγκλήματος και του αστυνομικού τμήματος χρησιμοποιήσα τον τύπο Haversine, ο οποίος είναι ο συνήθης τρόπος υπολογισμού των αποστάσεων σε μια σφαίρα. Αυτό υλοποιείται εξ ολοκλήρου με εγγενείς συναρτήσεις της Spark SQL για μέγιστη απόδοση.

Με όλα τα δεδομένα προετοιμασμένα και τις αποστάσεις υπολογισμένες, χρησιμοποιούμε την .groupBy("division") για να ομαδοποιήσουμε όλα τα περιστατικά με βάση το αστυνομικό τμήμα τους. Για κάθε τμήμα, υπολογίζουμε δύο συγκεντρωτικές τιμές, incidents_total και average_distance.

Τα αποτελέσματα φαίνονται παρακάτω:

division	incidents_total	average_distance
77TH STREET	67934	2.626
SOUTHWEST	49864	2.6
SOUTHEAST	44888	2.071
NEWTON	44385	2.062
CENTRAL	38978	0.97
OLYMPIC	38021	1.759
RAMPART	38006	1.518
MISSION	28600	4.713
HOLLYWOOD	27078	1.397
HARBOR	26003	3.913
NORTH HOLLYWOOD	24196	2.546
FOOTHILL	23962	4.13
HOLLENBECK	23328	2.567
WEST VALLEY	19200	3.372
NORTHEAST	18889	4.019
VAN NUYS	18364	2.079
DEVONSHIRE	17477	4.008
TOPANGA	17299	3.365
WILSHIRE	16479	2.342
PACIFIC	13784	3.944

Figure 6: Q4 results

7.1 Horizontal scaling

Executors	Cores per Executor	Memory per Executor	Time
2	4	8GB	58 sec
4	2	4GB	54 sec
8	1	2GB	60 sec

Το συγκεκριμένο query, περιλαμβάνει ένα join και ένα groupBy (λειτουργίες που προκαλούν shuffle), η ισορροπημένη προσέγγιση (4 executors × 2 cores) δίνει τον καλύτερο χρόνο εκτέλεσης. Οι 8 executors μπορεί να έχουν υπερβολικό network overhead, ενώ οι 2 executors μπορεί να μην αξιοποιούν πλήρως τον παραλληλισμό.

7.2 Vertical scaling

Executors	Cores per Executor	Memory per Executor	Time
2	1	2GB	51
2	2	4GB	49
2	4	8GB	42

Τα αποτελέσματα θα δείχνουν ξεκάθαρα ότι περισσότεροι πόροι οδηγούν σε καλύτερη απόδοση. Ο χρόνος εκτέλεσης μειώνεται καθώς αυξάνουμε τους πυρήνες και τη μνήμη ανά executor. Αυτό αποδεικνύει την ικανότητα της Spark να κλιμακώνεται κάθετα όταν της παρέχονται περισσότεροι πόροι.

8 Έβδομο ζητούμενο

8.1 Στρατηγική του Catalyst Optimizer για το Q3

Το βασικό join στο Query 3 είναι η συνένωση των δεδομένων πληθυσμού (population_df) με τα δεδομένα εισοδήματος (income_df) με κλειδί το zip_code.

Η επιλογή της στρατηγικής Broadcast Hash Join (BHJ) από τον Catalyst Optimizer είναι απόλυτα δικαιολογημένη και αποτελεί τη βέλτιστη επιλογή για το συγκεκριμένο join, αφού αυτή η στρατηγική χρησιμοποιείται όταν το ένα από τα δύο DataFrames που συμμετέχουν στο join είναι αρκετά μικρό ώστε να χωράει στη μνήμη ενός executor node.

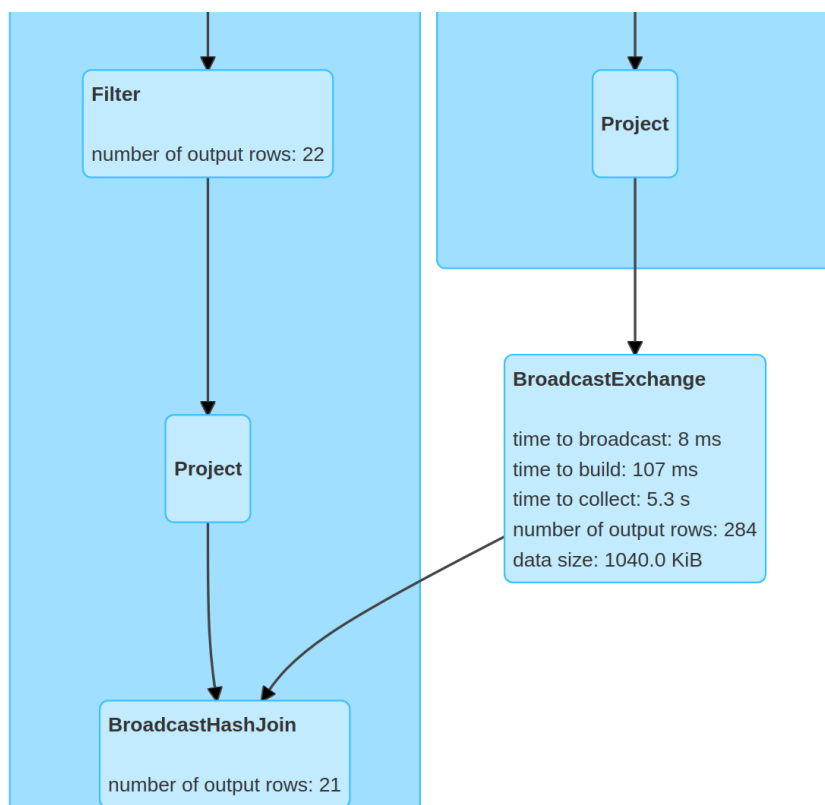


Figure 7: Q3 - PQP - Physical Query Plan

8.2 Στρατηγική του Catalyst Optimizer για το Q4

Το πρώτο join είναι μεταξύ των εγκλημάτων και του φιλτραρισμένου MoCodes dataset. Πάλι περιμένουμε να δούμε BHJ. Παρακάτω το πλάνο που δείχνει ο history server:

