**ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ**

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΠΜΣ ΕΠΙΣΤΗΜΗ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ

---

# Ανάλυση δεδομένων με χρήση Apache Hadoop και Spark

Διαχειριση Δεδομενων Μεγαλης Κλιμακας
ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

**Χρυσοβαλάντη Λυμπέρη**
03400257

Αθήνα 2025

# 1 Εισαγωγή - Προπαρασκευή

Στο πλαίσιο της προπαρασκευής, ολοκληρώθηκε η διαδικασία σύνδεσης με την απομακρυσμένη υποδομή Kubernetes του ΕΜΠ, ακολουθώντας τα βήματα που περιγράφονται στους οδηγούς του μαθήματος. Παράλληλα, διαμορφώθηκε τοπικά ο Spark Job History Server μέσω docker και docker compose, ώστε να αντλεί και να προβάλλει τα δεδομένα εκτέλεσης από το HDFS της απομακρυσμένης υποδομής.

# 2 Μετατροπή σε parquet

Για τη βελτιστοποίηση της αποθήκευσης και επεξεργασίας των δεδομένων, όλα τα αρχεία του project μετατράπηκαν σε μορφή Parquet και αποθηκεύτηκαν στο HDFS στον φάκελο `/user/{chrysovalantilymperi}/data/parquet/`.

Η διαδικασία έγινε με χρήση της βιβλιοθήκης PySpark ως εξής:

- Διαβάστηκαν τα αρχεία csv από το HDFS με χρήση του Spark.
- Καθαρίστηκαν τα ονόματα των στηλών από περιττά κενά για να αποφευχθούν σφάλματα στις ενώσεις (unions) και στα queries.
- Τα DataFrames που προέκυψαν αποθηκεύτηκαν ως αρχεία Parquet στο HDFS.

Ιδιαίτερη μεταχείριση χρειάστηκε το αρχείο `MO Codes`, το οποίο ήταν απλό κείμενο και όχι csv. Για αυτό:

- Διαβάστηκε κάθε γραμμή ως text.
- Διαχωρίστηκαν ο κωδικός και η περιγραφή με βάση τις θέσεις χαρακτήρων.
- Το τελικό DataFrame (Code, Description) αποθηκεύτηκε και αυτό σε μορφή Parquet.

Έτσι, διασφαλίστηκε ομοιομορφία και ταχύτητα για τα επόμενα ερωτήματα και την ανάλυση που ακολουθεί.

# 3 Query 1

Για το πρώτο ζητούμενο χρησιμοποιούμε το βασικό σύνολο δεδομένων που θα χρησιμοποιηθεί για όλη την εργασία και το οποίο προέρχεται από το δημόσιο αποθετήριο δεδομένων της κυβέρνησης των Ηνωμένων Πολιτειών της Αμερικής. Συγκεκριμένα, περιλαμβάνει δεδομένα καταγραφής εγκλημάτων για το Los Angeles από το 2010 μέχρι σήμερα. Ο στόχος του Query 1 ήταν η ομαδοποίηση των περιστατικών εγκλήματος που περιγράφονται ως “aggravated assault” με βάση τις ηλικιακές ομάδες των θυμάτων. Για την επίλυση, χρησιμοποιήθηκαν τρεις διαφορετικές υλοποιήσεις: με DataFrame API, με DataFrame API και UDF, και με το RDD API.

## 3.1 Με χρήση DataFrames

Στην προσέγγιση χρησιμοποιούμε το DataFrame API. Αρχικά διαβάζονται τα δεδομένα εγκλημάτων από δύο αρχεία CSV και ενώνονται σε ένα ενιαίο σύνολο, αφού προηγουμένως αφαιρεθούν πιθανά κενά στα ονόματα των στηλών. Στη συνέχεια, το ενωμένο DataFrame φιλτράρεται ώστε να διατηρηθούν μόνο τα περιστατικά που περιέχουν τον όρο *"aggravated assault"* στην περιγραφή του εγκλήματος. Προστίθεται μια νέα στήλη που κατηγοριοποιεί τα θύματα σε ηλικιακές ομάδες βάσει του πεδίου ηλικίας. Έπειτα, εφαρμόζεται ομαδοποίηση κατά ηλικιακή ομάδα και υπολογίζεται το πλήθος των περιστατικών ανά κατηγορία. Τέλος, τα αποτελέσματα ταξινομούνται κατά φθίνουσα σειρά και αποθηκεύονται σε μορφή CSV στο HDFS.

**Χρόνος εκτέλεσης:** `55s`

## 3.2 Με χρήση DataFrames + UDF

Σε αυτήν την προσέγγιση χρησιμοποιείται το DataFrame API σε συνδυασμό με User-Defined Function (UDF). Αρχικά διαβάζονται τα δεδομένα εγκλημάτων από δύο αρχεία Parquet και ενώνονται σε ένα ενιαίο σύνολο. Στη συνέχεια, το DataFrame φιλτράρεται ώστε να περιλαμβάνει μόνο τα περιστατικά που περιέχουν τον όρο *"aggravated assault"* στην περιγραφή του εγκλήματος. Η κατηγοριοποίηση των θυμάτων σε ηλικιακές ομάδες πραγματοποιείται μέσω μιας δικής μας συνάρτησης Python, η οποία ενσωματώνεται στο DataFrame ως UDF. Η νέα στήλη ηλικιακής ομάδας προστίθεται με τη χρήση της `withColumn`. Έπειτα, εφαρμόζεται ομαδοποίηση και

καταμέτρηση περιστατικών ανά ηλικιακή ομάδα και τα αποτελέσματα ταξινομούνται κατά φθίνουσα σειρά. Τέλος, το αποτέλεσμα αποθηκεύεται σε μορφή CSV στο HDFS.

