Πρότυπο αναφοράς άσκησης

Συστήματα Διαχείρισης Δεδομένων Μεγάλου Όγκου

Εργαστηριακή Άσκηση 2023/24

Όνομα	**Επώνυμο**	**ΑΜ**
Καλλίνικος	Κυριακουλόπουλος	1084583
Σπυρίδων	Ζήκος	1084581

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
Κυριακουλόπουλος Καλλίνικος

ΚΚ

19 / 09 / 2024

Υπογραφή
Ζήκος Σπυρίδων

19 / 09 / 2024

Συνημμένα αρχεία κώδικα

Μαζί με την παρούσα αναφορά υποβάλλουμε τα παρακάτω αρχεία κώδικα

Αρχείο	Αφορά το ερώτημα	Περιγραφή/Σχόλιο
Consumer.py	1	Είναι ο consumer για το topic που δημιουργεί ο producer και χρησιμοποιείται για επαλήθευση της λειτουργίας του kafka producer.
Producer.py	1	Αυτό το script τρέχει τον εξομοιωτή και στέλνει τα μηνύματα στο topic “vehicle_ positions”.
Queries.py	3	Σε αυτό το script εκτελούνται τα ζητούμενα queries.
Run_uxsim.ipynb	1	Εκτελείται προσομοίωση 1 ώρας και χρησιμοποιείται μόνο για να πάρουμε τα figure 1,2 της αναφοράς. Η προσομοίωση που χρησιμοποιείται εκτελείται στο αρχείο producer.py
Spark_processor.py	2,3	Είναι script που διαβάζει τα μηνύματα του topic “vehicle_ positions”, κάνει επεξεργασία στο εισερχόμενο stream δεδομένων και αποθηκεύει τα επεξεργασμένα και μη δεδομένα.
Report.docx		Αναφορά

Τεχνικά χαρακτηριστικά περιβάλλοντος λειτουργίας

Χαρακτηριστικό	**Τιμή**
CPU model	Intel(R) Core(TM) i7-1065G7
CPU clock speed	1.30GHz
Physical CPU cores	4
Logical CPU cores	8
RAM	8
Secondary Storage Type	SSD

Ερώτημα 1: Παραγωγή δεδομένων

Script παραγωγής δεδομένων εξομοίωσης (producer.py):

Σε περιβάλλον με wsl τρέχουμε το αρχείο producer.py και παράγουμε δεδομένα που αντιστοιχούν σε μια ώρα εξομοίωσης. Τα δεδομένα στο τέλος αποθηκεύονται στο dataframe “data” το οποίο θα διαβαστεί από τον kafka producer.

Οι παρακάτω εικόνες έχουν παραχθεί από το jupyter notebook (run_uxsim.ipynb). Το αρχείο αυτό εκτελεί την προσομοίωση δίνοντας κάποιες πληροφορίες για την προσομοίωση και στο τέλος αποθηκεύει τα δεδομένα σε αρχείο csv. Για το project όμως προτιμήθηκε το αρχείο producer.py στο οποίο εκτελείται η προσομοίωση και μετά αμέσως τα δεδομένα της προσομοίωσης που έχουν αποθηκευτεί σε dataframe χρησιμοποιούνται από τον kafka producer.