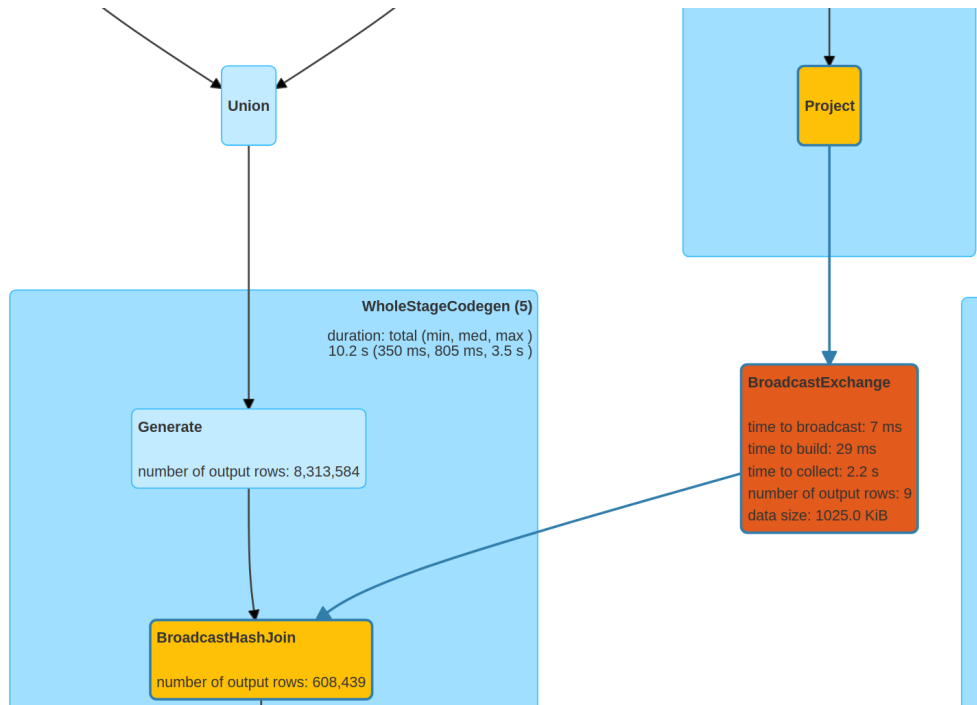


Figure 8: Q4 - PQP - 1st join

Όπως φαίνεται μετά το φιλτράρισμα του MoCodes έχουμε μόνο 9 rows, άρα προφανώς ο optimizer κάνει broadcast exchange αυτά τα πολύ λίγα data, και BHJ με το μεγάλο dataset.

Το δεύτερο join γίνεται μεταξύ του αποτελέσματος του προηγούμενου join (crimes_with_weapons) με το πολύ μικρό police_stations DataFrame. Και πάλι λογικό το BHJ.

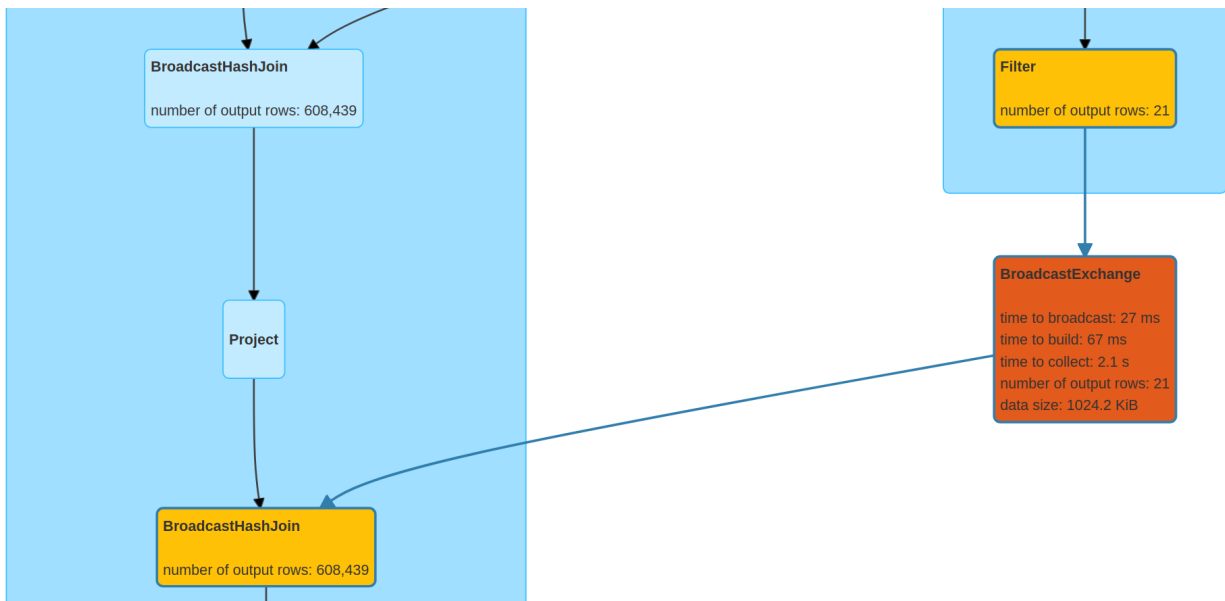


Figure 9: Q4 - PQP - 2nd join