**Χρόνος εκτέλεσης:** `51s`

## 3.3 Με χρήση RDD

Σε αυτήν την προσέγγιση χρησιμοποιούμε το RDD API. Αρχικά διαβάζονται τα δεδομένα εγκλημάτων από δύο αρχεία Parquet και ενώνονται σε ένα ενιαίο σύνολο. Από αυτό το σύνολο, εξάγονται τα πεδία περιγραφής εγκλήματος και ηλικίας θύματος, τα οποία και μετατρέπονται σε RDD. Στη συνέχεια, φιλτράρονται μόνο εκείνα τα περιστατικά που περιέχουν τον όρο *"aggravated assault"* στην περιγραφή. Με τη χρήση της συνάρτησης `map`, κάθε εγγραφή μετασχηματίζεται σε ένα ζεύγος `(ηλικιακή_ομάδα, 1)`. Έπειτα, εφαρμόζεται η `reduceByKey` για την ομαδοποίηση και καταμέτρηση των περιστατικών ανά ηλικιακή ομάδα. Τέλος, τα αποτελέσματα ταξινομούνται κατά φθίνουσα σειρά ως προς το πλήθος και αποθηκεύονται σε μορφή CSV στο HDFS.

**Χρόνος εκτέλεσης:** `31s`

## 3.4 Αποτελέσματα και Σύγκριση

Τα αποτελέσματα και των τριών μεθόδων ήταν ισοδύναμα, παράγοντας τις ίδιες ηλικιακές κατηγορίες και πλήθος περιστατικών, και παρουσιάζονται στον Πίνακα 1. Ωστόσο, οι διαφορές στην απόδοση και την ευκολία ανάπτυξης ήταν αισθητές.

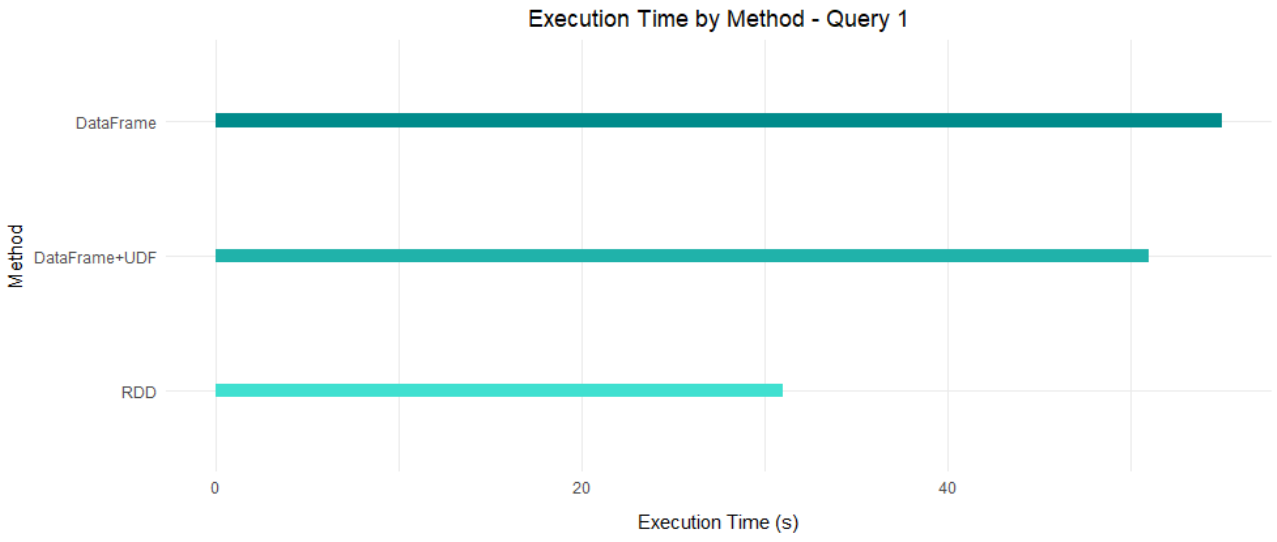
Η εκδοχή με καθαρό DataFrame API, αν και ίσως αναμενόταν να είναι η πιο αποδοτική λόγω του Catalyst, παρουσίασε τον μεγαλύτερο χρόνο εκτέλεσης. Αντίθετα, η εκδοχή με UDF, ενώ θεωρείται συνήθως λιγότερο αποδοτική, εμφάνισε βέλτιστη επίδοση.

Ενδεχομένως η υπεροχή του RDD να οφείλεται στο σχετικά μικρό μέγεθος των δεδομένων (της τάξης των 100mb: το ελέγξαμε με `hdfs dfs -du -h /user/chrysovalantilymperi/data/parquet/LA_Crime_Data_2020_2025`) που καθιστά πιο άμεση την εκτέλεση.

| **Age Group** | **Count** |
|---|---|
| Adults (25-64) | 122,727 |
| Young adults (18-24) | 34,036 |
| Children (<18) | 16,124 |
| Elderly (>64) | 6,082 |

Πίνακας 1: Κατανομή περιστατικών *aggravated assault* ανά ηλικιακή ομάδα

Ο χρόνος εκτέλεσης των τριών διαφορετικών προσεγγίσεων παρουσιάζεται στο Σχήμα 1.




Σχήμα 1: Χρόνος εκτέλεσης του Query 1 για κάθε διαφορετική μέθοδο που προτείνεται. Οι χρόνοι αντλήθηκαν από το UI του Spark History Server.

# 4 Query 2

Στο δεύτερο ερώτημα ζητείται να βρεθούν για κάθε έυος τα 3 Αστυνομικά Τμήματα με το υψηλότερο ποσοστό κλεισμένων (περατωμένων) υποθέσεων. Ως περατωμένες θεωρούνται οι υποθέσεις που δεν είναι ανοιχτές, δηλαδή αυτές που δέν έχουν κατάσταση άγνωστη ή σε εξέλιξη. Τα αποτελέσματα θα πρέπει να τυπωθούν σε σειρά αύξουσα ως προς το έτος και το ranking.

