ΕΡΓΑΣΙΑ ΣΤΑ ΠΡΟΧΩΡΗΜΕΝΑ ΒΑΣΕΩΝ

Ονοματεπώνυμο: Γιώργος Καρόζης
ΑΜ-03114878

Η εργασία πραγματεύεται ενασχόληση με pysparkSql.

Στο πρώτο ερώτημα μας ζητείται η φόρτωση των τριών CSV αρχείων στο hadoop. Για τον σκοπό αυτό, καταρχάς, αποθηκεύσαμε τα τρία αρχεία στο home directory του master μέσω της εντολής wget --no-check-certificate ‘http://www.cslab.ntua.gr/courses/atds/movie_data.tar.gz’ -O movie_data.tar.gz και αποσυμπίεση μέσω της εντολής tar -zxf movie_data.tar.gz. Στην συνέχεια για την φόρτωση των δεδομένων στο hadoop χρησιμοποιήσαμε την εντολή hadoop fs -mkdir hdfs://master:9000/Excercise όπου δημιουργήσαμε τον φάκελο Excercise στον base folder (master:9000) όπου θα βάλουμε τα τρία αρχεία και μέσω της εντολής -put βάλαμε το εκάστοτε csv αρχείο.

Όσον αφορά το δεύτερο ερώτημα πολύτιμη ήταν η βοήθεια της ιστοσελίδας του spark https://spark.apache.org/docs/latest/sql-data-sources-parquet.html όπου προγραμματιστικά φόρτωσα τα 3 csv αρχεία μέσω της εντολής spark.read.csv(“hdfs://master9000/Excercise/fileName.csv”) αφού πρώτα είχα αρχικοποιήσει την μεταβλητή spark σε ένα καινούργιο session. Εν συνεχεία αποθήκευσα τα αρχεία σε μορφή .parquet στο hdfs μέσω της αντίστοιχης εντολής spark.write(“hdfs://master9000/Excercise/fileName.parquet”). Ο κώδικας γράφτηκε σε Python σε ένα αρχείο με όνομα CreateParquet.py τον οποίο και έτρεξα μέσω της εντολής spark-submit CreateParquet.py.

Στο τρίτο ερώτημα θα αναλύσω λίγο τα queries και με ποιες τεχνικές τα επίλυσα. Στο πρώτο query μας ζητήθηκε η ταινία με το μεγαλύτερο κέρδος από το 2000 και μετά αγνοώντας εγγραφές με μηδενικά έσοδα και κόστη, καθώς και κενή ημερομηνία έκδοσης.ο ψευδοκώδικας είναι ο εξής:

```
Map(key, value){
    line = String(value);
    parts = line.split(“,”);
    deleted_dataset = parts.remove() // παίρνω την ώρα φιλτράροντας το dataset
    get_year = deleted_dataset[3][:4] // παίρνω τα 4 πρώτα του string για year
    profit = Get_profit(deleted_dataset[5],deleted_dataset[6])
    emit(get_year, profit)
}

 Reduce(key, list(value)):
     max = 0;
     for each element in list(value){
          if (element > max) max = element;
     }
     emit(key,max)
  }
```

Στο δεύτερο query μας ζητείται το ποσοστό των χρηστών που έχει βαθμολογήσει τις ταινιές με βαθμός μεγαλύτερο του 3.

```
Map(key, value){
    line = String(value);
    parts = line.split(“,”);
    Id = get_int(parts[0]);
    rate = parts[2];
    emit(id, rate);
}

Reduce(key,list(value)){
    sum = 0;
    count = 0;
    for each element in list(value){
        sum+=value;
        count++;
    }
    avg = sum/count;
    emit(id, avg);
}
Map(key,avg,NoUsers){
    count=0;
    for each key {
        if avg>3 count++
    }
    emit(count);
}
```

Στο τρίτο ερώτημα μας ζητείται για κάθε κατηγορία ταινίας η μέση βαθμολογία της, καθώς και ο αριθμός των ταινιών που υπάγονται στην κατηγορία. Έτσι αρχικά επεξεργαζόμαστε τον πίνακα ratings. Μέσα σε αυτόν βρίσκουμε την μέση βαθμολογία για κάθε ταινία και την κάνουμε emit με κλειδί το movieID.

#1

```
Map(key,value){
    value.toString();
    String parts = record.split(“,”);
    String key2 = parts[1]; //choose movieID as key
    String value2 = parts[2]; // choose movieRating as value
    emit(key2,value2)
}
```

#2

```
ReduceByKey(movieID, rating_list){
    count = 0;
    sum = 0;
```

```
        for i in list(rating){
            sum = sum + i;
            count = count+1;
        }
        avg = sum/count;
        emit(movieID,avg);
    }
```

Προς το παρόν έχουμε βρει για κάθε ταινία την μέση βαθμολογία της. Πάμε τώρα να βρούμε την μέση βαθμολογία κάθε κατηγορίας με την βοήθεια του movie_genres.csv .

