Συστήματα Διαχείρισης Δεδομένων Μεγάλου Όγκου

Εργαστηριακή Άσκηση 2023/24

| **Όνομα** | **Επώνυμο** | **ΑΜ** |
| --- | --- | --- |
| Ιάσονας | Παυλόπουλος | 1084565 |
| Ρουμπίνη-Μαρία | Αγγουρά | 1084634 |

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    13 / 06 / 2024 | Υπογραφή    13 / 06 / 2024 |
| --- | --- |

**Συνημμένα αρχεία κώδικα**

Μαζί με την παρούσα αναφορά υποβάλλουμε τα παρακάτω αρχεία κώδικα

| Αρχείο | Αφορά το ερώτημα | Περιγραφή/Σχόλιο |
| --- | --- | --- |
| simulation.py | 1 | Τρέχει το simulation και παράγει το αρχείο sim\_data.csv που έχει τα δεδομένα του Simulation |
| sim\_data.csv (Παραγόμενο από simulation.py) | 1 | Περιέχει τα δεδομένα του Simulation |
| producer.py | 1 | Είναι ο Producer που στέλνει τα δεδομένα στον Kafka Broker |
| consumer.py | 1 | Είναι ο Consumer που καταναλώνει τα δεδομένα από τον Kafka Broker |
| spark.py | 2-3 | Είναι το αρχείο που πραγματοποιεί την κατανάλωση και την επεξεργασία σε Spark, αλλά και την αποθήκευση στην MongoDB |
| mongoQueries.py | 3 | Είναι το αρχείο που έχει τα Queries προς την MongoDB και εμφανίζει και τα αποτελέσματα |

# Τεχνικά χαρακτηριστικά περιβάλλοντος λειτουργίας

| **Χαρακτηριστικό** | **Τιμή** |
| --- | --- |
| CPU model | Intel i5-9300H |
| CPU clock speed | 2.4Ghz Base, 4Ghz Boost |
| Physical CPU cores | 4 |
| Logical CPU cores | 8 |
| RAM | 16GB |
| Secondary Storage Type | SSD |

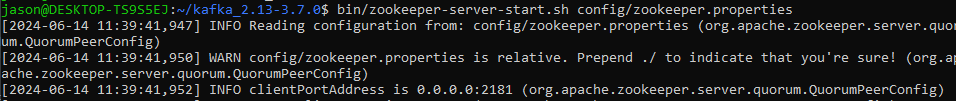
# Ερώτημα 1: Παραγωγή δεδομένων

Για την δημιουργία δεδομένων εξομοίωσης δημιουργούμε ένα αντικείμενο World το οποίο αποτελεί το σενάριό μας. Θέτουμε αρχικές παραμέτρους, με βασικότερη την tmax = 3600 (δηλαδή 1 ώρα simulation) και την deltan = 5 (δηλαδή το χρόνο ανάμεσα σε δύο στιγμιότυπα του simulation). Έπειτα, δημιουργούμε τους κόμβους και τις ακμές του (οδικού) δικτύου μας, θέτοντας τις παραμέτρους που επιθυμούμε. Έπειτα, θέτουμε τις «απαιτήσεις του δικτύου», δηλαδή το πόσα αμάξια θα ξεκινούν από τους αρχικούς κόμβους σε ποια χρονική στιγμή, προς ποιο τελικό κόμβο και τον ρυθμό εμφάνισής τους. Τέλος, ξεκινάμε την εξομοίωση και αποθηκεύουμε τα δεδομένα στο sim\_data.csv.

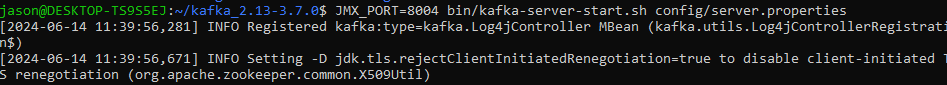
Στην συνέχεια, στο αρχείο producer.py, δημιουργείται το αντικείμενο KafkaProducer. Στην συνέχεια, παίρνει τα δεδομένα από το αρχείο sim\_data.csv και δημιουργεί ένα json μήνυμα για κάθε εγγραφή με τα απαραίτητα στοιχεία όπως αναφέρεται στην εκφώνηση. Τέλος, για κάθε χρονική στιγμή deltan αποστέλλει τα json μηνύμα που αντιστοιχεί σε αυτήν την χρονική στιγμή deltan στον Kafka Broker. Κάνει δηλαδή μία ομαδοποίηση με βάση το deltan.

Τέλος, στο αρχείο consumer.py, δημιουργείται το αντικείμενο KafkaConsumer, το οποίο απλά κάνει print αυτό το μήνυμα που περνάει μέσα από τον Kafka Broker.

Τρέχουμε την παρακάτω εντολή για τον zookeeper-server και τον ενεργοποιούμε.



Έπειτα τρέχουμε και τον kafka-server και παρατηρούμε στο 2ο Screenshot, ότι έχει συνδεθεί στον server zookeeper.