## 4.1 Με χρήση DataFrames

Στην πρώτη προσέγγιση χρησιμοποιείται το DataFrame API για να προσδιοριστούν, ανά έτος, τα τρία Αστυνομικά Τμήματα με το υψηλότερο ποσοστό περατωμένων υποθέσεων. Αρχικά, τα δεδομένα εγκλημάτων διαβάζονται από τα δύο αρχεία Parquet που μελετήσαμε και στο προηγούμενο ερώτημα και ενώνονται σε ένα ενιαίο σύνολο. Έπειτα, μετατρέπεται το πεδίο ημερομηνίας σε τύπο timestamp και εξάγεται το έτος διάπραξης κάθε περιστατικού. Ορίζεται ως περατωμένη κάθε υπόθεση της οποίας η περιγραφή κατάστασης (*Status Desc*) δεν είναι ούτε "UNK" ούτε "Invest Cont", και δημιουργείται αντίστοιχη binary στήλη. Στη συνέχεια, για κάθε συνδυασμό έτους και ονόματος τμήματος, υπολογίζεται το πλήθος συνολικών και περατωμένων υποθέσεων, καθώς και το αντίστοιχο ποσοστό. Με χρήση παραθύρου κατά έτος και ταξινόμηση κατά φθίνουσα τιμή ποσοστού, υπολογίζεται η κατάταξη κάθε τμήματος. Επιλέγονται τα τρία τμήματα με το υψηλότερο ποσοστό ανά έτος και τα αποτελέσματα ταξινομούνται κατά έτος και θέση στην κατάταξη. Τέλος, αποθηκεύονται σε μορφή CSV στο HDFS.

**Χρόνος εκτέλεσης:** `34s`

## 4.2 Με χρήση SQL API

Σε αυτήν την προσέγγιση γίνεται χρήση της SQL σύνταξης της Spark. Αρχικά, και πάλι τα δεδομένα διαβάζονται από δύο αρχεία Parquet και και ενώνονται σε ένα κοινό DataFrame. Δημιουργούνται επιπλέον στήλες timestamp και έτους βάσει της ημερομηνίας του περιστατικού. Το ενιαίο σύνολο δεδομένων καταχωρείται ως προσωρινός πίνακας (temp view) με όνομα `crimes`, ώστε να είναι προσβάσιμο μέσω SQL. Η SQL ερώτηση περιλαμβάνει δύο ενδιάμεσες φάσεις (CTEs): η πρώτη υπολογίζει το πλήθος συνολικών και περατωμένων υποθέσεων ανά έτος και αστυνομικό τμήμα, ενώ η δεύτερη υπολογίζει το ποσοστό περατωμένων υποθέσεων και την κατάταξη με βάση αυτό το ποσοστό. Στο τελικό αποτέλεσμα επιλέγονται τα τρία τμήματα με το υψηλότερο ποσοστό περατωμένων υποθέσεων για κάθε έτος. Η έξοδος ταξινομείται κατά έτος και θέση στην κατάταξη και αποθηκεύεται σε μορφή CSV στο HDFS.

**Χρόνος εκτέλεσης:** `37s`

## 4.3 Με χρήση RDD

Σε αυτήν την προσέγγιση γίνεται χρήση του RDD API. Αφού διαβαστούν και ενωθούν τα αρχεία, επιλέγονται τα απαραίτητα πεδία και κάθε εγγραφή μετασχηματίζεται ώστε να εξαχθεί το έτος του περιστατικού, το όνομα του αστυνομικού τμήματος και η πληροφορία για το αν η υπόθεση είναι περατωμένη. Περατωμένες θεωρούνται οι υποθέσεις των οποίων η περιγραφή κατάστασης δεν είναι ούτε *"UNK"* ούτε *"Invest Cont"*. Οι εγγραφές ομαδοποιούνται ανά έτος και τμήμα και για κάθε ομάδα υπολογίζεται το πλήθος συνολικών και περατωμένων υποθέσεων. Έπειτα, για κάθε έτος υπολογίζεται το ποσοστό περατωμένων υποθέσεων ανά τμήμα και επιλέγονται τα τρία τμήματα με το υψηλότερο ποσοστό. Τέλος, τα αποτελέσματα ταξινομούνται κατά έτος και κατάταξη, μετατρέπονται σε DataFrame και αποθηκεύονται σε μορφή CSV στο HDFS. Η υλοποίηση ακολουθεί το μοντέλο Map Reduce, περιλαμβάνοντας φάσεις map ( (k,v)=(year, (precinct, closed_case_rate)) και groupByKey.

**Χρόνος εκτέλεσης:** `66s`

## 4.4 Αποτελέσματα και Σύγκριση

Τα αποτελέσματα των τριών προσεγγίσεων παρουσιάζονται στον Πίνακα 2. Οσον αφορά τους χρόνους εκτέλεσης, παρατηρούμε ότι σε αντίθεση με το πρώτο ερώτημα, στο δεύτερο ερώτημα, η χρήση του DataFrame API ήταν η ταχύτερη (34 δευτερόλεπτα), ακολουθούμενη από την SQL εκδοχή (37 δευτερόλεπτα), ενώ η RDD προσέγγιση καθυστέρησε σημαντικά (66 δευτερόλεπτα). Η χειρότερη απόδοση των RDDs εδώ εξηγείται ενδεχομένως από την αυξημένη πολυπλοκότητα της επεξεργασίας: απαιτείται parsing ημερομηνιών, υπολογισμός ποσοστών, ταξινόμηση και εύρεση των κορυφαίων στοιχείων ανά έτος. Οι πράξεις αυτές απαιτούν πολλαπλά στάδια shuffling, για τα οποία οι RDDs δεν προσφέρουν εσωτερικές βελτιστοποιήσεις όπως συμβαίνει στα DataDrames. Οι χρονοι εκτέλεσης των τριών προσεγγίσεων παρουσιάζονται στο Σχήμα 2


Σχήμα 2: Χρόνος εκτέλεσης του Query 1 για κάθε διαφορετική μέθοδο που προτείνεται. Οι χρόνοι αντλήθηκαν από το UI του Spark History Server.