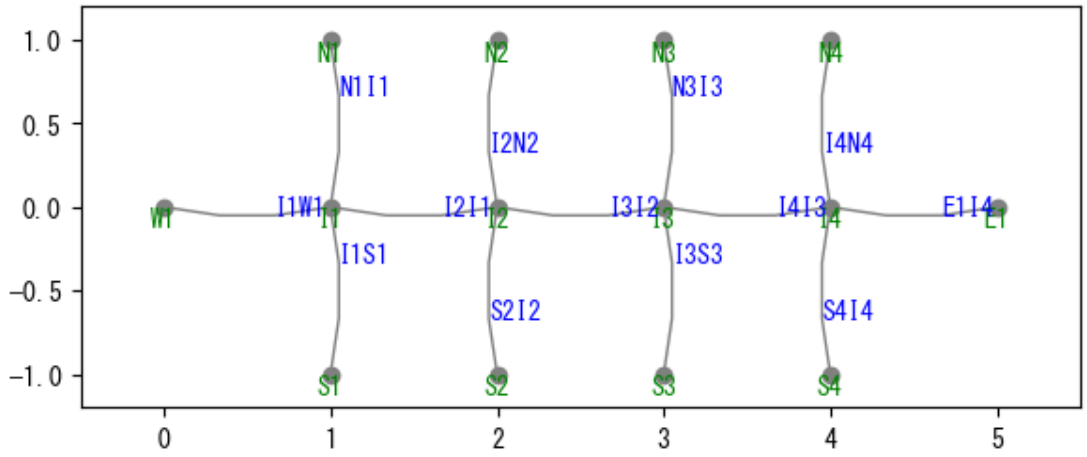

Figure 1: created network

	name	dn	orig	dest	t	link	x	s	v
0	0	5	N1	S1	20	N1I1	0.0	125.0	30.0
1	0	5	N1	S1	25	N1I1	100.0	175.0	20.0
2	0	5	N1	S1	30	N1I1	250.0	175.0	30.0
3	0	5	N1	S1	35	N1I1	400.0	-1.0	30.0
4	0	5	N1	S1	40	I1S1	50.0	-1.0	20.0
5	0	5	N1	S1	45	I1S1	200.0	-1.0	30.0
6	0	5	N1	S1	50	I1S1	350.0	-1.0	30.0
7	0	5	N1	S1	50	trip_end	-1.0	-1.0	-1.0
8	1	5	S2	N2	20	S2I2	0.0	125.0	30.0
9	1	5	S2	N2	25	S2I2	100.0	175.0	20.0

Figure 2: sample simulator results

Script παραγωγής δεδομένων ως kafka producer (producer.py):

Για τη δημιουργία του kafka broker χρειάζεται να εγκαταστήσουμε το kafka version:2.13-3.7.1. Έπειτα εκτελούμε τις ακόλουθες 2 εντολές, *kafka_2.13-3.7.1/bin/zookeeper-server-start.sh kafka_2.13-3.7.1/config/zookeeper.properties* για την ενεργοποίηση του zookeeper και *kafka_2.13-3.7.1/bin/kafka-server-start.sh kafka_2.13-3.7.1/config/server.properties* για την ενεργοποίηση του kafka broker.

Για το script producer.py χρειάζεται να εγκατασταθεί η βιβλιοθήκη kafka-python. Στο project έχει χρησιμοποιηθεί η version 2.0.2 .

Στο script ορίζονται οι μεταβλητές για τη σύνδεση με τον kafka broker. Συνδέεται στο port 9092, παράγει δεδομένα στο topic “vehicle_positions” και χρησιμοποιεί το Kafka API 7.3.1. Με βάση αυτές τις μεταβλητές δημιουργεί τον kafka producer.

```
KAFKA_BOOTSTRAP_SERVERS = os.environ.get("KAFKA_BOOTSTRAP_SERVERS",
"localhost:9092")
KAFKA_TOPIC_TEST = os.environ.get("KAFKA_TOPIC_TEST",
"vehicle_positions")
KAFKA_API_VERSION = os.environ.get("KAFKA_API_VERSION", "7.3.1")
producer = KafkaProducer(
    bootstrap_servers=[KAFKA_BOOTSTRAP_SERVERS],
    api_version=KAFKA_API_VERSION,
)
```

