Συστήματα Διαχείρισης Δεδομένων Μεγάλου Όγκου
Εργαστηριακή Άσκηση 2023/24

Όνομα	**Επώνυμο**	**ΑΜ**
Χρυσαυγή	Πατέλη	1084513
Μηλτιάδης	Μαντές	1084661

Βεβαιώνω ότι είμαι συγγραφέας της παρούσας εργασίας και ότι έχω αναφέρει ή παραπέμψει σε αυτήν, ρητά και συγκεκριμένα, όλες τις πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, προτάσεων ή λέξεων, είτε αυτές μεταφέρονται επακριβώς (στο πρωτότυπο ή μεταφρασμένες) είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για το συγκεκριμένο μάθημα/σεμινάριο/πρόγραμμα σπουδών.

Έχω ενημερωθεί ότι σύμφωνα με τον εσωτερικό κανονισμό λειτουργίας του Πανεπιστημίου Πατρών άρθρο 50§6, τυχόν προσπάθεια αντιγραφής ή εν γένει φαλκίδευσης της εξεταστικής και εκπαιδευτικής διαδικασίας από οιονδήποτε εξεταζόμενο, πέραν του μηδενισμού, συνιστά βαρύ πειθαρχικό παράπτωμα.

Υπογραφή

ΜΗΛΤΙΑΔΗΣ ΜΑΝΤΕΣ

_20_ / **_09_** / 2024

Υπογραφή

ΧΡΥΣΑΥΓΗ ΠΑΤΕΛΗ

_20_ / **_09_** / 2024

Συνημμένα αρχεία κώδικα

Μαζί με την παρούσα αναφορά υποβάλλουμε τα παρακάτω αρχεία κώδικα:

Αρχείο	**Αφορά το ερώτημα**	**Περιγραφή/Σχόλιο**
create-data.py producer.py consumer.py	1ο	Περιέχει όλα τα ερωτήματα για το ερ. 1
spark.py	2ο	Περιέχει όλα τα ερωτήματα για το ερ. 2
spark-mongo.py queries.py	3ο	Περιέχει όλα τα ερωτήματα για το ερ. 3

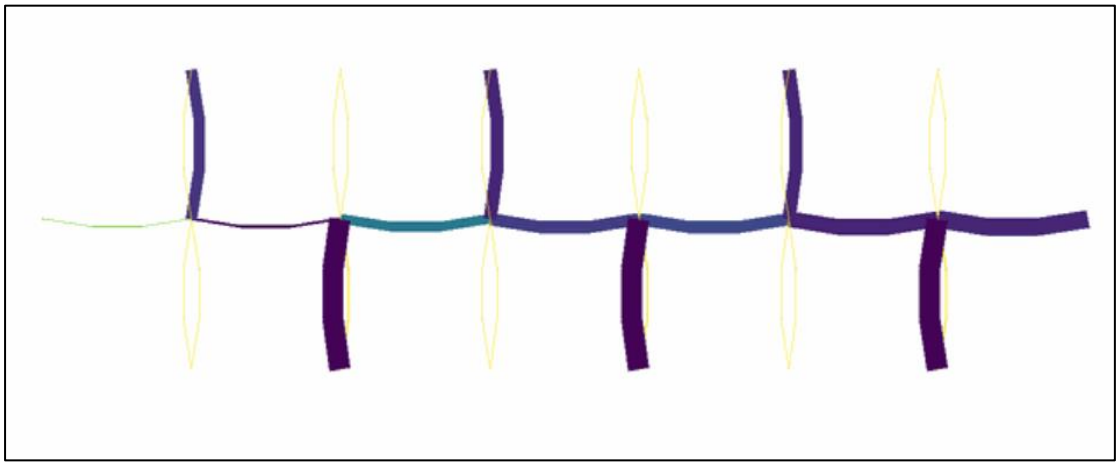
Τεχνικά χαρακτηριστικά περιβάλλοντος λειτουργίας

Χαρακτηριστικό	**Τιμή**
CPU model	Intel Core i3-5005U
CPU clock speed	2GHz
Physical CPU cores	1
Logical CPU cores	128
RAM	2MB
Secondary Storage Type	HDD

Ερώτημα 1: Παραγωγή δεδομένων

Για την παραγωγή του test dataset τρέχουμε το αρχείο **create-data.py** και τα δεδομένα που παράγονται από τον εξομοιωτή **UXSIM** εξάγονται στο αρχείο **vehicle_data.csv** Πιο κάτω αναλύουμε τον κώδικα όσον αφορά την υλοποίηση του **create-data.py**.