| Year | Precinct | Closed Case Rate (%) | # |
|---|---|---|---|
| 2010 | Rampart | 32.85 | 1 |
| 2010 | Olympic | 31.52 | 2 |
| 2010 | Harbor | 29.36 | 3 |
| 2011 | Olympic | 35.04 | 1 |
| 2011 | Rampart | 32.50 | 2 |
| 2011 | Harbor | 28.51 | 3 |
| 2012 | Olympic | 34.33 | 1 |
| 2012 | Rampart | 32.46 | 2 |
| 2012 | Harbor | 29.51 | 3 |
| 2013 | Olympic | 33.58 | 1 |
| 2013 | Rampart | 32.12 | 2 |
| 2013 | Harbor | 29.72 | 3 |
| 2014 | Southeast | 66.67 | 1 |
| 2014 | Topanga | 50.00 | 2 |
| 2014 | Mission | 33.33 | 3 |
| 2015 | Van Nuys | 32.27 | 1 |
| 2015 | Mission | 30.47 | 2 |
| 2015 | West Valley | 30.34 | 3 |
| 2016 | Van Nuys | 31.95 | 1 |
| 2016 | West Valley | 30.96 | 2 |
| 2016 | Foothill | 29.92 | 3 |
| 2017 | Van Nuys | 32.06 | 1 |
| 2017 | Mission | 31.07 | 2 |
| 2017 | Foothill | 30.47 | 3 |
| 2018 | Foothill | 30.73 | 1 |
| 2018 | Mission | 30.72 | 2 |
| 2018 | Van Nuys | 28.92 | 3 |
| 2019 | Mission | 30.72 | 1 |
| 2019 | West Valley | 30.60 | 2 |
| 2019 | N Hollywood | 29.23 | 3 |
| 2020 | West Valley | 30.76 | 1 |
| 2020 | Mission | 30.16 | 2 |
| 2020 | Harbor | 29.70 | 3 |
| 2021 | Mission | 30.31 | 1 |
| 2021 | West Valley | 29.03 | 2 |
| 2021 | Foothill | 28.02 | 3 |
| 2022 | West Valley | 26.66 | 1 |
| 2022 | Harbor | 26.39 | 2 |
| 2022 | Topanga | 26.31 | 3 |
| 2023 | Topanga | 26.95 | 1 |
| 2023 | Foothill | 26.91 | 2 |
| 2023 | Mission | 25.99 | 3 |
| 2024 | N Hollywood | 20.15 | 1 |
| 2024 | Foothill | 18.06 | 2 |
| 2024 | 77th Street | 17.73 | 3 |
| 2025 | Central | 100.00 | 1 |
| 2025 | Topanga | 20.00 | 2 |
| 2025 | Foothill | 14.29 | 3 |

Πίνακας 2: Ποσοστό περατωμένων υποθέσεων ανά έτος, τμήμα και κατάταξη.

# 5 Query 3

Στο ερώτημα αυτό εργαζόμαστε με το Median Household Income by Zip Code (Los Angeles County) dataset, που περιλαμβάνει το μέσο εισόδημα ανά νοικοκυριό και ταχυδρομικό κώδικα (ZIP Code) στην Κομητεία του Los Angeles το 2015, καθώς και την απογραφή του 2010 για την ίδια κομητεία. Καλούμαστε να , να υπολογίσουμε για κάθε ZipCode του Los Angeles το μέσο ετήσιο εισόδημα ανά άτομο.

## 5.1 Με χρήση DataFrames

### 5.1.1 Με Parquet

Σε αυτή την προσέγγιση χρησιμοποιούνται τα δύο σύνολα δεδομένων σε μορφή Parquet: η απογραφή του πληθυσμού του 2010 και τα στοιχεία εισοδήματος του 2015 για την Κομητεία του Los Angeles, ανά ταχυδρομικό κώδικα (ZIP Code). Αρχικά, καθαρίζεται η στήλη του εκτιμώμενου εισοδήματος ώστε να αφαιρεθούν σύμβολα νομισμάτων και το πεδίο μετατρέπεται σε αριθμητική μορφή. Έπειτα, τα δύο σύνολα δεδομένων ενοποιούνται μέσω inner join στο κοινό πεδίο "Zip Code". Ο υπολογισμός του μέσου εισοδήματος ανά άτομο γίνεται με διαίρεση του εισοδήματος ανά νοικοκυριό με το μέσο μέγεθος νοικοκυριού:

$$\text{Average Income per Person} = \frac{\text{Estimated Median Income}}{\text{Average Household Size}}$$

Το αποτέλεσμα περιλαμβάνει τον ταχυδρομικό κώδικα και το υπολογισμένο μέσο εισόδημα ανά άτομο και αποθηκεύεται σε μορφή CSV στο HDFS.

**Χρόνος εκτέλεσης**: 31s

### 5.1.2 Με CSV

Δεν αλλάζει τίποτα πέρα από τον τρόπο που διαβάζονται τα αρχεία. Τώρα τα τραβάμε από το csv που ήταν αρχικά αποθηκευμένο στο HDFS.

**Χρόνος εκτέλεσης**: 39s

## 5.2 Με χρήση RDD

Σε αυτήν την προσέγγιση γίνεται χρήση του RDD API. Τα δεδομένα εισοδήματος και απογραφής διαβάζονται αυτή τη φορά από αρχεία Parquet και μετατρέπονται σε RDDs, όπου κάθε εγγραφή αντιστοιχεί σε ένα ζεύγος `(Zip Code, τιμή)`. Οι δύο RDD ενώνονται με βάση τον ταχυδρομικό κώδικα, και για κάθε κοινό Zip Code δημιουργείται ένα ζεύγος που περιλαμβάνει το εκτιμώμενο μέσο εισόδημα ανά νοικοκυριό και το μέσο μέγεθος νοικοκυριού. Στη συνέχεια, για κάθε εγγραφή, υπολογίζεται το μέσο εισόδημα ανά άτομο ως λόγος εισοδήματος προς μέγεθος νοικοκυριού, και απορρίπτονται οι εγγραφές με μη έγκυρες τιμές. Το τελικό αποτέλεσμα μετατρέπεται σε DataFrame και αποθηκεύεται σε μορφή CSV στο HDFS.