Αφού οριστούν οι μεταβλητές για τον kafka broker και δημιουργηθεί ο kafka producer, τα δεδομένα του dataframe στέλνονται από τον kafka producer με χρήση for loop. Επειδή τα δεδομένα του εξομοιωτή έχουν χρόνο, με βάση πότε συμβαίνει κάθε εγγραφή από την έναρξη της εξομοίωσης θα πρέπει να στέλνονται από τον kafka producer με την αντίστοιχη χρονολογική σειρά. Επομένως θεωρούμε ως αρχή την τρέχουσα ώρα εκτέλεσης του script και κάθε εγγραφή στέλνεται X δευτερόλεπτα από την ώρα έναρξης. Για να γίνει αυτό, το for loop ξεκινά από το 0 μέχρι το μέγιστο χρόνο που υπάρχει στις εγγραφές του dataframe και κάνει βήμα N=5 αφού τα δεδομένα στην προσομοίωση δημιουργούνται κάθε 5 δευτερόλεπτα. Κάθε φορά επιλέγονται οι εγγραφές με χρόνο όσο η μεταβλητή t του for loop. Επίσης παραλείπονται οι εγγραφές στις οποίες το link είναι “waiting_at_origin_node.

Για την ανάγκη επαλήθευσης δημιουργήθηκε το script consumer.py το οποίο διαβάζει τα μηνύματα που υπάρχουν στο topic “vehicle_positions”.

Τα παραπάνω εκτελούνται σε Ubuntu WSL2.

Screenshots:

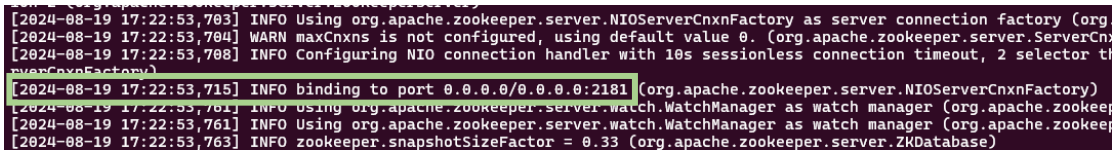

Figure 3: Zookeeper logs

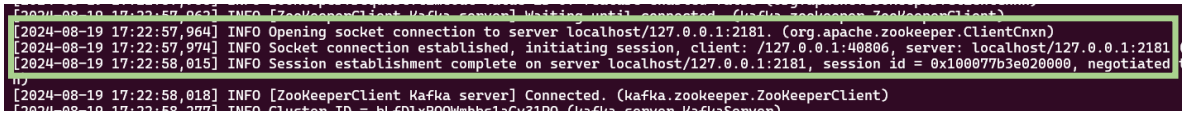

Figure 4:Kafka Broker log

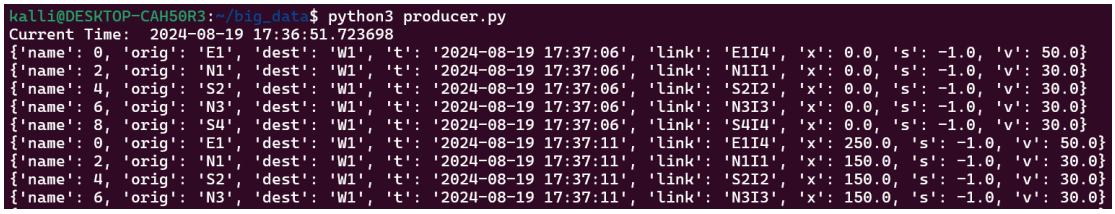

Figure 5: producer.py output

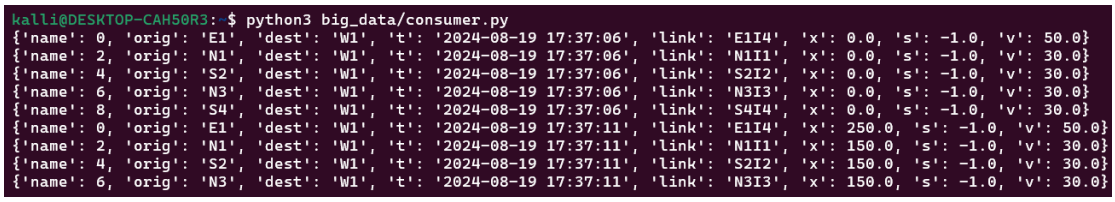

Figure 6: consumer.py output

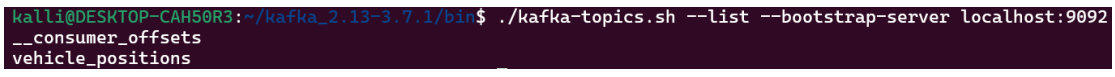

Figure 7: list with existing topics

Ερώτημα 2: Κατανάλωση και επεξεργασία με Spark

Script κατανάλωσης και επεξεργασίας δεδομένων από το Spark (spark_processor.py):