Αρχικά, δημιουργούμε ένα στιγμιότυπο **W** τύπου **World** με τις ζητούμενες παραμέτρους. Στη συνέχεια, κατασκευάζουμε το οδικό δίκτυο προς προσομοίωση, το οποίο φαίνεται στην εικόνα παρακάτω:



Εικόνα 1: Στιγμιότυπο δικτύου

Οι κόμβοι του δικτύου ορίζονται μέσω της μεθόδου **addNode( )** και κάθε κόμβος έχει:

- **Συντεταγμένες** (π.χ. 1, 0 για τον κόμβο I1).
- **Σηματοδότες** (χρονικά διαστήματα εναλλαγής σηματοδότη).

Οι κόμβοι περιλαμβάνουν τις διασταυρώσεις, τα σημεία εισόδου/εξόδου και τους βόρειους/νότιους κόμβους. Οι σύνδεσμοι μεταξύ των κόμβων ορίζονται σε λίστες ζευγών κόμβων, όπως:

- **Σύνδεσμοι** από τα δυτικά προς τα ανατολικά (π.χ. από W1 προς I1, από I1 προς I2).
- **Σύνδεσμοι** από βόρεια προς νότια και αντίστροφα.

Οι σύνδεσμοι προστίθενται με τη μέθοδο **addLink( )**, η οποία καθορίζει:

- Το **μήκος του δρόμου** (500 μέτρα).
- Την **ελεύθερη ταχύτητα** κυκλοφορίας (π.χ. 50 km/h).
- Την **πυκνότητα συμφόρησης** και τον αριθμό λωρίδων.

Κάθε 30 δευτερόλεπτα προστίθενται τυχαίες απαιτήσεις κυκλοφορίας:

- Από τους βόρειους κόμβους προς τους νότιους (π.χ. από N1 προς S1).
- Από διάφορους κόμβους προς το W1.

Για κάθε χρονικό διάστημα, ορίζεται μια τυχαία ζήτηση κυκλοφορίας π.χ. πόσα οχήματα θα κινηθούν ανάμεσα στους κόμβους και οι απαιτήσεις προστίθενται μέσω της μεθόδου **adddemand( )**.

Τέλος, εκτελείται η εξομοίωση στο αντικείμενο **W** για το ορισμένο χρονικό διάστημα, το οποίο είναι 3600 δευτερόλεπτα ή 1 ώρα. Το αρχείο **vehicle_data.csv** που παράγεται μετά το πέρας της εξομοίωσης διαβάζεται στη συνέχεια από το script του Kafka producer για την αποστολή των δεδομένων.

Για την εγκατάσταση της υπηρεσίας Kafka broker εγκαθιστούμε την έκδοση **2.13-3.8.0** για λειτουργικό σύστημα Linux (σε VM debian). Αυτό γίνεται μέσω της εντολής **tar -xzf kafka_2.13-3.8.0.tgz** η οποία κάνει extract το αρχείο προς download του Kafka. Στη συνέχεια, με την εντολή **cd kafka_2.13-3.8.0** μεταβαίνουμε στον αντίστοιχο φάκελο του Kafka και προκειμένου να ενεργοποιηθεί η υπηρεσία πρέπει αρχικά να εγκατασταθεί το Zookeeper μέσω της **εντολής bin/zookeeper-server-start.sh config/zookeeper.properties**. Έπειτα, ξεκινάμε το τρέξιμο του Kafka με την εντολή **bin/kafka-server-start.sh config/server.properties**. Εφόσον βεβαιωθούμε ότι τρέχει σωστά, δημιουργούμε το topic **vehicle_positions** με την εντολή **bin/kafka-topics.sh --create --topic vehicle_positions --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1**. Στο παρακάτω στιγμιότυπο βλέπουμε ότι το topic έχει δημιουργηθεί επιτυχώς:

```
(myenv) debian@debian:~/project/kafka_2.13-3.8.0$ bin/kafka-topics.sh --list --bootstrap-server localhost:9092
__consumer_offsets
vehicle_positions
```

Εικόνα 2: Επιτυχής εγκατάσταση Kafka και δημιουργία topic

Για να τρέξουμε τα δύο scripts **producer.py** και **consumer.py** πρέπει επίσης να κατεβάσουμε τη βιβλιοθήκη **kafka-python**. Αφού εγκατασταθεί σωστά η βιβλιοθήκη τρέχουμε τα δύο scripts για τα οποία αναλύουμε την υλοποίηση παρακάτω.