Η υλοποίηση ακολουθεί την τυπική λογική του υπολογιστικού μοντέλου Map Reduce: αρχικά εφαρμόζεται map για τη μετατροπή των δεδομένων σε ζεύγη κλειδιού-τιμής, στη συνέχεια join για τη σύνδεση των δύο RDD βάσει κοινού κλειδιού (Zip Code), και τέλος ο υπολογισμός του αποτελέσματος ανά εγγραφή.

**Χρόνος εκτέλεσης**: 34s

## 5.3 Αποτελέσματα και Σύγκριση

Ενδεικτικά, οι 5 πρώτες και 5 τελευταίες γραμμές του πίνακα των αποτελεσμάτων παρουσιάζονται στον Πίνακα 3.

| **Zip Code** | **Estimated Avg Income ($)** |
|---|---|
| 90001 | 7,701.59 |
| 90002 | 6,975.46 |
| 90003 | 7,299.76 |
| 90004 | 14,876.19 |
| 90005 | 12,456.80 |
| ⋮ | |
| 93551 | 26,567.92 |
| 93552 | 13,774.05 |
| 93553 | 20,091.22 |
| 93563 | 22,694.47 |
| 93591 | 11,118.53 |

Πίνακας 3: Εκτιμώμενο μέσο εισόδημα ανά ΤΚ (ενδεικτικά καταγράφονται οι πρώτες και τελευταίες 5 εγγραφές)

Όσον αφορά τον χρόνο εκτέλεσης, αυτός παρουσιάζεται για τις τρεις διαφορετικές προσεγγίσεις στο Σχήμα 3. Παρατηρούμε ότι, η ταχύτερη εκτέλεση επιτυγχάνεται με χρήση του DataFrame API σε συνδυασμό με δεδομένα

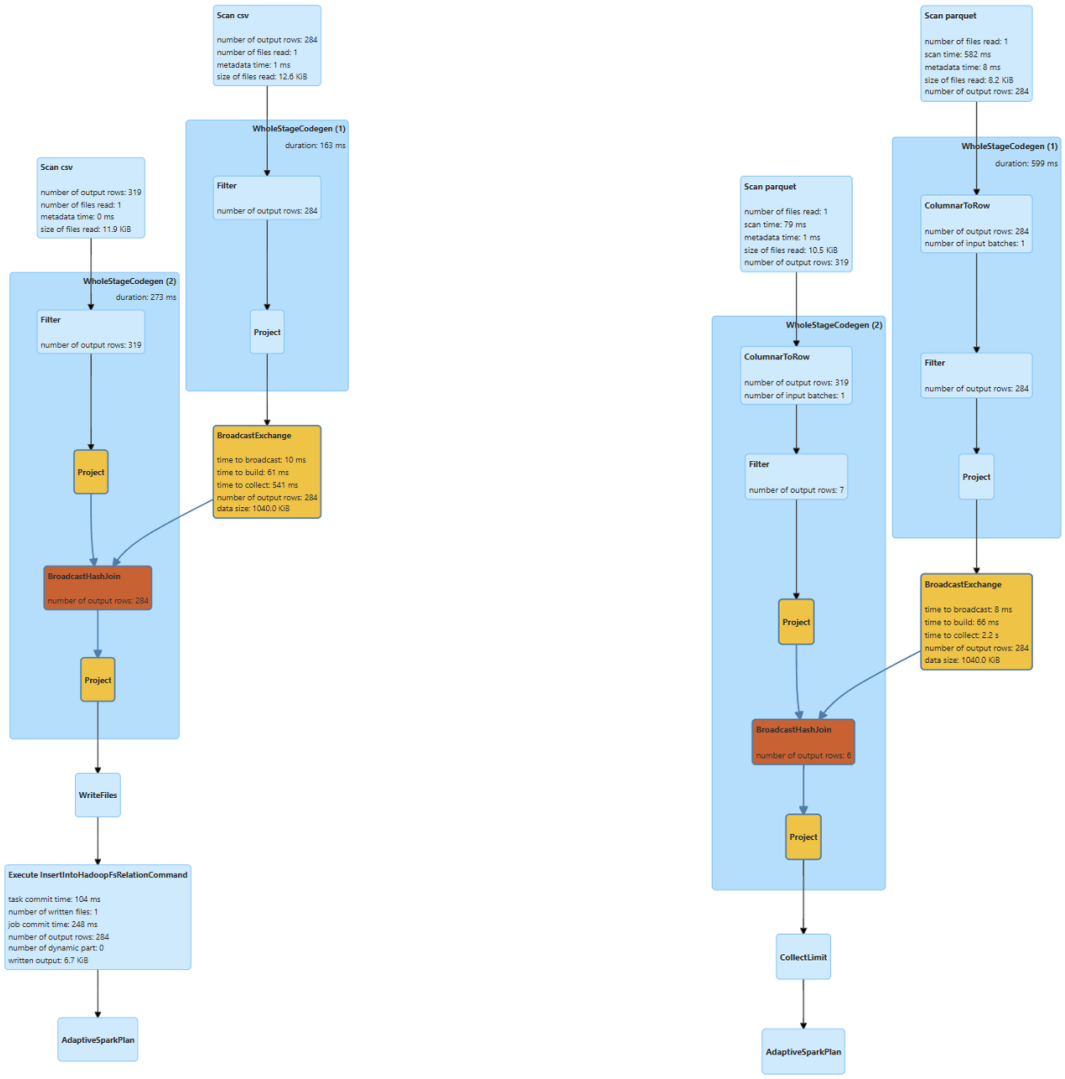

Σχήμα 3: Χρόνος εκτέλεσης του Query 3 για κάθε δια ορετική μέθοδο που προτείνεται. Οι χρόνοι αντλήθηκαν από το UI του Spark History Server

μορφής Parquet.