Τα δεδομένα τα οποία στέλνονται στο topic vehicle_positions καταναλώνονται από το Spark και επεξεργάζονται ώστε να εξαχθούν επιπλέον δεδομένα.

Αρχικά δημιουργείται ένα Spark Session και μετά διαβάζεται η ροή των δεδομένων από το topic vehicle_positions. Το dataframe που διαβάστηκε μορφοποιείται ώστε να έχει τις στήλες όπως στο αρχείο που παράγει ο εξομοιωτής και τέλος κάθε batch δεδομένων που διαβάζεται αποθηκεύεται στη βάση δεδομένων.

```
# Streaming data from Kafka topic as a dataframe
lines = spark\
    .readStream\
    .format("kafka")\
    .option("kafka.bootstrap.servers", 'localhost:9092')\
    .option('subscribe', 'vehicle_positions')\
    .load()

lines = lines\
    .selectExpr("CAST(value AS STRING)")\
    .select(from_json(col("value"), schema).alias("data")).select("data.*")

# Writing dataframe (raw data) to database
raw_data = lines \
    .writeStream \
    .foreachBatch(save_raw_data) \
    .start()
```

Figure 8: stream reading, schema passing, saving to mongodb

Το dataframe με τα μη επεξεργασμένα δεδομένα χρησιμοποιείται για την εξαγωγή του συνολικού αριθμού αυτοκινήτων σε κάθε ακμή κάθε χρονική στιγμή και της μέσης ταχύτητας σε κάθε ακμή κάθε χρονική στιγμή.

```
# Processing dataframe
lines = lines\
    .filter(lines.link!='trip_end') \
    .select("time", "link", "speed")\
    .groupBy("time", "link")\
    .agg(avg("speed").alias("avg_SPEED"),
         count("link").alias("num_VEHICLES"))\
    .select("time", "link", "avg_SPEED", "num_VEHICLES")

# Writing dataframe (processed data) to database in update mode
processed_data = lines \
    .writeStream \
    .foreachBatch(save_processed_data) \
    .outputMode("update")\
    .start()
```

Figure 9: processing raw data, saving to mongodb

Για την εκτέλεση του script χρειάζονται οι παρακάτω εξαρτήσεις, η πρώτη αφορά τη διασύνδεση spark με το kafka για την κατανάλωση των δεδομένων από το topic

vehicle_positions και η δεύτερη τη διασύνδεση spark με mongodb για την αποθήκευση των δεδομένων.

1. org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.1
2. org.mongodb.spark:mongo-spark-connector_2.12:10.3.0