producer.py:

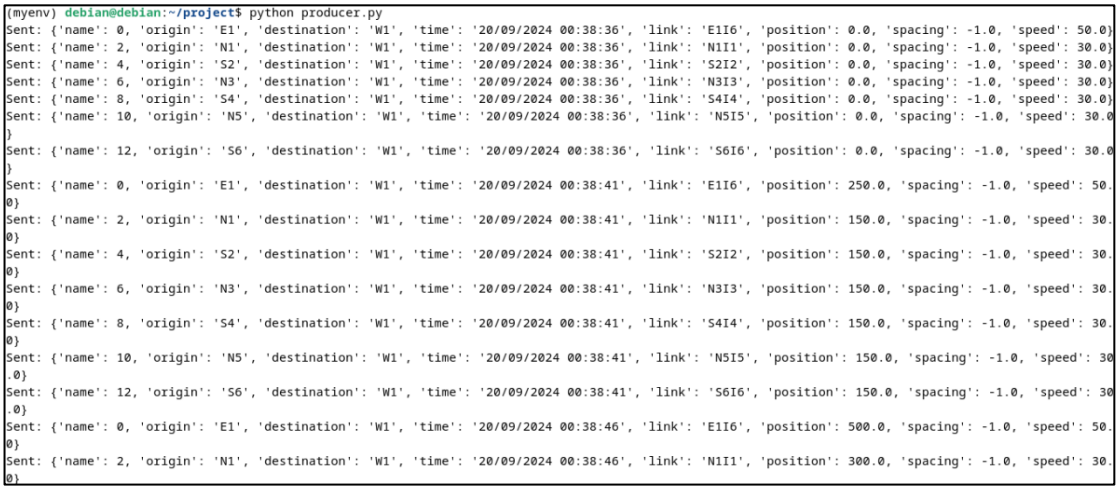
Αρχικά το script διαβάζει το αρχείο CSV και το φορτώνει σε ένα **DataFrame** της βιβλιοθήκης **Pandas**. Στη συνέχεια, συνδεόμαστε με τον Kafka broker που τρέχει τοπικά στον υπολογιστή μας στην θύρα 9092 μέσω του **KafkaProducer**, ο οποίος είναι υπεύθυνος για την αποστολή των μηνυμάτων στο topic **vehicle_positions**. Χρησιμοποιείται και ένας **value_serializer** που καθορίζει τη μορφοποίηση των μηνυμάτων σε JSON μορφή (χρησιμοποιώντας **json.dumps** για τη μετατροπή σε string και encode για τη μετατροπή σε bytes) προτού

σταλούν στο Kafka. Το **start_time** αποθηκεύει την τρέχουσα χρονική στιγμή που ξεκινά η προσομοίωση και ορίζεται ένα διάστημα αναμονής 5 δευτερολέπτων για την αποστολή μηνυμάτων. Επίσης, η συνάρτηση **create_json_object( )** δημιουργεί το μήνυμα που θα σταλεί στον Kafka broker για κάθε γραμμή του CSV μετατρέποντάς το στην επιθυμητή μορφή. Πιο συγκεκριμένα, για κάθε γραμμή, υπολογίζουμε την ώρα αποστολής βασισμένη στην εκκίνηση της προσομοίωσης και την τιμή του βήματος **k**, το οποίο παίρνει τιμές 0,5,10,15,… για χρονικό διάστημα μιας ώρας που κρατάει η εξομοίωση. Στη συνέχεια, επιλέγονται οι γραμμές του CSV όπου η τιμή της στήλης **t** αντιστοιχεί στο τρέχον **k** (δηλαδή, τα οχήματα που υπάρχουν στον χρόνο αυτό στο δίκτυο). Αν η τιμή της στήλης **link** είναι **"waiting at origin node"**, τότε παραλείπεται η αποστολή αυτού του μηνύματος. Τέλος, υπολογίζουμε το χρόνο που πρέπει να περιμένει πριν στείλει το μήνυμα με τη συνάρτηση **time.sleep( )** και μόλις έρθει η ώρα να σταλεί το μήνυμα, δημιουργείται το JSON αντικείμενο και αποστέλλεται στον Kafka broker. Θέτουμε ως χρόνο αναμονής μέχρι να σταλθεί το επόμενο μήνυμα το 1 sec.

