ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΑΘΗΜΑ: **ΣΥΣΤΗΜΑΤΑ ΔΙΑΧΕΙΡΙΣΗΣ ΔΕΔΟΜΕΝΩΝ ΜΕΓΑΛΟΥ ΌΓΚΟΥ**

ΤΙΤΛΟΣ ΕΡΓΑΣΙΑΣ

Εργαστηριακή Άσκηση 2023/24

ΟΝΟΜ/ΜΟ:	**ΔΑΜΙΑΝΟΣ ΔΙΑΣΑΚΟΣ**	**ΙΑΣΩΝ ΡΑΙΚΟΣ**
ΕΤΟΣ:	**Δ΄**	**Δ΄**
ΑΡ. ΜΗΤΡΩΟΥ:	**1084632**	**1084552**

ΠΑΤΡΑ
ΣΕΠΤΕΜΒΡΙΟΣ 2024
Συστήματα Διαχείρισης Δεδομένων Μεγάλου Όγκου

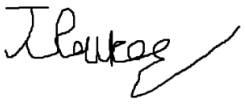
Εργαστηριακή Άσκηση 2023/24

Ονομα	**Επώνυμο**	**ΑΜ**
ΔΑΜΙΑΝΟΣ	ΔΙΑΣΑΚΟΣ	1084632
ΙΑΣΩΝ	ΡΑΙΚΟΣ	1084552

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



ΡΑΙΚΟΣ ΙΑΣΩΝ
19/09/2024

Υπογραφή

ΔΑΜΙΑΝΟΣ ΔΙΑΣΑΚΟΣ
19/09/2024

Συνημμένα αρχεία κώδικα

Μαζί με την παρούσα αναφορά υποβάλλουμε τα παρακάτω αρχεία κώδικα.

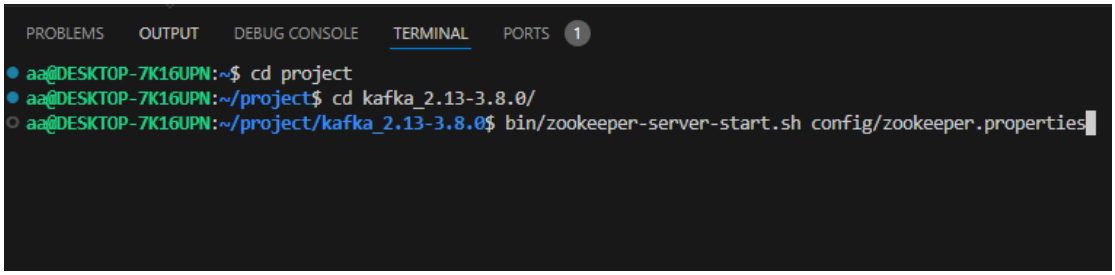
Αρχείο	Αφορά το ερώτημα	Περιγραφή/Σχόλιο
Producer.ipynb	1	Παραγωγή δεδομένων και προσομοίωσης
Consumer.ipynb	1	Κατανάλωση δεδομένων μέσω Kafka broker
kafka_spark.py	2	Διάβασμα δεδομένων και επεξεργασία μέσω spark και εκτύπωση στην κονσόλα
kafka_spark_mongo.py	3	Αποστολή δεδομένων στην Mongo
mongodb_queries.py	3	Εκτέλεση των queries
clear_mongo.py	3	Καθαρισμός βάσης

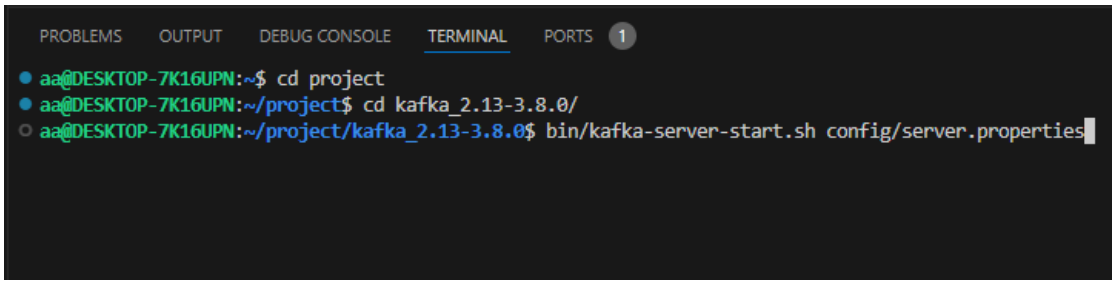
Τεχνικά χαρακτηριστικά περιβάλλοντος λειτουργίας

Χαρακτηριστικό	**Τιμή**
CPU model	*Intel i5-4590*
CPU clock speed	*3.3GHz*
Physical CPU cores	*4*
Logical CPU cores	*4*
RAM	*8GB*
Secondary Storage Type	*SSD*

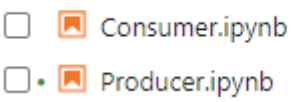
Ερώτημα 1: Παραγωγή δεδομένων

1. Απαραίτητες εντολές για την εκκίνηση του Kafka





2. Άνοιγμα jupyter notebook



Για τη δημιουργία δεδομένων εξομοίωσης χρησιμοποιούμε ένα μοντέλο εξομοίωσης κυκλοφορίας, το οποίο αποστέλλει δεδομένα σε έναν Kafka broker χρησιμοποιώντας έναν Kafka Producer. Τα δεδομένα περιλαμβάνουν πληροφορίες όπως το όνομα του οχήματος, την ταχύτητα, την τοποθεσία, τον σύνδεσμο (link) στον οποίο βρίσκεται το όχημα, και τον χρόνο της εξομοίωσης.