Ερώτημα 3: Αποθήκευση σε MongoDB

Η βάση και οι συλλογές της δημιουργούνται από το script spark_processor.py

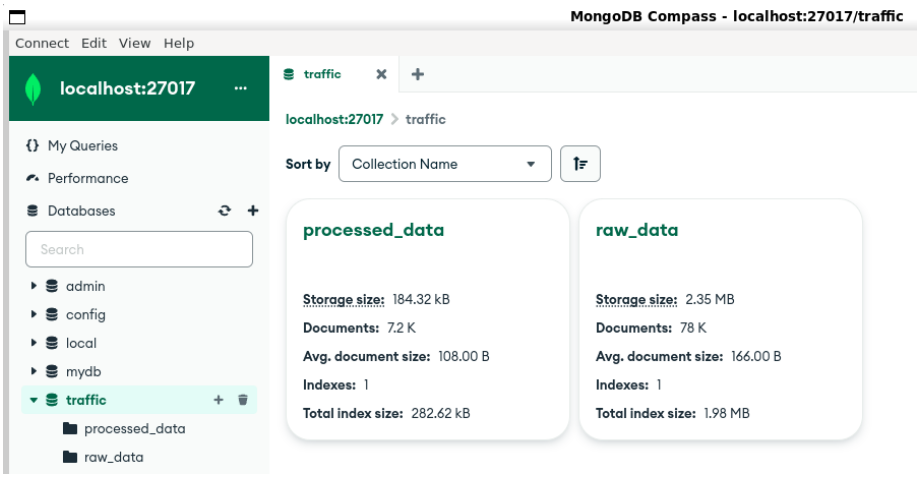

Figure 10: db infos

Script αποθήκευσης δεδομένων σε MongoDB (spark_processor.py):

Χρησιμοποιείται η βιβλιοθήκη pymongo.

Κάθε batch δεδομένων που φτάνει στο spark αποθηκεύεται σε NoSQL βάση δεδομένων. Για αυτό δημιουργείται η βάση traffic και 2 συλλογές.

```
def init_db():
    client = MongoClient('localhost', 27017)

    # Drop the database if it exists & Create the new database
    if "traffic" in client.list_database_names():
        client.drop_database("traffic")
    client["traffic"]
```

Figure 11: db creation

Οι συλλογές δημιουργούνται με την εισαγωγή των δεδομένων. Για την αποθήκευση χρησιμοποιούνται οι συναρτήσεις:

save_raw_data(batch_df, batch_id)

save_processed_data(batch_df, batch_id)

Στην πρώτη συνάρτηση των ακατέργαστων δεδομένων κάθε batch αποθηκεύεται στη συλλογή raw_data και στη δεύτερη τα δεδομένα που προκύπτουν με επεξεργασία αποθηκεύονται στη συλλογή processed_data όπου κάθε φορά εάν υπάρχει στη βάση ήδη document με ίδιο time και link ενημερώνεται και δεν γίνεται νέα εισαγωγή.

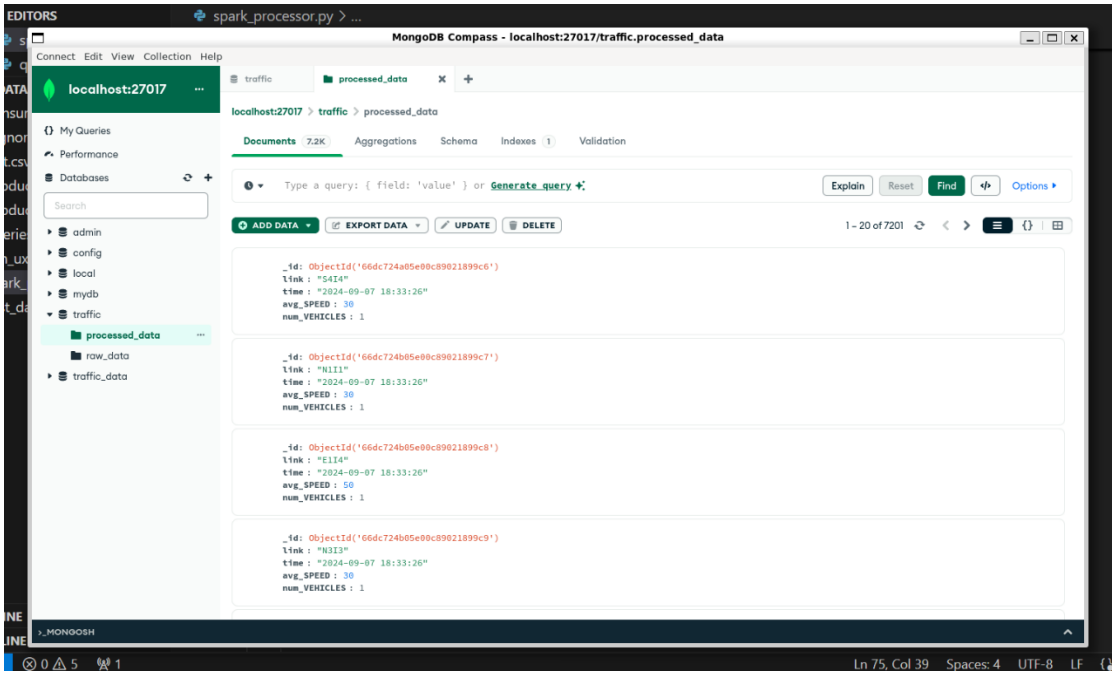

Figure 12: db with UI

Script εκτέλεσης queries σε MongoDB (queries.py):