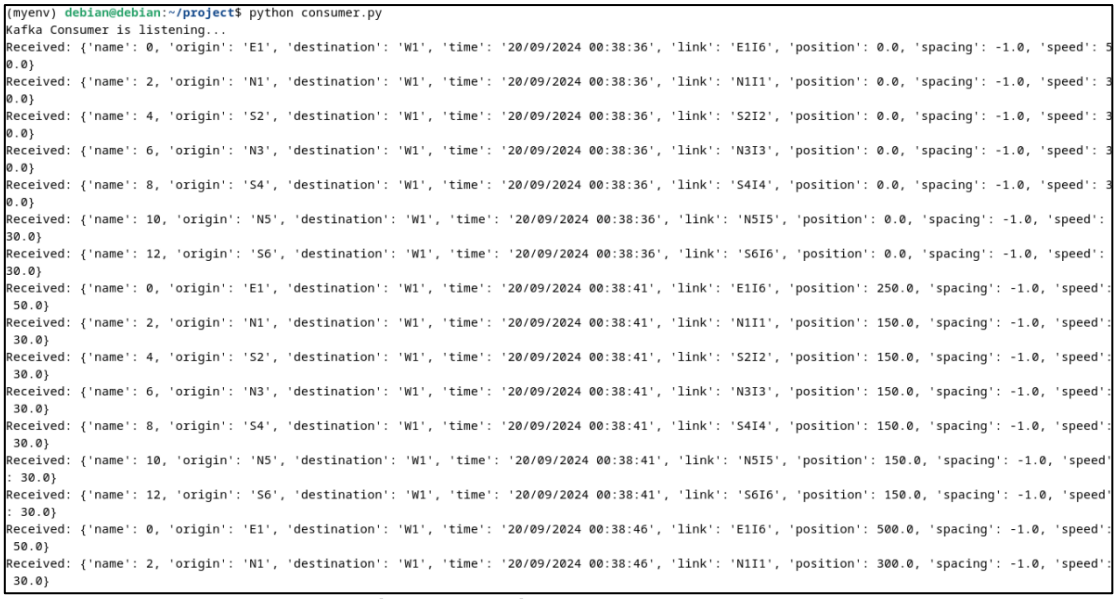
consumer.py:

Όμοια συνδεόμαστε στον ίδιο Kafka broker με πριν και το αντικείμενο **KafkaConsumer** που δημιουργείται καταναλώνει μηνύματα από το συγκεκριμένο Kafka topic. Ο Kafka consumer ξεκινά να διαβάζει τα μηνύματα από το topic χρησιμοποιώντας έναν βρόχο for όπου για κάθε μήνυμα που λαμβάνεται αποθηκεύεται ως ένα **Kafka Message** αντικείμενο, από το οποίο ανακτάται στη συνέχεια το περιεχόμενο του εκάστοτε μηνύματος. Το περιεχόμενο του μηνύματος που στάλθηκε στο topic αποσυμπιέζεται σε JSON μορφή μέσω του **value_deserializer**. Τέλος, εάν ο χρήστης διακόψει την εκτέλεση του script χειροκίνητα εμφανίζεται ένα μήνυμα ότι η εκτέλεση διακόπηκε.

Πιο κάτω παρατίθενται στιγμιότυπα από την ορθή λειτουργία και των δύο:



Εικόνα 3: Στιγμιότυπο Kafka producer



Εικόνα 4: Στιγμιότυπο Kafka consumer

Ερώτημα 2: Κατανάλωση και επεξεργασία με Spark

Για τη λειτουργία του Apache Spark χρειάζεται πρώτα να γίνει εγκατάσταση της JAVA. Αυτό πραγματοποιείται με τις εντολές **sudo apt update** και **sudo apt install openjdk-11-jdk**. Έπειτα, κατεβάζουμε το αρχείο του Spark από την αντίστοιχη ιστοσελίδα και με την εντολή **tar -xzf spark-3.4.0-bin-hadoop3.tgz** το κάνουμε extract στον φάκελο που επιθυμούμε. Πρέπει επίσης, να κάνουμε setup και τις αντίστοιχες μεταβλητές περιβάλλοντος για το Spark στο σύστημά μας, το οποίο το επιτυγχάνουμε με τις εντολές **export SPARK_HOME=/opt/spark** και **export PATH=$PATH:$SPARK_HOME/bin**. Για να εφαρμοστούν αυτές οι αλλαγές θα πρέπει να εκτελεστεί και η εντολή **source ~/.bashrc** στη συνέχεια. Τέλος, όσον αφορά το python script, θα πρέπει να εγκατασταθεί και η βιβλιοθήκη **pyspark**.

spark.py:

Ξεκινάμε αρχικοποιώντας ένα Spark Session, το οποίο είναι απαραίτητο για τη χρήση του Spark με όνομα **"KafkaSparkProcessing".** Επίσης, πρέπει να οριστεί και το σχήμα (schema) των δεδομένων που αναμένονται από το Kafka από τα JSON αντικείμενα. Το κάθε μήνυμα θα έχει τα πεδία: **name**, **origin**, **destination**, **time**, **link**, **position**, **spacing**, **speed**. Πάλι δημιουργείται ένας Kafka Consumer που συνδέεται στον broker στο localhost:9092 και διαβάζει μηνύματα από το topic **vehicle_positions**, τον οποίο θέτουμε να διαβάζει από το παλαιότερο διαθέσιμο μήνυμα αν δεν έχει ήδη καταναλώσει κάποια μηνύματα μέσω του **auto_offset_reset='earliest'**. Στη συνέχεια, δημιουργείται ένα κενό Spark DataFrame με βάση το καθορισμένο σχήμα (schema) το οποίο ονομάζεται **accumulated_df**. Αυτό θα χρησιμοποιηθεί για να συσσωρευτούν όλα τα δεδομένα που λαμβάνονται από το Kafka. Υλοποιούμε και τη συνάρτηση **process_message( )**, η οποία παίρνει το μήνυμα από το Kafka και εξάγει τα διάφορα πεδία του, επιστρέφοντας ένα αντικείμενο Python dictionary με τα δεδομένα. Για κάθε μήνυμα, γίνεται εκτύπωση του περιεχομένου του και έπειτα επεξεργασία του μηνύματος μέσω της **process_message( )**, η οποία το

μετατρέπει σε Python dictionary. Μετά δημιουργείται ένα νέο DataFrame από το επεξεργασμένο μήνυμα με τη στήλη **time** (που αρχικά είναι string) να μετατρέπεται σε τύπο Timestamp για να επιτρέπονται χρονικοί υπολογισμοί. Το νέο DataFrame που δημιουργήθηκε από το μήνυμα προστίθεται στο **accumulated_df**, το οποίο διατηρεί όλα τα δεδομένα που έχουν ληφθεί μέχρι εκείνη τη στιγμή. Τέλος, εκτελείται ομαδοποίηση στο συσσωρευμένο DataFrame με βάση τις στήλες **time** και **link** και για κάθε χρονικό σημείο και **link** υπολογίζονται:

- **vcount**: το πλήθος των οχημάτων στο **link**.
- **vspeed**: η μέση ταχύτητα των οχημάτων στο **link**.

Για να τρέξουμε το script στο τερματικό θα χρησιμοποιήσουμε την εντολή **spark-submit spark.py**. Αν το τρέξουμε για ένα μικρό χρονικό διάστημα βλέπουμε ότι το DataFrame που προκύπτει όντως έχει την επιθυμητή μορφή:

```
+-------------------+----+------+------+
|               time|link|vcount|vspeed|
+-------------------+----+------+------+
|2024-09-20 00:51:49|E1I6|     1|  50.0|
|2024-09-20 00:51:49|N1I1|     2|  30.0|
|2024-09-20 00:51:49|S2I2|     2|  30.0|
|2024-09-20 00:51:49|N3I3|     2|  30.0|
|2024-09-20 00:51:49|S4I4|     2|  30.0|
|2024-09-20 00:51:49|N5I5|     2|  30.0|
|2024-09-20 00:51:49|S6I6|     2|  30.0|
|2024-09-20 00:51:54|E1I6|     2|  50.0|
|2024-09-20 00:51:54|N1I1|     2|  30.0|
|2024-09-20 00:51:54|S2I2|     2|  30.0|
|2024-09-20 00:51:54|N3I3|     2|  30.0|
|2024-09-20 00:51:54|S4I4|     2|  30.0|
|2024-09-20 00:51:54|N5I5|     2|  30.0|
|2024-09-20 00:51:54|S6I6|     2|  30.0|
|2024-09-20 00:51:59|E1I6|     2|  50.0|
|2024-09-20 00:51:59|N1I1|     2|  30.0|
|2024-09-20 00:51:59|S2I2|     2|  30.0|
|2024-09-20 00:51:59|N3I3|     1|  30.0|
|2024-09-20 00:51:59|S4I4|     1|  30.0|
|2024-09-20 00:51:59|N5I5|     1|  30.0|
+-------------------+----+------+------+
only showing top 20 rows
```

Εικόνα 5: Στιγμιότυπο Spark consumer

Στο DataFrame ενημερώνονται οι τιμές των στηλών μετά από το διάβασμα των τιμών από τον Kafka producer κάθε φορά. Επίσης, εμφανίζονται μόνο οι κορυφαίες 20 γραμμές του DataFrame στο παραπάνω στιγμιότυπο.