Producer.ipynb:

```
# Function to send vehicle data to Kafka broker
def send_vehicle_data_to_kafka(vehicle_data, simulator_start_time):
    for index, row in vehicle_data.iterrows():
        # Skip sending if 'link' contains "waiting_at_origin_node"
        if "waiting_at_origin_node" in row['link']:
            continue

        # Calculate the timestamp based on simulator start time and t
        timestamp = (simulator_start_time + timedelta(seconds=row['t'])).strftime("%d/%m/%Y %H:%M:%S")

        # Construct JSON object including 't'
        vehicle = {
            "name": row['name'],
            "origin": row['orig'],
            "destination": row['dest'],
            "time": timestamp,
            "t": row['t'],  # Include the simulation time 't' in the message
            "simulator_start_time": simulator_start_time.strftime("%d/%m/%Y %H:%M:%S"),

            "link": row['link'],
            "position": row['x'],
            "spacing": row['s'],
            "speed": row['v']
        }

        # Send the JSON message to Kafka
        producer.send('vehicle_positions', json.dumps(vehicle).encode('utf-8'))  # Encode string to bytes and send
        time.sleep(1)

    # Flush the producer to ensure all messages are sent
    producer.flush()
```

Αρχικά αν στο link του οχήματος περιέχετε η τιμή "waiting_at_origin_node", τότε παραλείπεται από το να σταλεί στον broker.

Η χρόνος υπολογίζεται με βάση την ώρα έναρξης του προσομοιωτή (simulator_start_time) και το πεδίο t, που αντιπροσωπεύει το χρόνο που έχει περάσει από την έναρξη της προσομοίωσης τα οποία τα προσθέτουμε στο JSON.

Για κάθε γραμμή δεδομένων από το vehicle_data, δημιουργείται ένα JSON αντικείμενο που περιλαμβάνει τα στοιχεία:

name → Όνομα οχήματος

origin → Αφετηρία

destination → προορισμός οχήματος

time → Χρονική σήμανση

t → χρόνος από την αρχή της εκτέλεσης της εξομοίωσης

simulator_start_time → Timestamp έναρξης εξομοίωσης

link → τρέχουσα οδική ακμή

position → θέση εντός της οδικής ακμής από την αρχή της

spacing → την απόσταση του οχήματος από το προπορευόμενο

speed → τρέχουσα ταχύτητα του οχήματος

Το JSON μετατρέπεται σε bytes και αποστέλλεται στο Kafka topic vehicle_positions. Το Kafka Producer περιμένει 1 δευτερολέπτο (time.sleep(1)) πριν στείλει το επόμενο μήνυμα, για να προσομοιώσει real time αποστολή δεδομένων.

Μετά την αποστολή όλων των μηνυμάτων, το producer.flush() διασφαλίζει ότι όλα τα μηνύματα έχουν αποσταλεί επιτυχώς στον Kafka broker πριν ολοκληρωθεί η διαδικασία.

```
# Function to simulate vehicles and send data to Kafka
def simulate_and_send_data():
    W = World(
        name="",
        deltan=5,
        tmax=360,  # 1 hour simulation
        print_mode=1,
        save_mode=0,
        show_mode=1,
        random_seed=None,
        duo_update_time=600
    )
    random.seed(None)

    # Network definition
    signal_time = 20
    sf_1 = 1
    sf_2 = 1

    I1 = W.addNode("I1", 1, 0, signal=[signal_time * sf_1, signal_time * sf_2])
    I2 = W.addNode("I2", 2, 0, signal=[signal_time * sf_1, signal_time * sf_2])
    I3 = W.addNode("I3", 3, 0, signal=[signal_time * sf_1, signal_time * sf_2])
    I4 = W.addNode("I4", 4, 0, signal=[signal_time * sf_1, signal_time * sf_2])
    W1 = W.addNode("W1", 0, 0)
    E1 = W.addNode("E1", 5, 0)
    N1 = W.addNode("N1", 1, 1)
    N2 = W.addNode("N2", 2, 1)
    N3 = W.addNode("N3", 3, 1)
    N4 = W.addNode("N4", 4, 1)
    S1 = W.addNode("S1", 1, -1)
    S2 = W.addNode("S2", 2, -1)
    S3 = W.addNode("S3", 3, -1)
    S4 = W.addNode("S4", 4, -1)
```

Αυτό το κομμάτι κώδικα δημιουργεί ένα μοντέλο δικτύου οδών για την εξομοίωση κίνησης οχημάτων χρησιμοποιώντας την κλάση World και διάφορους κόμβους που αντιπροσωπεύουν διασταυρώσεις, σημεία εισόδου/εξόδου κ.λπ

tmax=360: Η εξομοίωση διαρκεί για 360 δευτερόλεπτα (1 ώρα).

Το signal_time καθορίζει τη διάρκεια των φάσεων των φαναριών.

Κάθε κόμβος προστίθεται με τη μέθοδο addNode(), η οποία δέχεται ως παραμέτρους:

όνομα κόμβου: π.χ. "I1".

x, y: Συντεταγμένες θέσης του κόμβου.

signal=[signal_time * sf_1, signal_time * sf_2]: Ο πίνακας signal καθορίζει τη διάρκεια κάθε φάσης των φαναριών.

Οι κόμβοι που προστίθενται είναι: I1, I2, I3, I4:

random_seed=None: η εξομοίωση θα είναι διαφορετική κάθε φορά.

```
    # E <-> W direction: signal group 0
    for n1, n2 in [[W1, I1], [I1, I2], [I2, I3], [I3, I4], [I4, E1]]:
        W.addLink(n2.name + n1.name, n2, n1, length=500, free_flow_speed=50, jam_density=0.2, number_of_lanes=3, signal_group=0)

    # N -> S direction: signal group 1
    for n1, n2 in [[N1, I1], [I1, S1], [N3, I3], [I3, S3]]:
        W.addLink(n1.name + n2.name, n1, n2, length=500, free_flow_speed=30, jam_density=0.2, signal_group=1)

    # S -> N direction: signal group 2
    for n1, n2 in [[N2, I2], [I2, S2], [N4, I4], [I4, S4]]:
        W.addLink(n2.name + n1.name, n2, n1, length=500, free_flow_speed=30, jam_density=0.2, signal_group=1)

    # Random demand definition every 30 seconds
    dt = 60  # Increase this to reduce vehicle entry frequency
    demand = 0.5  # Reduce this to decrease the number of vehicles
    demands = []
    for t in range(0, 3600, dt):
        dem = random.uniform(0, demand)
        for n1, n2 in [[N1, S1], [S2, N2], [N3, S3], [S4, N4]]:
            W.adddemand(n1, n2, t, t + dt, dem * 0.25)
            demands.append({"start": n1.name, "dest": n2.name, "times": {"start": t, "end": t + dt}, "demand": dem})
        for n1, n2 in [[E1, W1], [N1, W1], [S2, W1], [N3, W1], [S4, W1]]:
            W.adddemand(n1, n2, t, t + dt, dem * 0.75)
            demands.append({"start": n1.name, "dest": n2.name, "times": {"start": t, "end": t + dt}, "demand": dem})

    W.exec_simulation()
    W.analyzer.print_simple_stats()

    # Capture simulation start time
    simulator_start_time = datetime.now()
    print("Simulation Start Time:", simulator_start_time)
    # Get vehicle data as pandas DataFrame
    vehicle_data = W.analyzer.vehicles_to_pandas()
    # Print all rows of vehicle data
    print(vehicle_data.to_string())
    # Send vehicle data to Kafka broker
    send_vehicle_data_to_kafka(vehicle_data, simulator_start_time)

# Start simulating and sending data to Kafka
simulate_and_send_data()
```