Ζητείται από τον χρήστη να ορίσει το χρονικό διάστημα και ποιο από τα 3 queries θέλει να εκτελεστεί.

Παράδειγμα:

start_time "2024-09-07 13:56:49"

end_time "2024-09-08 13:57:04"

```
Give Start Time (YYYY-MM-DD hh:mm:ss): 2024-09-07 13:56:49
Give End Time (YYYY-MM-DD hh:mm:ss): 2024-09-08 13:57:04
Select query (1 or 2 or 3): 1
```

Figure 13: user input

```
# 1 ================================================================================
def min_vehicles(start_time, end_time):
    return [
        {"$match": {"time": { "$gte": start_time, "$lte": end_time }}},
        {"$group": {"_id": "$link", "totalVehicles": {"$sum": "$num_VEHICLES"}}},
        {"$group": {"_id": "$totalVehicles", "links": {"$push":"$_id"}}},
        {"$sort": { "_id": 1 }},
        {"$limit": 1}
    ]
```

Figure 14: query 1, which link has the min num of vehicles in the given time period

Για να βρεθεί η ακμή με το λιγότερο αριθμό οχημάτων θα επιλεγούν αρχικά τα documents που η τιμή time είναι μεταξύ των ημερομηνιών που δόθηκαν. Έπειτα θα γίνει ομαδοποίηση με βάση τις ακμές διότι ψάχνουμε ποια ακμή έχει τα λιγότερα αυτοκίνητα και αθροίζουμε

τον αριθμό των οχημάτων. Έχοντας αυτό το αποτέλεσμα κάνουμε ομαδοποίηση με βάση τον αριθμό των οχημάτων ώστε να βρούμε όλες τις ακμές που έχουν αυτή την τιμή. Με αύξουσα ταξινόμηση και περιορισμό αποτελεσμάτων 1 κρατάμε την μικρότερη τιμή μαζί με όλες τις ακμές που την έχουν.

```
Minimum Number of Vehicles = 185
Links = I4N4, I3S3, I2N2
```

Figure 15: query 1 output

```
# 2 ================================================================================
def max_avg_speed(start_time, end_time):
    return [
        {"$match": {"time": { "$gte": start_time, "$lte": end_time }}},
        {"$group": {"_id": "$link", "avgSpeed": {"$max": "$avg_SPEED"}}},
        {"$group": {"_id": "$avgSpeed", "links":{"$push":"$_id"}}},
        {"$sort": { "_id": -1 }},
        {"$limit": 1}
    ]
```

Figure 16: query 2, which link has the max avg speed in the given time period

Για να βρεθεί η ακμή με τη μεγαλύτερη μέση ταχύτητα θα επιλεγούν αρχικά τα documents που η τιμή time είναι μεταξύ των ημερομηνιών που δόθηκαν. Έπειτα θα γίνει ομαδοποίηση με βάση τις ακμές διότι ψάχνουμε ποια ακμή έχει τη μεγαλύτερη μέση ταχύτητα και βρίσκουμε την μέγιστη τιμή του πεδίου avg_SPEED για κάθε ακμή. Έχοντας αυτό το αποτέλεσμα κάνουμε ομαδοποίηση με βάση τη μέγιστη μέση ταχύτητα ώστε να βρεθούν όλες οι ακμές που έχουν την ίδια μέγιστη avgSpeed. Με φθίνουσα ταξινόμηση και περιορισμό αποτελεσμάτων 1 κρατάμε την μεγαλύτερη τιμή μαζί με όλες τις ακμές που την έχουν.

```
Maximum Avg Speed = 50.00000000000001
Links = I1W1
```