Ερώτημα 3: Αποθήκευση σε MongoDB

Αρχικά, πρέπει να εγκαταστήσουμε τη MongoDB και στη συνέχεια τον MongoDB Spark connector, ο οποίος ενώνει το Spark με τη MongoDB. Για την εγκατάσταση της MongoDB εκτελούμε τις εξής εντολές: **sudo apt update** και **sudo apt install -y mongodb-org**. Στη συνέχεια, ενεργοποιούμε τη MongoDB στη συσκευή μας με την εντολή **sudo systemctl start mongod** και μέσω της **sudo systemctl status mongod** επιβεβαιώνουμε ότι έχει εγκατασταθεί σωστά, όπως φαίνεται και παρακάτω:

```
● mongod.service - MongoDB Database Server
     Loaded: loaded (/usr/lib/systemd/system/mongod.service; enabled; preset: enabled)
     Active: active (running) since Fri 2024-09-20 01:53:26 EEST; 11h ago
       Docs: https://docs.mongodb.org/manual
   Main PID: 1003 (mongod)
     Memory: 82.2M
        CPU: 7min 15.223s
     CGroup: /system.slice/mongod.service
             └─1003 /usr/bin/mongod --config /etc/mongod.conf
```

Εικόνα 6: Επιτυχής εγκατάσταση MongoDB

Επίσης, εγκαθιστούμε και τον Spark connector και μέσω της **pyspark --packages org.mongodb.spark:mongo-spark-connector_2.12:3.0.1 –verbose** επιβεβαιώνουμε ότι εγκαταστάθηκε σωστά:

Εικόνα 7: Επιτυχής εγκατάσταση Spark connector

Τέλος, όσον αφορά τα python scripts, πρέπει να εγκατασταθεί και η βιβλιοθήκη **pymongo**.

spark-mongo.py:

Αρχικά, ορίζεται η σύνδεση με τον MongoDB server και δημιουτγούμε τις συλλογές:

- **raw_positions**: για αποθήκευση των ακατέργαστων δεδομένων.
- **processed_data**: για αποθήκευση των επεξεργασμένων δεδομένων.

Ακόμα, δημιουργούμε ένα SparkSession και ορίζουμε πάλι το σχήμα των δεδομένων που θα λαμβάνονται από το Kafka. Τα δεδομένα που διαβάζονται από το Kafka (που είναι σε μορφή byte) μετατρέπονται σε string και μετά κάνουμε parsing τα json-encoded data χρησιμοποιώντας το σχήμα που ορίστηκε προηγουμένως. Στη συνάρτηση **process_and_save_data( )** τα ωμά δεδομένα που διαβάστηκαν από το Kafka αποθηκεύονται στη συλλογή **raw_positions** της MongoDB. Επίσης, η επεξεργασία των δεδομένων γίνεται όπως και πριν και επεξεργασμένα δεδομένα που υπολογίζονται αποθηκεύονται στη συλλογή **processed_data** της MongoDB. Αν όμως υπάρχει ήδη εγγραφή με τον ίδιο **time** και **link**, τότε απλώς ενημερώνεται η ήδη υπάρχουσα εγγραφή και δεν προστίθεται καινούργια στη συλλογή. Το Spark Streaming διαβάζει τα δεδομένα από τον Kafka και τα ομαδοποιεί σε batches. Για κάθε batch καλείται η συνάρτηση **process_and_save_data( )** με δύο παραμέτρους:

- **df**: Ένα DataFrame που περιέχει τις εγγραφές του τρέχοντος batch.
- **epoch_id**: Ένα μοναδικό αναγνωριστικό για το τρέχον batch.