Τα links μεταξύ των nodes γίνονται μέσω της addLink(). Κάθε σύνδεση συνδέει δύο κόμβους και έχει συγκεκριμένα χαρακτηριστικά:

n1 και n2: Οι κόμβοι που συνδέονται.

length=500: Το μήκος του δρόμου σε μέτρα.

free_flow_speed: Η ταχύτητα κίνησης σε κανονικές συνθήκες.

jam_density=0.2: Η μέγιστη πυκνότητα κυκλοφορίας.

number_of_lanes: Ο αριθμός λωρίδων

signal_group: Ορίζει σε ποιο φανάρι ανήκει η σύνδεση

Διαδρομή Ανατολή - Δύση (signal group 0)

Κατεύθυνση Βοράς - Νότος (signal group 1)

Κατεύθυνση Νότος - Βοράς (signal group 2)

Η ζήτηση οχημάτων ορίζεται κάθε 60 δευτερόλεπτα (dt = 60).

Το demand είναι τυχαίο κάθε φορά και ορίζεται με βάση την τυχαία τιμή random.uniform(0, demand).

Με την συνάρτηση W.exec_simulation() εκτελείτε η εξομοίωση.

Έπειτ η W.analyzer.print_simple_stats() εκτυπώνει βασικά στατιστικά στοιχεία της εξομοίωσης.

Τα δεδομένα των οχημάτων ομαδοποιούνται με την συνάρτηση W.analyzer.vehicles_to_pandas() και μετατρέπονται σε ένα DataFrame της pandas για την αποστολή με Kafka

Η συνάρτηση

send_vehicle_data_to_kafka(vehicle_data, simulator_start_time) αποστέλλει τα δεδομένα οχημάτων σε έναν Kafka broker.

Το simulator_start_time καταγράφει την έναρξη της εξομοίωσης για συγχρονισμό

Consumer.ipynb:

```
import os

import json

from kafka import KafkaConsumer


# Create Kafka consumer

consumer = KafkaConsumer(

  'vehicle_positions', # Topic to subscribe to

  bootstrap_servers='localhost:9092', # Kafka broker address

  auto_offset_reset='latest', # Start reading at the latest message

  group_id='my-group', # Consumer group ID

  enable_auto_commit=True, # Automatically commit offsets

)


# Start consuming messages

try:

  for message in consumer:

    # Decode the message and parse the JSON

    vehicle_data = json.loads(message.value.decode("utf-8"))


    # Remove the simulator_start_time from the message if it exists

    vehicle_data.pop('simulator_start_time', None)


    # Print the updated message without simulator_start_time

    print(json.dumps(vehicle_data))

except KeyboardInterrupt:

  print("Consumer stopped.")

  consumer.close()
```

Δημιουργείτε ένας Kafka Consumer που διαβάζει μηνύματα από το topic vehicle_positions στον Kafka broker στην localhost:9092

Το for loop message in consumer διαβάζει συνεχώς νέα μηνύματα από το Kafka topic και τα εκτυπώνει στην κονσόλα. Παραλείπει την εκτύπωση του simulation start time το οποίο χρειαζόμαστε για την διαχείριση με spark αργότερα.

Η κατανάλωση συνεχίζεται μέχρι να διακοπεί η διαδικασία από τον χρήστη με KeyboardInterrupt (π.χ. Ctrl + C). Σε αυτήν την περίπτωση, ο καταναλωτής κλείνει με την εντολή consumer.close().

Αποτελέσματα τρέχοντας πρώτα τον Consumer και μετά τον Producer σε jupyter notebook.

```
# Start simulating and sending data to Kafka
simulate_and_send_data()
```

```
Simulating...
      time| # of vehicles| ave speed| computation time
       0 s|         0 vehs|   0.0 m/s|     0.00 s
     355 s|        60 vehs|  43.3 m/s|     0.11 s
 simulation finished
results:
 average speed:  24.2 m/s
 number of completed trips:      260 / 3230
 average travel time of trips:   54.6 s
 average delay of trips:         12.8 s
 delay ratio:                    0.235
Simulation Start Time: 2024-09-17 23:21:15.159195
    name  dn orig dest    t                    link           x           s           v
0      0   5   N1   S1   60                    N1I1    0.000000  500.000000  30.000000
1      0   5   N1   S1   65                    N1I1  150.000000  350.000000  30.000000
2      0   5   N1   S1   70                    N1I1  300.000000  200.000000  30.000000
3      0   5   N1   S1   75                    N1I1  450.000000   -1.000000  30.000000
4      0   5   N1   S1   80                    I1S1  100.000000   -1.000000  10.000000
```

```
except KeyboardInterrupt:
    print("Consumer stopped.")
    consumer.close()
```


```
{"name": "0", "origin": "N1", "destination": "S1", "time": "17/09/2024 23:22:15", "t": 60, "link": "N1I1", "position": 0.0,
"spacing": 500.0, "speed": 30.0}
{"name": "0", "origin": "N1", "destination": "S1", "time": "17/09/2024 23:22:20", "t": 65, "link": "N1I1", "position": 150.0,
"spacing": 350.0, "speed": 30.0}
{"name": "0", "origin": "N1", "destination": "S1", "time": "17/09/2024 23:22:25", "t": 70, "link": "N1I1", "position": 300.0,
"spacing": 200.0, "speed": 30.0}
{"name": "0", "origin": "N1", "destination": "S1", "time": "17/09/2024 23:22:30", "t": 75, "link": "N1I1", "position": 450.0,
"spacing": -1.0, "speed": 30.0}
{"name": "0", "origin": "N1", "destination": "S1", "time": "17/09/2024 23:22:35", "t": 80, "link": "I1S1", "position": 100.0,
"spacing": -1.0, "speed": 10.0}
```

