Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Προχωρημένα Θέματα Βάσεων Δεδομένων
Ροή Λ

Εξαμηνιαία Εργασία: Χρήση του Apache Spark στις Βάσεις Δεδομένων

Ομάδα
Άγγελος Σταυρόπουλος 03109643

Μέρος 1: Υπολογισμός Αναλυτικών Ερωτημάτων με τα APIs του Apache Spark

Ζητούμενο 1:
Φορτώνουμε τα 3 δοθέντα αρχεία στο HDFS με την εντολή hdfs dfs -put.
Συγκεκριμένα, εκτελούμε:

hdfs dfs -put movies.csv movie_genres.csv ratings.csv /atds_project_21

Ζητούμενο 2:
Μετατρέπουμε τα csv αρχεία της εκφώνησης σε parquet κάνοντας χρήση της συνάρτησης write.parquet() των spark dataframes. Συγκεκριμένα, χρησιμοποιούμε το script csv_to_parquet.py που βρίσκεται στον φάκελο code, το οποίο υποβάλλουμε στο spark με ορίσματα τα ονόματα του csv αρχείου εισόδου και του parquet αρχείου εξόδου. Ενδεικτικά:

spark-submit csv_to_parquet.py /atds_project_21/movies.csv /atds_project_21/movies.parquet

Ζητούμενο 3:

- **Q1**

Για να βρούμε την ταινία με το μεγαλύτερο κέρδος για κάθε χρονιά από το 2000 και μετά, αρχικά φιλτράρουμε τις ταινίες και κρατάμε αυτές που έχουν βγει από το 2000 και μετά, καθώς επίσης έχουν μη-μηδενικές τιμές στο κόστος και τα έσοδα. Στη συνέχεια υπολογίζουμε για κάθε ταινία το κόστος βάσει του τύπου που μας δίνεται και μετά χρησιμοποιώντας το έτος κυκλοφορίας ως κλειδί, κρατάμε το ζεύγος (τίτλος, κέρδος) για το οποιο το κέρδος είναι μέγιστο.
Ο ψευδοκώδικας των συναρτήσεων map & reduce που χρησιμοποιήθηκαν είναι:

```
map(line):
for each line:
        parse line;
        emit(title, year, cost, revenue)

map(key, value):
for each line:
        filter values;
        compute profit;
        emit(title, year, profit)

aggregateByKey(key, value):             #(year, (title, profit))
for i in values:
        find max(profit)
emit(year, title, max_profit)
```

- **Q2**

Για να βρεθεί το ποσοστό των χρηστών που έχουν δώσει σε ταινίες μέση βαθμολογία μεγαλύτερη του 3, αρχικά υπολογίζουμε τον συνολικό αριθμό των χρηστών. Κατόπιν, από τα δεδομένα εισόδου υπολογίζουμε τη μέση βαθμολογία για κάθε χρήστη και καρατάμε

αυτούς που η μέση βαθμολογία τους είναι μεγαλύτερη του 3. Τέλος, διαιρούμε τους 2 αριθμούς για να προκύψει το ζητούμενο ποσοστό.
Ο ψευδοκώδικας των συναρτήσεων map & reduce που χρησιμοποιήθηκαν είναι:

```
map(line):
for each line:
        parse line;
        emit(user_id, rating)

reduce(key, value):             #(user, rating)
count keys;
emit total_users

aggregateByKey(key, value):
for i in values:
        count entries and calculate sum;
        calculate average_rating;
        filter average_rating >3);
        emit (user_id, average_rating)

reduce(key, value):             #(user, average_rating_above_3)
count keys;
emit total_users_above_3
```

- **Q3**

Για να βρεθεί η μέση βαθμολογία ανά είδος ταινίας, αρχικά υπολογίζουμε για κάθε ταινία τη μέση βαθμολογία της, όπως αυτή προκύπτει από τις αξιολογήσεις των χρηστών. Στη συνέχεια, κάνουμε join με την είσοδο που αφορά στα είδη των ταινιών, προκειμένου να προσθέσουμε την πληροφορία του είδους σε κάθε γραμμή του συνόλου των ταινιών. Το join εφαρμόζεται ως γινόμενο στις εγγραφές που matchαρουν, επομένως αν μια ταινία αντιστοιχεί σε περισσότερα από ένα είδος, προσμετράται σε κάθε ένα από αυτά τα είδη. Έπειτα, για κάθε είδος υπολογίζουμε τον αριθμό των ταινιών που ανήκουν σ' αυτό, καθώς και το άθροισμα των αντίστοιχων μέσων βαθμολογιών των ταινιών. Διαιρώντας τα 2 αυτά νούμερα προκύπτει η μέση βαθμολογία του είδους.
Ο ψευδοκώδικας των συναρτήσεων map & reduce που χρησιμοποιήθηκαν είναι:

```
map(line):
for each line:
        parse line;
        emit(movie_id, genre)           #for movie_genres.csv input
        emit(movie_id, rating)          #for ratings.csv input

aggregateByKey(key, value):             #(movie_id, rating)
for i in values:
        count entries and calculate sum;
        calculate average_rating;
        emit (movie_id, average_rating)

join(movies_average_ratings, genres) on movie_id
```

```
aggregateByKey(key, value):                #(genre, average_rating)
for i in values:
        count entries and calculate sum;
        calculate average_rating;
        emit (genre, average_rating)
```