Σταματάμε το τρέξιμο του script μετά από λίγη ώρα και πρέπει να μεταβούμε στο αντίστοιχο shell της MongoDB για να δούμε τη βάση που δημιουργήθηκε. Αυτό θα γίνει μέσω της εντολής **mongosh**. Στη συνέχεια εκτελούμε την εντολή **use vehicle_data** για να χρησιμοποιήσουμε τη τρέχουσα βάση και μέσω της **show collections** βλέπουμε τις δύο συλλογές που προέκυψαν:

```
test> use vehicle_data
switched to db vehicle_data
vehicle_data> show collections
processed_data
raw_positions
```

Εικόνα 8: Δημιουργία βάσης

Επίσης, οι εντολές **db.raw_positions.find( )** και **db.processed_data.find( )** μας δείχνουν τα documents που έχουν αποθηκευτεί μέχρι τώρα στη βάση μας:

```
vehicle_data> db.raw_positions.find()
[
  {
    _id: ObjectId('66ecbf1eef6e9619b1c42bd6'),
    name: 0,
    origin: 'E1',
    destination: 'W1',
    time: ISODate('2024-09-20T02:02:49.000Z'),
    link: 'E1I6',
    position: 0,
    spacing: -1,
    speed: 50
  },
  {
    _id: ObjectId('66ecbf1eef6e9619b1c42bd7'),
    name: 2,
    origin: 'N1',
    destination: 'W1',
    time: ISODate('2024-09-20T02:02:49.000Z'),
    link: 'N1I1',
    position: 0,
    spacing: -1,
    speed: 30
  },
  {
    _id: ObjectId('66ecbf1eef6e9619b1c42bd8'),
    name: 4,
    origin: 'S2',
    destination: 'W1',
    time: ISODate('2024-09-20T02:02:49.000Z'),
    link: 'S2I2',
    position: 0,
    spacing: -1,
    speed: 30
```

Εικόνα 9: Ωμά δεδομένα

```
vehicle_data> db.processed_data.find()
[
  {
    _id: ObjectId('66ecbf29b34c396cf7db60d3'),
    link: 'N1I1',
    time: '2024-09-20 02:17:26',
    vcount: 1,
    vspeed: 30
  },
  {
    _id: ObjectId('66ecbf29b34c396cf7db60d5'),
    link: 'S2I2',
    time: '2024-09-20 02:17:31',
    vcount: 2,
    vspeed: 30
  },
  {
    _id: ObjectId('66ecbf29b34c396cf7db60d7'),
    link: 'N5I5',
    time: '2024-09-20 02:18:11',
    vcount: 8,
    vspeed: 12.5
  },
  {
    _id: ObjectId('66ecbf29b34c396cf7db60d9'),
    link: 'N6I6',
    time: '2024-09-20 02:18:46',
```

Εικόνα 10: Επεξεργασμένα δεδομένα

Εφόσον, όλα είναι έτοιμα θα χρησιμοποιήσουμε τη συλλογή **processed_data** για την υλοποίηση των queries.

queries.py:

Αρχικά, υλοποιείται η συνάρτηση **query_data_between_period( )** του πρώτου query, η οποία χρησιμοποιείται για την ανάκτηση δεδομένων από τη συλλογή

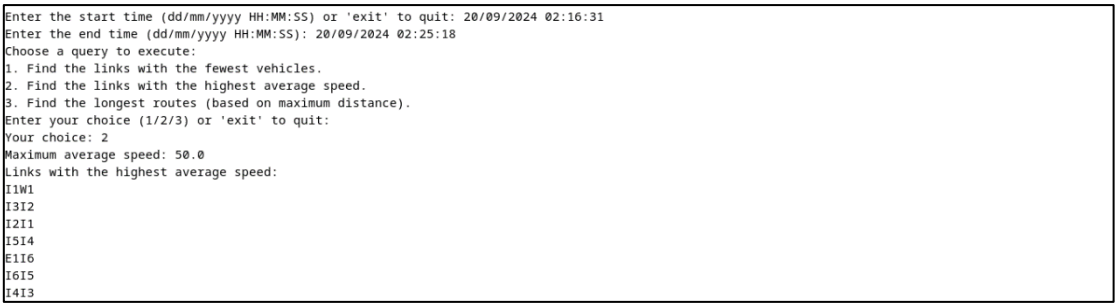
processed_data με βάση ένα χρονικό διάστημα που καθορίζεται από τον χρήστη. Τα δεδομένα που επιστρέφονται είναι μια λίστα εγγραφών **data** που αντιστοιχούν στο συγκεκριμένο διάστημα. Έπειτα, υλοποιείται η συνάρτηση **find_links_with_fewest_vehicles( )** που αναζητά τα **links** που είχαν το λιγότερο πλήθος από οχήματα (**vcount**) από τα δεδομένα που έχουν ανακτηθεί. Πιο συγκεκριμένα, καθορίζει το ελάχιστο **vcount** και δημιουργεί ένα λεξικό με τις ακμές που είχαν τον ίδιο μικρότερο αριθμό οχημάτων. Έτσι, το query θα επιστρέψει παραπάνω όλες τις ακμές σε περίπτωση που παραπάνω από μια αντιστοιχούν στο ελάχιστο **vcount**. Με τον ίδιο τρόπο υλοποιείται και η **find_links_with_highest_avg_speed( )** του δεύτερου query, η οποία όμως τώρα υπολογίζει το μέγιστο **vspeed** και δημιουργεί ένα λεξικό με τις ακμές που έχουν αυτήν τη μέγιστη τιμή. Τέλος, για τη συνάρτηση **find_longest_routes( )** του τρίτου query χρησιμοποιείται η ακατέργαστη συλλογή **raw_positions** για να βρεθούν οι μεγαλύτερες διαδρομές βάσει της θέσης των οχημάτων σε αυτήν. Θα χρησιμοποιηθεί ένα **aggregation pipeline** της MongoDB για να ομαδοποιήσει τις εγγραφές ανά **link** και να υπολογίσει τη διαφορά μεταξύ της μέγιστης και της ελάχιστης θέσης κάθε συνδέσμου. Στη συνέχεια, ταξινομούνται τα αποτελέσματα και επιστρέφονται οι μεγαλύτερες διαδρομές.