Ερώτημα 2: Κατανάλωση και επεξεργασία με Spark

Ο κώδικας του kafka_spark.py έχει σχεδιαστεί για να καταναλώνει δεδομένα σε πραγματικό χρόνο από ένα θέμα Kafka που ονομάζεται vehicle_positions, το οποίο επεξεργάζεται χρησιμοποιώντας τη δομημένη ροή του Apache Spark και εμφανίζει συγκεντρωτικά αποτελέσματα με βάση τις κινήσεις των οχημάτων. Τα δεδομένα περιλαμβάνουν την ταχύτητα των οχημάτων, τη θέση τους και άλλα σχετικά χαρακτηριστικά και το τελικό αποτέλεσμα εμφανίζει μια ανά λεπτό καταμέτρηση των οχημάτων και τη μέση ταχύτητά τους για κάθε συγκεκριμένη οδική σύνδεση.

Ο κώδικας ξεκινά με την αρχικοποίηση μιας περιόδου λειτουργίας Spark με το όνομα της εφαρμογής να έχει οριστεί σε KafkaSparkProcessing. Αυτή η συνεδρία χρησιμεύει ως σημείο εισόδου για τη λειτουργικότητα του Spark και διαχειρίζεται την αλληλεπίδραση με τη συστάδα Spark.

```
# Initialize SparkSession
spark = SparkSession.builder \
    .appName("KafkaSparkProcessing") \
    .getOrCreate()
```

Το επίπεδο καταγραφής έχει οριστεί σε WARN για να μειωθεί η “πολυλογία” και να επικεντρωθεί σε σημαντικά συμβάντα και σφάλματα.

```
# Set log level to WARN to reduce verbosity
spark.sparkContext.setLogLevel("WARN")
```

Τα δεδομένα διαβάζονται από ένα θέμα Kafka με όνομα vehicle_positions. Καθορίζονται οι ρυθμίσεις Kafka, συμπεριλαμβανομένης της τοποθεσίας του διακοσμητή Kafka (localhost:9092) και του ονόματος του θέματος.

```
# Read data from Kafka
df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "vehicle_positions") \
    .load()
```

Τα δεδομένα που προέρχονται από την Kafka είναι σε δυαδική μορφή, οπότε ο κώδικας μετατρέπει το πεδίο τιμών των μηνυμάτων της Kafka σε μορφή συμβολοσειράς.

```
# Convert Kafka data to DataFrame
df = df.selectExpr("CAST(value AS STRING) AS json")
```

Τα εισερχόμενα δεδομένα θεωρούνται ότι είναι σε μορφή JSON. Για να δομηθούν σωστά τα δεδομένα, ορίζεται ένα σχήμα με τη χρήση StructType με τα ακόλουθα πεδία:

- name: Κωδικός του οχήματος (ή της ομάδας / διμοιρίας οχημάτων) (String)
- origin: Αφετηρία οχήματος (String)
- destination: Προορισμός οχήματος (Integer)
- time: Χρόνος από την αρχή της εκτέλεσης της εξομοίωσης (String)
- link: Τρέχουσα οδική ακμή του οχήματος ή ο κωδικός κατάστασης (String)
- simulator_start_time: Η χρονική στιγμή που άρχισε η εξομοίωση (String)
- position: Θέση εντός της οδικής ακμής από την αρχή της (Double)
- spacing: Η απόσταση του οχήματος από το προπορευόμενο (Double)
- speed: Τρέχουσα ταχύτητα του οχήματος (Double)

```
# Define schema for the incoming data, including simulator_start_time
schema = StructType([
    StructField("name", StringType(), True),
    StructField("origin", StringType(), True),
    StructField("destination", IntegerType(), True),
    StructField("time", StringType(), True),
    StructField("link", StringType(), True),
    StructField("simulator_start_time", StringType(), True),  # Include simulator_start_time
    StructField("position", DoubleType(), True),
    StructField("spacing", DoubleType(), True),
    StructField("speed", DoubleType(), True)
])
```

Στη συνέχεια, τα δεδομένα JSON αναλύονται στο καθορισμένο σχήμα.

```
# Parse the JSON data into the schema
df = df.selectExpr("json").select(from_json(col("json"), schema).alias("data")).select("data.*")
```

Ο κώδικας μετατρέπει τα πεδία time και simulator_start_time από συμβολοσειρές σε χρονοσφραγίδες για ακριβείς λειτουργίες με βάση τον χρόνο. Η μορφή που χρησιμοποιείται για τη μετατροπή είναι dd/MM/yyyy HH:mm:ss.

```
# Convert 'time' and 'simulator_start_time' to timestamps
df = df.withColumn("time", unix_timestamp(col("time"), "dd/MM/yyyy HH:mm:ss").cast(TimestampType()))
df = df.withColumn("simulator_start_time", unix_timestamp(col("simulator_start_time"), "dd/MM/yyyy HH:mm:ss").cast(TimestampType()))
```

Στην συνέχεια ο κώδικας εκτελεί τους ακόλουθους μετασχηματισμούς στα δεδομένα:

- Windowing: Τα δεδομένα ομαδοποιούνται σε παράθυρα του 1 λεπτού με βάση το time.
- Link Grouping: Τα δεδομένα ομαδοποιούνται επίσης με βάση το οδικό τμήμα (link) με το οποίο συνδέονται τα οχήματα.
- Vehicle Count: Ο κώδικας μετρά τον αριθμό των οχημάτων σε κάθε link κατά τη διάρκεια κάθε παραθύρου 1 λεπτού.
- Average Speed: Υπολογίζει τη μέση ταχύτητα των οχημάτων σε κάθε link κατά τη διάρκεια του παραθύρου.