Η υπεροχή της μορφής Parquet οφείλεται στα εγγενή της πλεονεκτήματα, καθώς βελτιστοποιούν το I/O και υποστηρίζουν predicate pushdown που επιταχύνουν σημαντικά την ανάγνωση και επεξεργασία των δεδομένων. Αντίθετα, η μορφή CSV είναι πιο αργή λόγω της απουσίας πληροφοριών τύπων δεδομένων και της ανάγκης για parsing κατά την ανάγνωση.

Η υλοποίηση με RDD παρότι δεν αξιοποιεί τις βελτιστοποιήσεις του Catalyst, είχε ικανοποιητική απόδοση, μάλλον λόγω της απλότητας των πράξεων (join και αριθμητικός υπολογισμός ανά εγγραφή).

Ακόμα μπορούμε να δούμε το Physical Execution Plan (Σχήμα 4 για κάθε query μέσω του Spark History UI. Παρατηρούμε ότι ο Catalyst επέλεξε να πραγματοποιήσει Broadcast Join, κάτι που ευθυγραμμίζεται με την σχετική θεωρία. Το μικρό μέγεθος των αρχείων που διαβάζονται (της τάξης των λίγων mb) καθιστούν ιδανική την επιλογή του Broadcast Join.




Σχήμα 4: Physical Execution Plan για το query 3. Αριστερά: φόρτωση αρχείου ως csv. Δεξιά: φόρτωση αρχείου ως parquet.

# 6 Query 4

Για το Query 4 εργαζόμαστε με με τα δύο σύνολα δεδομένων για τα εγκλήματα στο Los Angeles από το 2010 ως το 2019 και από το 2020 ως το 2025, καθώς και με το σύνολο δεδομένων που αφορά τα στοιχεία των αστυνομικών τμημάτων της ίδιας κομητείας. Καλούμαστε να υπολογίσουμε, ανά αστυνομικό τμήμα, τον αριθμό εγκλημάτων που έλαβαν χώρα πλησιέστερα σε αυτό με εμπλοκή όπλων (πυροβόλων ή όχι), καθώς και τη μέση απόστασή του από τις τοποθεσίες όπου σημειώθηκαν τα συγκεκριμένα περιστατικά. Τα αποτελέσματα πρέπει να εμφανιστούν ταξινομημένα κατά αριθμό περιστατικών, με φθίνουσα σειρά.

## 6.1 Με χρήση Dataframes

Στην υλοποίηση του Query 4 χρησιμοποιήθηκε αποκλειστικά το Spark DataFrame API. Πρώτα φορτώθηκαν τα δύο αρχεία εγκλημάτων και ενοποιήθηκαν, ενώ το λεξικό MO Codes διαβάστηκε ως κείμενο, διαχωρίστηκε σε κωδικό{περιγραφή και συνδέθηκε (inner join) με το σύνολο εγκλημάτων. Έτσι κρατήσαμε μόνο τις εγγραφές των οποίων η περιγραφή MO περιέχει τους όρους «weapon» ή «gun». Στη συνέχεια απέκλεισα τα περιστατικά που αναφέροντυαι στο Null Island. Το DataFrame των αστυνομικών τμημάτων ενώθηκε (cartesian join) με τα φιλτραρισμένα εγκλήματα. Για κάθε ζεύγος υπολογίστηκε η σφαιρική (Haversine) απόσταση σε χιλιόμετρα,

$$d = 2R \arctan\left(\frac{\sqrt{\sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos\varphi_1 \cos\varphi_2 \sin^2\left(\frac{\Delta\lambda}{2}\right)}}{\sqrt{1 - \sin^2\left(\frac{\Delta\varphi}{2}\right) - \cos\varphi_1 \cos\varphi_2 \sin^2\left(\frac{\Delta\lambda}{2}\right)}}\right),$$

όπου $R = 6371km$, $\varphi_1, \varphi_2$ είναι τα γεωγραφικά πλάτη και $\Delta\varphi, \Delta\lambda$ οι διαφορές πλάτους και μήκους (σε ακτίνια). Οι στήλες `X`, `Y` του αρχείου σταθμών είναι ήδη γεωγραφικές συντεταγμένες, όπως και στο αρχείο των εγκλημάτων.

Στη συνέχεια, με παράθυρο `row_number` επιλέχθηκε για κάθε κωδικό συμβάντος ο σταθμός με τη μικρότερη απόσταση και, τέλος, υπολογίσθηκαν ανά τμήμα ο συνολικός αριθμός περιστατικών και η μέση απόσταση, ενώ το αποτέλεσμα ταξινομήθηκε με φθίνουσα σειρά ως προς το πλήθος.

## 6.2 Αποτελέσματα

Στον Πίνακα 4 παρουσιάζονται τα αποτελέσματα του query.

| **Division** | **Incident Count** | **Average Distance (km)** |
|---|---|---|
| CENTRAL | 4,492 | 0.75 |
| PACIFIC | 3,186 | 3.85 |
| VAN NUYS | 2,981 | 2.79 |
| HOLLYWOOD | 2,750 | 1.83 |
| TOPANGA | 2,010 | 3.08 |
| NORTH HOLLYWOOD | 1,862 | 2.46 |
| MISSION | 1,823 | 3.86 |
| WILSHIRE | 1,812 | 2.31 |
| FOOTHILL | 1,545 | 3.93 |
| RAMPART | 1,540 | 1.50 |
| SOUTHEAST | 1,526 | 2.17 |
| WEST VALLEY | 1,391 | 2.99 |
| NORTHEAST | 1,111 | 3.80 |
| HARBOR | 941 | 3.70 |
| WEST LOS ANGELES | 907 | 2.64 |
| HOLLENBECK | 870 | 2.58 |
| NEWTON | 853 | 1.59 |
| SOUTHWEST | 685 | 2.02 |
| 77TH STREET | 662 | 1.68 |
| DEVONSHIRE | 628 | 2.83 |
| OLYMPIC | 430 | 1.79 |

Πίνακας 4: Πλήθος περιστατικών και μέση απόσταση ανά αστυνομικό τμήμα.