- **Q4**

Για να βρεθεί το μέσο μήκος περίληψης ταινίας του είδους Drama ανά 5ετία από το 2000 και μετά, αρχικά φιλτράρουμε τις ταινίες και κρατάμε αυτές των οποίων η ημερομηνία κυκλοφορίας είναι από το 2000 και μετά, καθως επίσης αντιστοιχίζουμε τη συγκεκριμένη ημερομηνία στην κατάλληλη 5ετία. Κάνουμε join τις ταινίες με τα είδη και κρατάμε μόνο αυτές που ανήκουν στο είδος Drama. Τέλος, για κάθε 5ετία υπολογίζουμε αριθμό ταινιών και αθροιστικό μήκος περιλήψεων, τα οποία και διαιρούμε προκειμένου να προκύψει το μέσο μήκος περίληψης ανά 5ετία.
Ο ψευδοκώδικας των συναρτήσεων map & reduce που χρησιμοποιήθηκαν είναι:

```
map(line):
for each line:
        parse line;
        emit(movie_id, summary, year)   #for movies.csv input
        emit(movie_id, genre)                   #for movie_genres.csv input

map(key, value):
for each(movie_id, summary, year):
        filter values;
        calculate 5etia;
        calculate summary length;
        emit(movie_id, sum_len, 5etia)

map(key, value):
for each(movie_id, genre):
        filter values;
        emit(movie_id, drama)

join(movies, drama_genre) on movie_id

aggregateByKey(key, value):                     #(5etia, sum_len)
for in in values:
        count entries and calculate sum;
        calculate average_length;
        emit (5etia, average_length)
```

- **Q5**

Το ερώτημα ερμηνεύθηκε ως ότι η περισσότερο και λιγότερο αγαπημένη ταινία του χρήστη θα πρέπει να ανήκουν κάθε φορά στο αντίστοιχο genre στο οποίο προκύπτει ως αυτός με τις περισσότερες κριτικές. Αρχικά κάνουμε join τις αξιολογήσεις με τα genres και για κάθε key (genre, user) υπολογίζουμε το πλήθος των αξιολογήσεων. Έπειτα χρησιμοποιώντας ως key το κάθε genre βρίσκουμε τον χρήστη με τις περισσότερες

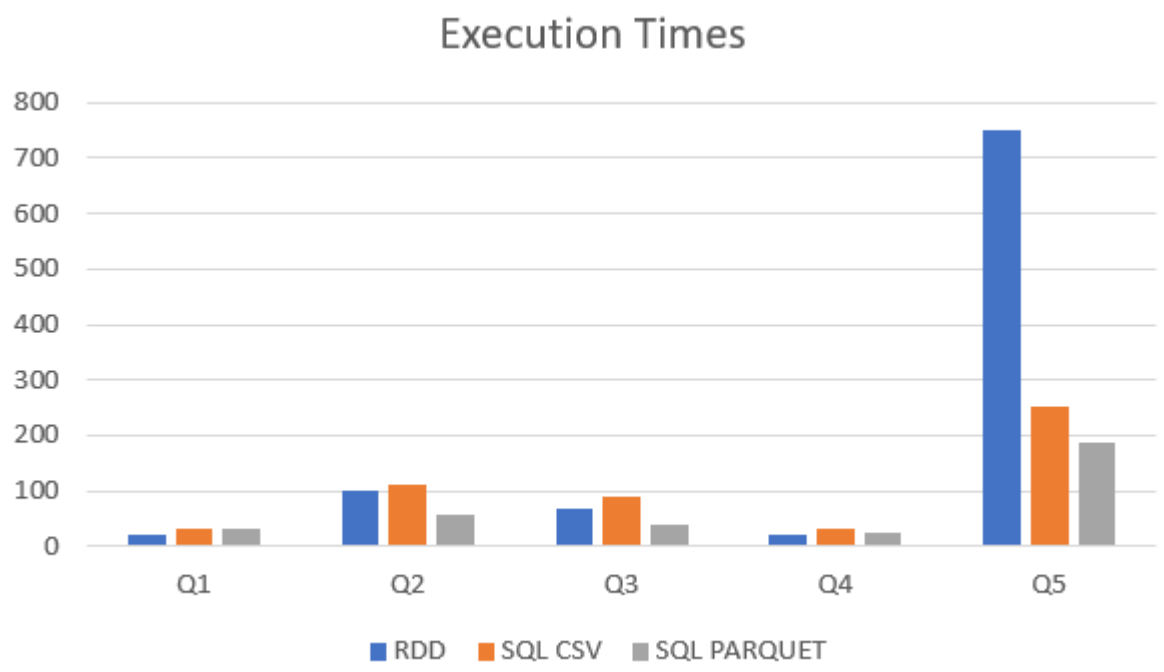
αξιολογήσεις. Στη συνέχεια, κάνουμε join τις αξιολογήσεις με τα genres και τις ταινίες και για κάθε key (genre, user) βρίσκουμε την περισσότερο/λιγότερο αγαπημένη του ταινία συμφωνα με τα δοθέντα κριτήρια. Τέλος, για κάθε ζεύγος (genre, user_with_most_ratings) επιλέγουμε μέσω join από τα αντίστοιχα RDDs την περισσότερο και λιγότερο αγαπημένη του ταινία από το συγκεκριμένο genre.
Ο ψευδοκώδικας των συναρτήσεων map & reduce που χρησιμοποιήθηκαν είναι:

```
map(line):
for each line:
        parse line;
        emit(movie_id, title, popularity)   #for movies.csv input
        emit(movie_id, genre)                       #for movie_genres.csv input
        emit(user_id, movie_id, rating)    #for ratings.csv

join(genres, ratings) on movie_id

groupByKey(key, value):                     #((genre, user) rating)
for i in values:
        count;
        emit((genre, user), no_of_ratings)

aggregateByKey(key, value):                         #(genre, (user, no_of_ratings))
for i in values:
        find max(no_of_ratings)
emit ((genre, user) max_ratings)

join(movies, genres, ratings) on movie_id

aggregateByKey(key, value):                         #((genre, user), (title, rating, popularity))
for i in values:
        find max(rating) or max(popularity) if equal ratings
emit ((genre, user) (title, max_rating))

aggregateByKey(key, value):                         #((genre, user), (title, rating, popularity))
for i in values:
        find min(rating) or max(popularity) if equal ratings
emit ((genre, user) (title, min_rating))

join(max_ratings, favorite_movie, least_favorite_movie) on (genre, user)
```