- Minimum Simulator Start Time: Η νωρίτερη ώρα έναρξης του προσομοιωτή (simulator_start_time) σε κάθε παράθυρο.

```
# Extract and process the required fields
result_df = df.groupBy(
    window(col("time"), "1 minute"),  # Window duration
    col("link")
).agg(
    count("name").alias("vcount"),       # Count of vehicles per link
    avg("speed").alias("vspeed"),        # Average speed of vehicles per link
    expr("min(simulator_start_time)").alias("min_simulator_start_time")  # Minimum simulator_start_time in each window
)
```

Υπολογίζεται επίσης η διάρκεια σε δευτερόλεπτα μεταξύ της ελάχιστης ώρας έναρξης του προσομοιωτή και της τρέχουσας ώρας.

```
# Calculate the duration in seconds between simulator_start_time and the current window's start time
result_df = result_df.withColumn(
    "duration_since_start",
    expr("unix_timestamp(current_timestamp()) - unix_timestamp(min_simulator_start_time)")  # Calculate difference in seconds
)
```

Επιλέγουμε μόνο τις απαραίτητες στήλες για έξοδο:

- Window: Το χρονικό παράθυρο για τη συνάθροιση.
- Link: Το οδικό τμήμα.
- Vcountοχημάτων: Αριθμός οχημάτων.
- Vspeed: Μέση ταχύτητα των οχημάτων.
- Duration_since_start: Διάρκεια από την ελάχιστη ώρα έναρξης του προσομοιωτή.

```
# Select only the necessary columns
result_df = result_df.select(
    col("window"),
    col("link"),
    col("vcount"),
    col("vspeed"),
    col("duration_since_start")
)
```

Τα αποτελέσματα εξάγονται στην κονσόλα χρησιμοποιώντας τον μηχανισμό ροής του Spark. Η λειτουργία εξόδου έχει οριστεί σε πλήρη, πράγμα που σημαίνει ότι κάθε φορά που επεξεργάζεται μια νέα παρτίδα, εκτυπώνεται ο πλήρης πίνακας αποτελεσμάτων.

```
# Output the results to the console
query = result_df.writeStream \
    .outputMode("complete") \
    .format("console") \
    .option("truncate", "false") \
    .start()
```

Τέλος, ο κώδικας περιμένει τον τερματισμό της διεργασίας ροής, διασφαλίζοντας ότι εκτελείται συνεχώς μέχρι να διακοπεί χειροκίνητα.

```
# Await termination
query.awaitTermination()
```

Εκτέλεση του αρχείου kafka_spark.py

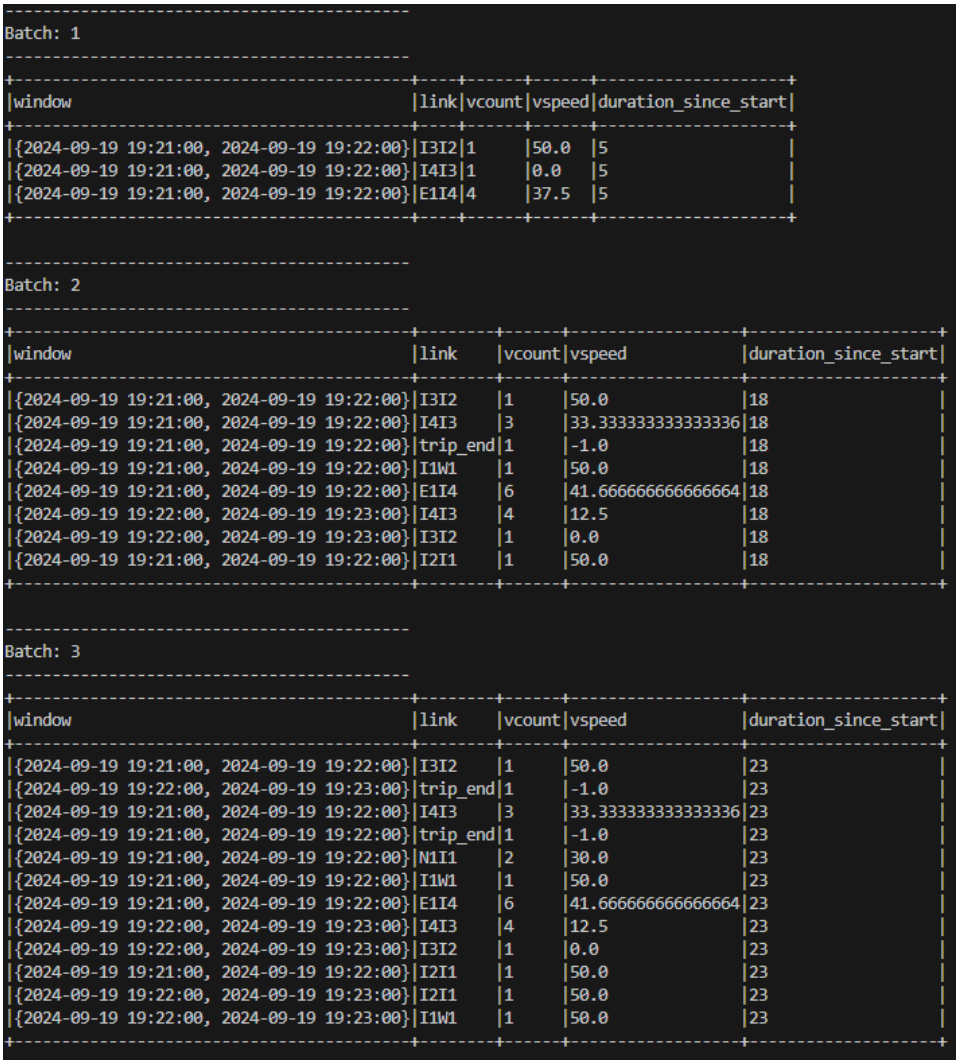
Ξεκινάμε την εκτέλεση του αρχείου με την εντολή:

```
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.1 kafka_spark.py
```