## 6.3 Οριζόντια και Κάθετη κλιμάκωση των πόρων

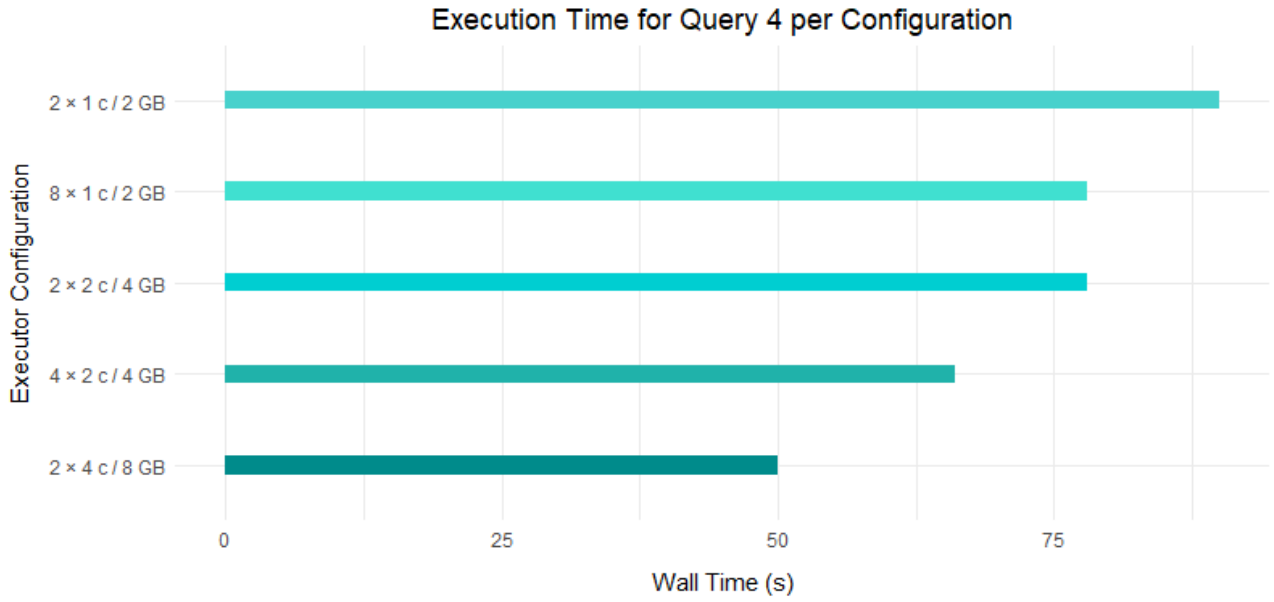
Για να μελετηθεί η επίδραση της κατανομής πόρων στον χρόνο εκτέλεσης του query, το ίδιο εκτελέστηκε πέντε φορές, με συνολικό ανώτατο όριο 8 πυρήνων και 16 GB μνήμης. Στην οριζόντια κλιμάκωση ($2 \times 4\,\text{c}/8\,\text{GB}$, $4 \times 2\,\text{c}/4\,\text{GB}$, $8 \times 1\,\text{c}/2\,\text{GB}$) ο αριθμός των executors αυξήθηκε ενώ οι διαθέσιμοι πυρήνες και η μνήμη ανά executor μειώθηκαν αναλόγως. Αντίστροφα,, στην κάθετη κλιμάκωση κρατήθηκε σταθερός ο αριθμός των executors (2) και αυξήθηκαν οι πόροι (1 c / 2 GB → 2 c / 4 GB → 4 c / 8 GB).

Οι χρόνοι εκτέλεσης για κάθε configuration παρουυσιάονται στον Πίνακα 5 και οπτικοποιούνται στο Σχήμα 5.

| **Διάταξη executors** | **Συνολικοί πόροι** | **Διάρκεια** |
|---|---|---|
| $2 \times 4$ c / 8 GB | 8 c, 16 GB | 50 s |
| $4 \times 2$ c / 4 GB | 8 c, 16 GB | 1.1 min |
| $8 \times 1$ c / 2 GB | 8 c, 16 GB | 1.3 min |
| $2 \times 2$ c / 4 GB | 4 c, 8 GB | 1.3 min |
| $2 \times 1$ c / 2 GB | 2 c, 4 GB | 1.5 min |

Πίνακας 5: Χρόνος εκτέλεσης του Query 4 στις πέντε διατάξεις

Παρατηρούμε ότι το βέλτιστο configuration είναι το σχήμα με δύο executors, τέσσερις πυρήνες και οκτώ GB μνήμης ο καθένας. Με μόνο δύο εκτελεστές, ο Spark στέλνει το μικρό αρχείο των αστυνομικών τμημάτων (21 γραμμές) δύο φορές αντί για τέσσερις ή οκτώ, οπότε ξοδεύει λιγότερο χρόνο σε μεταφορές και αρχικοποιήσεις. Ταυτόχρονα, κάθε εκτελεστής διαθέτει τέσσερις πυρήνες, αρκετούς για να τρέχουν παράλληλα οι πράξεις (υπολογισμός απόστασης, ταξινόμηση, ομαδοποίηση) χωρίς να «στριμώχνονται» σε μικρό χώρο μνήμης. Αν μοιράσουμε τους ίδιους 8 πυρήνες σε περισσότερους, μικρότερους executors, κερδίζουμε ελάχιστη επιπλέον ταχύτητα αλλά πληρώνουμε παραπάνω «οργανωτικά» έξοδα: περισσότερες διεργασίες, περισσότερες αντιγραφές δεδομένων, περισσότερη