Οι πλήρεις υλοποιήσεις σε κώδικα των παραπάνω ερωτημάτων, τόσο με τη χρήση του RDD API όσο και με Spark SQL βρίσκονται στο φάκελο code, στα αντίστοιχα αρχεία.

Τα αρχεία εξόδου των υλοποιήσεων, με τα τελικά αποτελέσματα στη μορφή που ζητείται στην εκφώνηση μπορούν να βρεθούν στον συνημμένο φάκελο output, οπως επίσης και στο hdfs site στο path /atds_project_21/output.

Ζητούμενο 4:
Το ζητούμενο διάγραμμα που απεικονίζει τους χρονους se sec για κάθε υλοποίηση είναι το ακόλουθο:



Παρατηρούμε πως στα ερωτήματα που δεν απαιτούν μεγάλη αναλυτική επεξεργασία οι χρόνοι μεταξύ RDD & SQL δεν παρουσιάζουν μεγάλες διαφορές. Πολλές φορές η υλοποίηση με το spark RDD αποδεικνύεται πιο γρήγορη, πράγμα που ενδεχομένως οφείλεται στο ότι η συγκεκριμένη υλοποίηση βρίσκεται πιο κοντά στα δεδομένα, χωρίς τα επίπεδα αφαίρεσης που εισάγει η SQL, με αποτέλεσμα ορισμένες ενέργειες να εκτελούνται πιο άμεσα χωρίς το overhead της αφαίρεσης. Στο ερώτημα 5, στο οποίο η αναλυτική επεξεργασία είναι ιδιαίτερα αυξημένη, παρατηρούμε πως υπερέχουν οι υλοποιήσεις σε Spark SQL, καθώς παρέχουν περισσότερες δυνατότητες επεξεργασίας των δεδομένων, χωρίς να δαπαναται υπολογιστική ισχύς σε αλλεπάλληλες προσπελάσεις τους, προκειμένου να υποστούν επεξεργασία από τις χαμηλού επιπέδου ενέργειες που μας παρέχονται από το RDD API.
Επίσης, παρατηρούμε πως οι υλοποιήσεις της SQL που έχουν ως είσοδο τα αρχεία parquet είναι πιο γρήγορες συγκριτικά με αυτές που διαβάζουν τα csv αρχεία, καθώς τα αρχεία parquet είναι φτιαγμένα ώστε να βελτιστοποιούν το I/O και τις μετέπειτα μετακινήσεις των δεδομένων, επομένως -ειδικά σε ερωτήματα με μεγάλο όγκο δεδομένων- οι συγκεκριμένες υλοποιήσεις κερδίζουν μεγάλη διαφορα από τις υπόλοιπες.

Τέλος, στις υλοποιήσεις με είσοδο parquet αρχεία δεν υπάρχει ανάγκη για τη χρήση του inferSchema, καθώς το schema των δεδομένων σε αυτή την περίπτωση “περιλαμβάνεται” στη δομή του αρχείου, επομένως γίνεται evaluate αυτόματα κατά το διάβασμα των δεδομένων.