Στη **main( )** ο χρήστης μπορεί να εισάγει τις ημερομηνίες έναρξης και λήξης για το χρονικό διάστημα που θέλει να εξετάσει και να επιλέξει ποιο query θέλει να εκτελεστεί κάθε φορά.

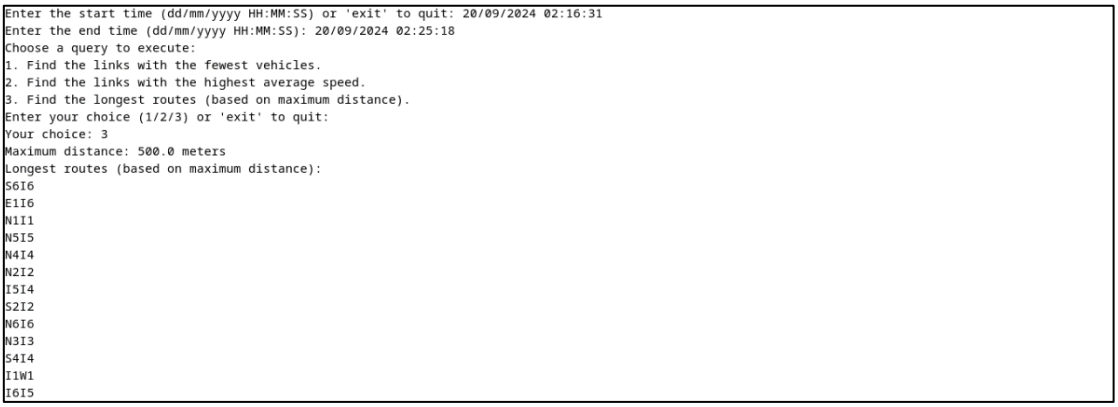
Υλοποιούμε τα ακόλουθα ερωτήματα προς τη βάση και παίρνουμε τις εξής απαντήσεις:

```
(myenv) debian@debian:~/project$ python  queries.py
Enter the start time (dd/mm/yyyy HH:MM:SS) or 'exit' to quit: 20/09/2024 02:16:31
Enter the end time (dd/mm/yyyy HH:MM:SS): 20/09/2024 02:25:18
Choose a query to execute:
1. Find the links with the fewest vehicles.
2. Find the links with the highest average speed.
3. Find the longest routes (based on maximum distance).
Enter your choice (1/2/3) or 'exit' to quit:
Your choice: 1
Minimum vehicle count: 1
Links with the fewest vehicles:
 N1I1
 N6I6
 I2I1
 N4I4
 I3I2
 trip_end
 S4I4
 I4S4
 I5I4
 I1W1
 I2S2
 I6S6
 N2I2
 S2I2
 S6I6
 N5I5
 I3S3
 I5S5
 I1S1
 N3I3
 E1I6
 I4I3
```

Εικόνα 11: Query 1



Εικόνα 12: Query 2



Εικόνα 13: Query 3

Βιβλιογραφία

https://kafka.apache.org/quickstart

https://dev.to/hesbon/apache-kafka-with-python-laa

https://sparkbyexamples.com/pyspark/pyspark-groupby-count-explained/

Working with JSON in Apache Spark | by Neeraj Bhadani | Expedia Group Technology | Medium

Getting started with MongoDB, PySpark, and Jupyter Notebook | MongoDB