Figure 17: query 2 output

```
# 3 ================================================================================
def max_route(start_time, end_time):
    return [
        {"$match": {"time": { "$gte": start_time, "$lte": end_time }}},
        {"$group": {"_id": "$name", "route": {"$push": "$link"}, "distance": {"$sum": "$speed"}}},
        {"$group": {"_id": "$distance", "names":{"$push":"$_id"}, "routes": {"$push": "$route"}}},
        {"$sort": { "_id": -1 }},
        {"$limit": 1}
    ]
```

Figure 18: query3, what was the longest route in the given time period

Για την εύρεση την μεγαλύτερης διαδρομής που έχει κάνει κάποιο αυτοκίνητο σε ένα χρονικό διάστημα, έγινε η υπόθεση ότι η ταχύτητα μεταξύ του διαστήματος των 5 δευτερολέπτων που γίνεται η μέτρηση από τον εξομοιωτή είναι σταθερή. Επομένως το αυτοκίνητο που έχει το μεγαλύτερο άθροισμα ταχυτήτων είναι και αυτό που έχει διανύσει τη μεγαλύτερη απόσταση. Για να βρεθεί η ζητούμενη διαδρομή, αρχικά περιορίζουμε το χρονικό διάστημα και μετά ομαδοποιούμε με βάση τα οχήματα, κρατάμε τις ακμές από τις οποίες διέρχεται για

να γνωρίζουμε τη διαδρομή που κάνει και αθροίζουμε ώστε να βρούμε αυτό με το μεγαλύτερο άθροισμα ταχυτήτων. Έπειτα ομαδοποιούμε με βάση το άθροισμα των ταχυτήτων το οποίο ονομάζουμε distance, κρατάμε τα αυτοκίνητα που έχουν την αντίστοιχη τιμή distance και κρατάμε επίσης τις διαδρομές των αυτοκινήτων με την αντίστοιχη τιμή distance. Τέλος ταξινομούμε με φθίνουσα σειρά και περιορισμό αποτελεσμάτων 1.

```
Maximum Distance = 500.6666666666667
carID: 355   Max Route: E1I4, I4I3, I3I2, I2I1, I1W1, trip_end
carID: 315   Max Route: E1I4, I4I3, I3I2, I2I1, I1W1, trip_end
carID: 610   Max Route: E1I4, I4I3, I3I2, I2I1, I1W1, trip_end
carID: 393   Max Route: E1I4, I4I3, I3I2, I2I1, I1W1, trip_end
carID: 390   Max Route: E1I4, I4I3, I3I2, I2I1, I1W1, trip_end
carID: 352   Max Route: E1I4, I4I3, I3I2, I2I1, I1W1, trip_end
```

Figure 19: query 3 output

Εντολές εκτέλεσης

- Ενεργοποίηση zookeeper:
  kafka_2.13-3.7.1/bin/zookeeper-server-start.shkafka_2.13-3.7.1/config/zookeeper.properties
- Ενεργοποίηση kafka broker:
  kafka_2.13-3.7.1/bin/kafka-server-start.sh kafka_2.13-3.7.1/config/server.properties
- Ενεργοποίηση MongoDB:
  mongod --dbpath /var/lib/mongo --logpath /var/log/mongodb/mongod.log --fork
- Εκτέλεση scripts:
  python3 path_to_file/spark_processor.py
  python3 path_to_file/big_data/producer.py

Βιβλιογραφία

https://toruseo.jp/UXsim/docs/notebooks/demo_notebook_01en.html

https://kafka-python.readthedocs.io/en/master/index.html

https://kafka.apache.org/quickstart

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

https://sandeepkattepogu.medium.com/python-spark-transformations-on-kafka-data-8a19b498b32c

https://medium.com/expedia-group-tech/working-with-json-in-apache-spark-1ecf553c2a8c

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

https://www.mongodb.com/docs/