Καθώς η Spark περιμένει δεδομένα, εκτελούμε τον Producer μέσω του jupyter notebook.

```
Simulation Start Time: 2024-09-19 19:20:42.657463
    name  dn orig dest   t                      link           x           s          v
0      0   5   E1   W1  30                      E1I4    0.000000   -1.000000  50.000000
1      0   5   E1   W1  35                      E1I4  250.000000   -1.000000  50.000000
2      0   5   E1   W1  40                      E1I4  500.000000   -1.000000  50.000000
3      0   5   E1   W1  45                      E1I4  500.000000   -1.000000   0.000000
4      0   5   E1   W1  50                      I4I3  250.000000   -1.000000   0.000000
5      0   5   E1   W1  55                      I3I2  250.000000   -1.000000  50.000000
6      0   5   E1   W1  60                      I2I1  250.000000   -1.000000  50.000000
7      0   5   E1   W1  65                      I1W1  250.000000   -1.000000  50.000000
8      0   5   E1   W1  65                 trip_end   -1.000000   -1.000000  -1.000000
9      1   5   E1   W1  60                      E1I4    0.000000   -1.000000  50.000000
10     1   5   E1   W1  65                      E1I4  250.000000   -1.000000  50.000000
11     1   5   E1   W1  70                      I4I3    0.000000   -1.000000  50.000000
12     1   5   E1   W1  75                      I4I3  250.000000   -1.000000  50.000000
13     1   5   E1   W1  80                      I4I3  500.000000   -1.000000  50.000000
14     1   5   E1   W1  85                      I4I3  500.000000   -1.000000   0.000000
15     1   5   E1   W1  90                      I4I3  500.000000   -1.000000   0.000000
```

Με βάση τα αποτελέσματα των πρώτων πακέτων δεδομένων βλέπουμε ότι οι υπολογισμοί γίνονται σωστά.



```
-------------------------------------------
Batch: 1
-------------------------------------------
+------------------------------------------+----+------+------+-------------------+
|window                                    |link|vcount|vspeed|duration_since_start|
+------------------------------------------+----+------+------+-------------------+
|{2024-09-19 19:21:00, 2024-09-19 19:22:00}|I3I2|1     |50.0  |5                  |
|{2024-09-19 19:21:00, 2024-09-19 19:22:00}|I4I3|1     |0.0   |5                  |
|{2024-09-19 19:21:00, 2024-09-19 19:22:00}|E1I4|4     |37.5  |5                  |
+------------------------------------------+----+------+------+-------------------+


-------------------------------------------
Batch: 2
-------------------------------------------
+------------------------------------------+--------+------+-----------------+-------------------+
|window                                    |link    |vcount|vspeed           |duration_since_start|
+------------------------------------------+--------+------+-----------------+-------------------+
|{2024-09-19 19:21:00, 2024-09-19 19:22:00}|I3I2    |1     |50.0             |18                 |
|{2024-09-19 19:21:00, 2024-09-19 19:22:00}|I4I3    |3     |33.333333333333336|18                 |
|{2024-09-19 19:21:00, 2024-09-19 19:22:00}|trip_end|1     |-1.0             |18                 |
|{2024-09-19 19:21:00, 2024-09-19 19:22:00}|I1W1    |1     |50.0             |18                 |
|{2024-09-19 19:21:00, 2024-09-19 19:22:00}|E1I4    |6     |41.666666666666664|18                 |
|{2024-09-19 19:22:00, 2024-09-19 19:23:00}|I4I3    |4     |12.5             |18                 |
|{2024-09-19 19:22:00, 2024-09-19 19:23:00}|I3I2    |1     |0.0              |18                 |
|{2024-09-19 19:21:00, 2024-09-19 19:22:00}|I2I1    |1     |50.0             |18                 |
+------------------------------------------+--------+------+-----------------+-------------------+


-------------------------------------------
Batch: 3
-------------------------------------------
+------------------------------------------+--------+------+-----------------+-------------------+
|window                                    |link    |vcount|vspeed           |duration_since_start|
+------------------------------------------+--------+------+-----------------+-------------------+
|{2024-09-19 19:21:00, 2024-09-19 19:22:00}|I3I2    |1     |50.0             |23                 |
|{2024-09-19 19:22:00, 2024-09-19 19:23:00}|trip_end|1     |-1.0             |23                 |
|{2024-09-19 19:21:00, 2024-09-19 19:22:00}|I4I3    |3     |33.333333333333336|23                 |
|{2024-09-19 19:21:00, 2024-09-19 19:22:00}|trip_end|1     |-1.0             |23                 |
|{2024-09-19 19:21:00, 2024-09-19 19:22:00}|N1I1    |2     |30.0             |23                 |
|{2024-09-19 19:21:00, 2024-09-19 19:22:00}|I1W1    |1     |50.0             |23                 |
|{2024-09-19 19:21:00, 2024-09-19 19:22:00}|E1I4    |6     |41.666666666666664|23                 |
|{2024-09-19 19:22:00, 2024-09-19 19:23:00}|I4I3    |4     |12.5             |23                 |
|{2024-09-19 19:22:00, 2024-09-19 19:23:00}|I3I2    |1     |0.0              |23                 |
|{2024-09-19 19:21:00, 2024-09-19 19:22:00}|I2I1    |1     |50.0             |23                 |
|{2024-09-19 19:22:00, 2024-09-19 19:23:00}|I2I1    |1     |50.0             |23                 |
|{2024-09-19 19:22:00, 2024-09-19 19:23:00}|I1W1    |1     |50.0             |23                 |
+------------------------------------------+--------+------+-----------------+-------------------+
```

Ερώτημα 3: Αποθήκευση σε MongoDB

Το ερώτημα 3 μοιάζει με το 2ο όσον αφορά την σύνδεση Kafka - Spark αλλά χρειάστηκε να προσθεθει το παρακάτω κομμάτι για την σύνδεση με την MongoDB: kafka_spark_mongo.py:

```
# Define the MongoDB connection options
mongo_uri = "mongodb://localhost:27017/"
database = "vehicle_db"
raw_collection = "raw_vehicle_positions"
processed_collection = "processed_vehicle_positions"


# Write raw data to MongoDB
def write_raw_to_mongo(df, epoch_id):
    df.write.format("mongo") \
        .mode("append") \
        .option("uri", mongo_uri) \
        .option("database", database) \
        .option("collection", raw_collection) \
        .save()


# Write processed data to MongoDB
def write_processed_to_mongo(df, epoch_id):
    df.write.format("mongo") \
        .mode("append") \
        .option("uri", mongo_uri) \
        .option("database", database) \
        .option("collection", processed_collection) \
        .save()


# Start streaming queries to MongoDB
raw_query = df.writeStream \
    .foreachBatch(write_raw_to_mongo) \
    .outputMode("append") \
    .start()


processed_query = result_df.writeStream \
    .foreachBatch(write_processed_to_mongo) \
    .outputMode("complete") \
    .start()
```

URI: Η διεύθυνση του MongoDB server → localhost.