Στα παρακάτω screenshots μπορούμε να δούμε την λειτουργία consumer/producer για τα πρώτα 14 μηνύματα:

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

# Ερώτημα 2: Κατανάλωση και επεξεργασία με Spark

Αρχικά, ορίζουμε το schema των δεδομένων που θα λαμβάνουμε από τον kafka Broker. έπειτα, δημιουργούμε ένα SparkSession και ένα readStream το οποίο θα διαβάζει από το topic vehicle\_positions. Έπειτα κάνουμε ένα aggregation των δεδομένων που εισέρχονται. Συγκεκριμένα, κάνουμε Group By τα δεδομένα ως προς time και link. Επίσης, προσθέτουμε τις στήλες vcount και vspeed οι οποίες δημιουργούνται παίρνοντας το count και το avg των ζητούμενων δεδομένων αντίστοιχα. Τέλος, παίρνουμε τα επεξεργασμένα δεδομένα και τα εμφανίζουμε στο console, αλλά επίσης τα γράφουμε και στην MongoDB μας με την χρήση της foreachBatch().

Για την προβολή των δεδομένων μέσα στην βάση δεδομένων μας, χρησιμοποιούμε το mongodb-compass.

Στο παρακάτω screeshot βλέπουμε ότι το Spark επιτυχώς παραλαμβάνει τα δεδομένα από το Kafka Broker και επίσης επιτυχώς τα επεξεργάζεται και τα εμφανίζει. Για λόγους διευκρίνισης, σε κάθε batch εμφανίζονται δεδομένα πολλών χρονικών στιγμών deltan, καθώς το σύστημα δεν προλαβαίνει να επεξεργαστεί αρκετά γρήγορα την ροή δεδομένων για κάθε deltan ξεχωριστά.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, κατάλογος

Περιγραφή που δημιουργήθηκε αυτόματα

# Ερώτημα 3: Αποθήκευση σε MongoDB

Ο σχεδιασμός της βάσης μπορεί να αποτυπωθεί από τις εκτελέσεις των παρακάτω εντολών:

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματαΕικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Οι εντολές για δημιουργία της βάσης και των συλλογών είναι οι εξής:

use project (Για την δημιουργία της βάσης, παρόλο που χωρίς συλλογές δεν έχει καμία λειτουργία)

db.createCollection(“vehicle\_raw\_data”)

db.createCollection(“vehicle\_processed\_data”)

Παρατηρούμε, πως η εκχώρηση στην MongoDB είναι επιτυχής. Συγκεκριμένα, για τα raw data βλέπουμε ότι ο αριθμός των γραμμών στο excel συμφωνεί με αυτόν στην συλλογή vehicle\_raw\_data (7480 + 1 η αρχική γραμμή με τα labels των στηλών)

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, αριθμός

Περιγραφή που δημιουργήθηκε αυτόματαΕικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, αριθμός

Περιγραφή που δημιουργήθηκε αυτόματα

Στην συνέχεια για τον έλεγχο των αποθηκευμένων επεξεργασμένων δεδομένων, δημιουργήσαμε ένα python script (το οποίο δεν συμπεριλήφθηκε καθώς δεν ήταν ζητούμενο) για να πάρουμε τον σωστό αριθμό των επεξεργασμένων δεδομένων. Παραθέτουμε τον κώδικα:

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα



Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Εκτελώντας το αρχείο mongoQueries.py το οποίο παίρνει ορίσματα από το command line το “Start Date” και το “End Date” παίρνουμε τα εξής αποτελέσματα:

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, λογισμικό

Περιγραφή που δημιουργήθηκε αυτόματα

# Σχολιασμός αποτελεσμάτων

Το συγκεκριμένο project φάνηκε και στα δύο μέλη της ομάδας ιδιαίτερα ενδιαφέρον. Είχαμε την ευκαιρία να καταλάβουμε και να πραγματοποιήσουμε μία ροή και επεξεργασία δεδομένων (σχεδόν σε realtime χρόνο), αλλά και να τα αποθηκεύσουμε σε μία βάση δεδομένων. Η πρακτικότητα και η εφαρμοσιμότητα αυτών των τεχνικών που χρησιμοποιήσαμε μας ενθουσίασε και με την διεκπεραίωση της εργασίας αντλήσαμε κάποιες βασικές και πρακτικές γνώσεις από τον τομέα της συλλογής, επεξεργασίας και αποθήκευσης των μεγάλων δεδομένων.

# Βιβλιογραφία

* [Οδηγίες και Παραδείγματα του UXsim](https://toruseo.jp/UXsim/docs/index.html)
* [Κατανάλωση δεδομένων από Kafka και επεξεργασία JSON σε Spark](https://sandeepkattepogu.medium.com/python-spark-transformations-on-kafka-data-8a19b498b32c)
* [Εγκατάσταση και Λειτουργίες MongoDB](https://www.mongodb.com/pt-br/docs/manual/installation/)
* [Εγκατάσταση και Λειτουργίες MongoDB Compass](https://www.mongodb.com/pt-br/docs/compass/current/)
* [Παραδείγματα εκτέλεσης MongoDB Queries](https://www.mongodb.com/pt-br/docs/manual/tutorial/query-documents/)