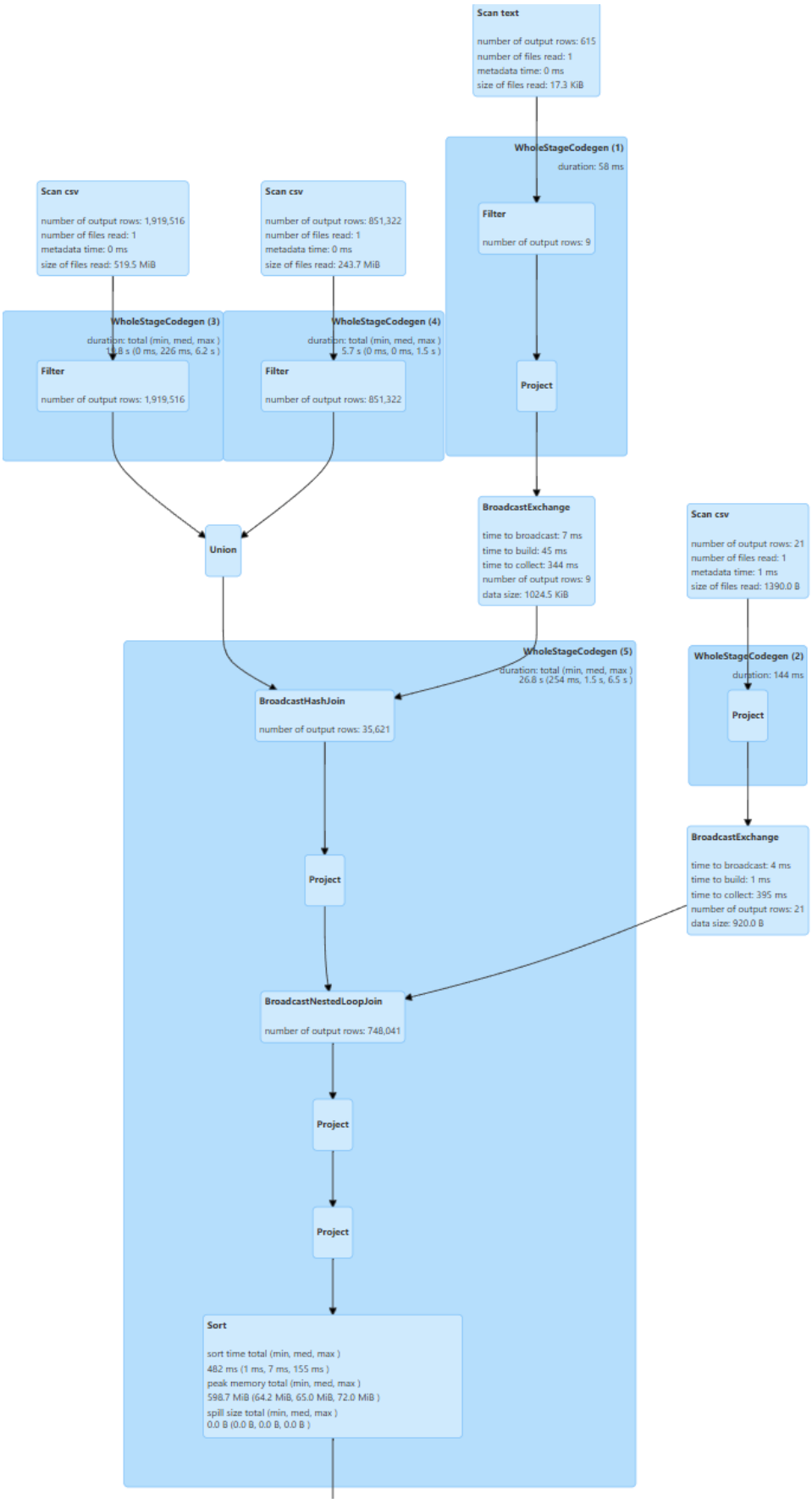

Σχήμα 5: Χρόνος εκτέλεσης του Query 4 στις πέντε διατάξεις

διαχείριση μνήμης. Έτσι, η λύση με λίγους αλλά πιο «ισχυρούς» executors ισορροπεί ιδανικά ανάμεσα σε παράλληλη επεξεργασία και χαμηλό overhead.

## 6.4 Physical Execution Plan

Όσον αφορά την επιλογή του join που έκανε ο Catalyst, μπορούμε να δούμε το Physical Execution Plan (Σχήμα 6 μέσω του Spark History UI. Τα δύο διαδοχικά joins που επιλέχτηκαν είναι τα εξής:

- **Πρώτο join.** Ο πίνακας των MO Codes περιέχει μόλις 9 γραμμές, επομένως ο Spark τον μεταδίδει αυτούσιο σε κάθε εκτελεστή (broadcast) και στη συνέχεια ταιριάζει απευθείας τον κωδικό `Mocodes` με τον μεγάλο πίνακα εγκλημάτων. Η μετάδοση του τόσο μικρού πίνακα εξοικονομεί χρόνο, επειδή δεν απαιτούνται επιπλέον μεταφορές δεδομένων στο δίκτυο.
- **Δεύτερο join.** Ο πίνακας των αστυνομικών τμημάτων έχει 21 γραμμές και, για τον ίδιο λόγο, μεταδίδεται επίσης ως broadcast. Εδώ όμως δεν υπάρχει κοινό κλειδί: χρειάζονται όλοι οι συνδυασμοί (εγκλημα × τμήμα) για τον υπολογισμό απόστασης. Ο Spark επιλέγει τον απλούστερο τρόπο, δηλαδή έναν nested-loop πάνω στο ήδη μεταδομένο μικρό σύνολο, που είναι ταχύτερος από οποιονδήποτε περίπλοκο αλγόριθμο όταν το δεύτερο input είναι τόσο μικρό.




Σχήμα 6: Physical Execution Plan για το query 4.