Βάση δεδομένων→ vehicle_db.

raw_vehicle_positions για τα ωμά δεδομένα.

processed_vehicle_positions για τα επεξεργασμένα δεδομένα.

Η συνάρτηση write_raw_to_mongo αποθηκεύει τα ωμά δεδομένα από το Spark Streaming στη συλλογή raw_vehicle_positions.

Παρόμοια με την παραπάνω, η συνάρτηση write_processed_to_mongo αποθηκεύει τα επεξεργασμένα δεδομένα στη συλλογή processed_vehicle_positions.

raw_query: Ροή για την αποθήκευση των ωμών δεδομένων.

processed_query: Ροή για την αποθήκευση των επεξεργασμένων δεδομένων.

```
vehicle_db> db.raw_vehicle_positions.find().sort({ _id: 1 }).forEach(printjson)
{
  _id: ObjectId('66ec50232e640109c52a7698'),
  name: '0',
  origin: 'E1',
  time: ISODate('2024-09-19T16:24:31.000Z'),
  link: 'E1I4',
  simulator_start_time: ISODate('2024-09-19T16:24:01.000Z'),
  position: 0,
  spacing: -1,
  speed: 50
}
{
  _id: ObjectId('66ec502a2e640109c52a769c'),
  name: '0',
  origin: 'E1',
  time: ISODate('2024-09-19T16:24:36.000Z'),
  link: 'E1I4',
  simulator_start_time: ISODate('2024-09-19T16:24:01.000Z'),
  position: 250,
  spacing: -1,
  speed: 50
}
{
  _id: ObjectId('66ec502a2e640109c52a769d'),
  name: '0',
  origin: 'E1',
  time: ISODate('2024-09-19T16:24:41.000Z'),
  link: 'E1I4',
  simulator_start_time: ISODate('2024-09-19T16:24:01.000Z'),
  position: 500,
  spacing: -1,
  speed: 50
}
{
  _id: ObjectId('66ec502a2e640109c52a769f'),
  name: '0',
  origin: 'E1',
  time: ISODate('2024-09-19T16:24:46.000Z'),
  link: 'E1I4',
  simulator_start_time: ISODate('2024-09-19T16:24:01.000Z'),
  position: 500,
  spacing: -1,
  speed: 0
}
```

```
vehicle_db> db.processed_vehicle_positions.find().sort({ _id: 1 }).forEach(printjson)
{
  _id: ObjectId('66ec50262e640109c52a769a'),
  window: {
    start: ISODate('2024-09-19T16:24:00.000Z'),
    end: ISODate('2024-09-19T16:25:00.000Z')
  },
  link: 'E1I4',
  vcount: Long('1'),
  vspeed: 50,
  duration_since_start: Long('0')
}
{
  _id: ObjectId('66ec502b2e640109c52a76a6'),
  window: {
    start: ISODate('2024-09-19T16:25:00.000Z'),
    end: ISODate('2024-09-19T16:26:00.000Z')
  },
  link: 'I1W1',
  vcount: Long('1'),
  vspeed: 50,
  duration_since_start: Long('9')
}
{
  _id: ObjectId('66ec502c2e640109c52a76a8'),
  window: {
    start: ISODate('2024-09-19T16:24:00.000Z'),
    end: ISODate('2024-09-19T16:25:00.000Z')
  },
  link: 'I4I3',
  vcount: Long('1'),
  vspeed: 0,
  duration_since_start: Long('9')
}
{
  _id: ObjectId('66ec502c2e640109c52a76aa'),
  window: {
    start: ISODate('2024-09-19T16:24:00.000Z'),
    end: ISODate('2024-09-19T16:25:00.000Z')
  },
  link: 'E1I4',
  vcount: Long('4'),
  vspeed: 37.5,
  duration_since_start: Long('9')
}
```

ΣΗΜΑΝΤΙΚΟ:

Για να τρέξουμε το αρχείο kafka_spark_mongo.py πρέπει να εκτελέσουμε αυτήν την εντολή:

spark-submit --packages org.apache.spark:spark-streaming-kafka-0-10_2.12:3.0.1,org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.1,org.mongodb.spark:mongo-spark-connector_2.12:3.0.1 kafka_spark_mongo.py localhost:9092 subscribe uxsim

Μόλις τρέξουμε αυτήν την εντολή θα χρειαστεί να εκκινήσουμε τον Producer και έπειτα τα δεδομένα θα εμφανιστούν στην MongoDB στην βάση vehicle_db με τις εντολές:

db.raw_vehicle_positions.find().sort({ _id: 1 }).forEach(printjson)

db.processed_vehicle_positions.find().sort({ _id: 1 }).forEach(printjson)

mongo_queries.py:

```
from pymongo import MongoClient

from datetime import datetime


# Connect to MongoDB

client = MongoClient("mongodb://localhost:27017/")

db = client["vehicle_db"]


def get_vehicle_count_least_edge(start_time, end_time):

…

…

…

def get_highest_avg_speed_edge(start_time, end_time):

…

…

…

def get_longest_distance(start_time, end_time):

…

…

…


# Define time period for queries

start_time = datetime(2024, 9, 2, 12, 28, 0)  # Adjusted to match ISODate format in MongoDB

end_time = datetime.now()  # Set end time to the current time


# Execute queries

get_vehicle_count_least_edge(start_time, end_time)

get_highest_avg_speed_edge(start_time, end_time)

get_longest_distance(start_time, end_time)
```

Συνδέεται με τη MongoDB χρησιμοποιώντας το URI mongodb://localhost:27017/ και επιλέγει τη βάση δεδομένων vehicle_db.

Χρησιμοποιεί την εντολή aggregate για να βρει τον σύνδεσμο (link) με τον μικρότερο αριθμό οχημάτων κατά τη διάρκεια ενός συγκεκριμένου χρονικού διαστήματος.

Φιλτράρει τα δεδομένα με βάση το χρονικό διάστημα (start_time και end_time).

Ομαδοποιεί τα δεδομένα κατά link και υπολογίζει το συνολικό πλήθος οχημάτων για κάθε σύνδεσμο.

Ταξινομεί τα δεδομένα με βάση το πλήθος οχημάτων σε αύξουσα σειρά και επιλέγει το σύνδεσμο με το μικρότερο πλήθος.

Αναζητά τον σύνδεσμο με την υψηλότερη μέση ταχύτητα κατά τη διάρκεια του ίδιου χρονικού διαστήματος.

Φιλτράρει τα δεδομένα με βάση το χρονικό διάστημα.

Ομαδοποιεί τα δεδομένα κατά link και υπολογίζει τον μέσο όρο της ταχύτητας για κάθε σύνδεσμο.

Ταξινομεί τα δεδομένα με βάση την μέση ταχύτητα σε φθίνουσα σειρά και επιλέγει το σύνδεσμο με την υψηλότερη μέση ταχύτητα.

Υπολογίζει την συνολική απόσταση που διένυσε ένα όχημα (ως ταχύτητα * χρόνο) για κάθε σύνδεσμο και αναζητά τον σύνδεσμο με τη μεγαλύτερη απόσταση.

Φιλτράρει τα δεδομένα ώστε να συμπεριλάβει μόνο όσες εγγραφές έχουν ταχύτητα μεγαλύτερη από το μηδέν.

Ομαδοποιεί τα δεδομένα κατά σύνδεσμο, όνομα και προέλευση, και υπολογίζει τη συνολική απόσταση.

Ταξινομεί τα δεδομένα με βάση την απόσταση σε φθίνουσα σειρά και επιστρέφει το σύνδεσμο με τη μεγαλύτερη συνολική απόσταση.

```
dam@DESKTOP-H5HSJ17:~$ python3 mongo_queries.py
Querying vehicle count for least edge...
Link with the least number of vehicles: N3I3 with count 6
Querying highest average speed for edge...
Link with the highest average speed: E1I4 with average speed 44.875
Querying longest distance...
Link: E1I4
Window: {'start': datetime.datetime(2024, 9, 19, 16, 25), 'end': datetime.datetime(2024, 9, 19, 16, 26)}
Average Speed (vspeed): 50.0 km/h
Duration Since Start: 74 seconds
Vehicle Count (vcount): 1
Longest Total Distance: 3700.0 meters
dam@DESKTOP-H5HSJ17:~$ python3 mongo_queries.py
Querying vehicle count for least edge...
Link with the least number of vehicles: N3I3 with count 21
Querying highest average speed for edge...
Link with the highest average speed: E1I4 with average speed 44.895833333333336
Querying longest distance...
Link: E1I4
Window: {'start': datetime.datetime(2024, 9, 19, 16, 25), 'end': datetime.datetime(2024, 9, 19, 16, 26)}
Average Speed (vspeed): 50.0 km/h
Duration Since Start: 83 seconds
Vehicle Count (vcount): 1
Longest Total Distance: 4150.0 meters
dam@DESKTOP-H5HSJ17:~$ python3 mongo_queries.py
Querying vehicle count for least edge...
Link with the least number of vehicles: N3I3 with count 36
Link with the least number of vehicles: I3I2 with count 36
Querying highest average speed for edge...
Link with the highest average speed: E1I4 with average speed 44.90384615384615
Querying longest distance...
Link: E1I4
Window: {'start': datetime.datetime(2024, 9, 19, 16, 25), 'end': datetime.datetime(2024, 9, 19, 16, 26)}
Average Speed (vspeed): 50.0 km/h
Duration Since Start: 89 seconds
Vehicle Count (vcount): 1
Longest Total Distance: 4450.0 meters
```

Σχολιασμός αποτελεσμάτων

Η συγκεκριμένη εργασία παρείχε μια ολοκληρωμένη εμπειρία στην ανάπτυξη και αξιοποίηση συστημάτων επεξεργασίας δεδομένων σε πραγματικό χρόνο, χρησιμοποιώντας σύγχρονες τεχνολογίες όπως το Apache Kafka και το Apache Spark. Η δυνατότητα παραγωγής και κατανάλωσης δεδομένων μέσω Kafka καθώς και η επεξεργασία τους με το Spark μας επέτρεψε να διαχειριστούμε μεγάλους όγκους δεδομένων σε πραγματικό χρόνο, επιτυγχάνοντας ακριβείς μετρήσεις και αναλύσεις.

Επιπλέον, η σύνδεση με την MongoDB και η αποθήκευση τόσο των ακατέργαστων όσο και των επεξεργασμένων δεδομένων προσέφερε μια αξιόπιστη και επεκτάσιμη λύση για την αποθήκευση και ανάλυση δεδομένων. Με τη χρήση συγκεκριμένων queries, καταφέραμε να αντλήσουμε πληροφορίες όπως η μέση ταχύτητα, ο αριθμός οχημάτων και η συνολική απόσταση για κάθε οδική σύνδεση, καθιστώντας δυνατή την εξαγωγή πολύτιμων συμπερασμάτων από τα δεδομένα της εξομοίωσης.

Η διαδικασία ανέδειξε τις προκλήσεις που αφορούν τον συγχρονισμό δεδομένων, την απόδοση των συστημάτων σε πραγματικό χρόνο και την ανάγκη για βέλτιστη διαχείριση των ροών δεδομένων. Συνολικά, η εμπειρία αυτή μας προσέφερε πολύτιμες γνώσεις και ενίσχυσε τις δεξιότητές μας στη διαχείριση συστημάτων μεγάλων δεδομένων.

Βιβλιογραφία

[1] https://kafka.apache.org/quickstart

[2] https://toruseo.jp/UXsim/docs/index.html

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

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

[5] S. Park and J. -H. Huh, "A Study on Big Data Collecting and Utilizing Smart Factory Based Grid Networking Big Data Using Apache Kafka," in IEEE Access, vol. 11, pp. 96131-96142, 2023, doi: 10.1109/ACCESS.2023.3305586.