#3

```
    Map(key,value){
        value.toString();
        String parts = record.split(“,”);
        String key2 = parts[0]; //choose movieID as key
        String value2 = parts[1]; // choose Category as value
        emit(movieID,Category)
    }
```

Και τώρα κάνουμε join τον #2 και #3 με βάση το κοινό τους κλειδί.

```
    Map(movieID,tuple_list){
    #tuple_list is a list that each element has two values (avg,Category)
        for i in tuple_list:
           String key2 = i[1]; //choose Category as key
           String value2 = i[0]; // choose movie_avg as value
           emit(key2,value2)
    }
```

Και τώρα κάνουμε reduceByKey με κλειδί την κάθε κατηγορία για να βρούμε τον μέσο όρο κάθε κατηγορίας. Η κάθε κατηγορία εμφανίζεται όσες είναι οι εγγραφές για κάθε ταινία της κατηγορίας. Οπότε εύκολα μπορούμε να βρούμε τις εγγραφές κάθε κατηορίας.

```
    ReduceByKey(Category, movie_avg_list){
        count = 0;
        sum = 0;
        for i in movie_avg_list{
            sum = sum + i;
            count = count+1;
        }
        avg = sum/count;
        emit(movieID,avg,count);
    }
```

Στο τέταρτο ερώτημα μας ζητείται για την κατηγορία Drama για κάθε πενταετία το μέσο μήκος περίληψης της κάθε ταινίας (από το 2000 και μετά). Αρχικά παίρνουμε από το movies.csv το μήκος της κάθε ταινίας , καθώς και την ημερομηνία κυκλοφορίας τους.

```
#1
    Map(key,value){
        value.toString();
        String parts = record.split(“,”);
        String key2 = parts[0]; //choose movieID as key
        String value1 = get_dec(parts[3]); // choose movieRating as value
        String value2 = get_length(parts[4]);
        value = (value1, value2);
        emit(key2,value);
    }
#2
    Map(key,value){ //( movie_genres.csv)
        value.toString();
        String parts = record.split(“,”);
        String key2 = parts[0]; //choose movieID as key
        String value2 = parts[1]; // choose Category as value
        emit(movieID,Category)
    }
#3
    ReduceByKey(movieID, rating_list){
        count = 0;
        sum = 0;
        for i in list(rating){
            sum = sum + i;
            count = count+1;
        }
        avg = sum/count;
        emit(movieID,avg);
    }
```

Στο πέμπτο και τελευταίο ερώτημα ψάχνουμε να βρούμε για κάθε είδος ταινίας τον χρήστη με τις περισσότερες κριτικές μαζί με την λιγότερο και περισσότερο αγαπημένη του ταινία. Σε περίπτωση που ο χρήστης έχει παραπάνω από μία τέτοια ταινία επιλέγει την δημοφιλέστερη.

```
#1
    Map(key,value){ // movie_genres.csv
        value.toString();
        String parts = record.split(“,”);
```

```
        String key2 = parts[0]; //choose movieID as key
        String value2 = parts[1]; // choose Category as value
        emit(key2,value2);
    }
#2
    Map(key,value){ // ratings.csv
        value.toString();
        String parts = record.split(“,”);
        String key2 = parts[1]; //choose movieID as key
        String value1 = parts[0]; // choose User as value1
        String value2 = parts[2]; //choose Rate as value2
        value = (value1,value2);
        emit(key2,value);
    }

#3
    Map(key,value){ // movies.csv
        value.toString();
        String parts = record.split(“,”);
        String key2 = parts[0]; //choose movieID as key
        String value1 = parts[1]; // choose Title as value1
        String value2 = parts[2]; //choose Popularity as value2
        value = (value1,value2);
        emit(key2,value);
    }
```

Στην συνέχεια κάνω join με βάση το movieID τα 3 παραπάνω maps. Έτσι κάνω map με βάση την κατηγορία και τον χρήστη και έχω:

```
#4
    Map(key,values_list){ // (Category,(User,Rate),(Title,Popularity)) element of list
        for i in values_list:
            key2 = (i[0],i[1][0]); //(Category,User) as key
            value2 = (i[1][1],i[2][0],i[2][1]); (Rate,Title,Popularity) as value
            emit(key2, value2);
    }
```

Κάνουμε reduceByKey για να βρούμε για κάθε χρήστη και κάθε κατηγορία τις κριτικές και μετά ένα δεύτερο για να βρούμε για κάθε κατηγορία τον χρήστη με τις μέγιστες κριτικες.

```
#5
    ReduceByKey(Categor&User, values_list){
        count = 0;
        for i in values_list{
            count = count+1;
        }
```

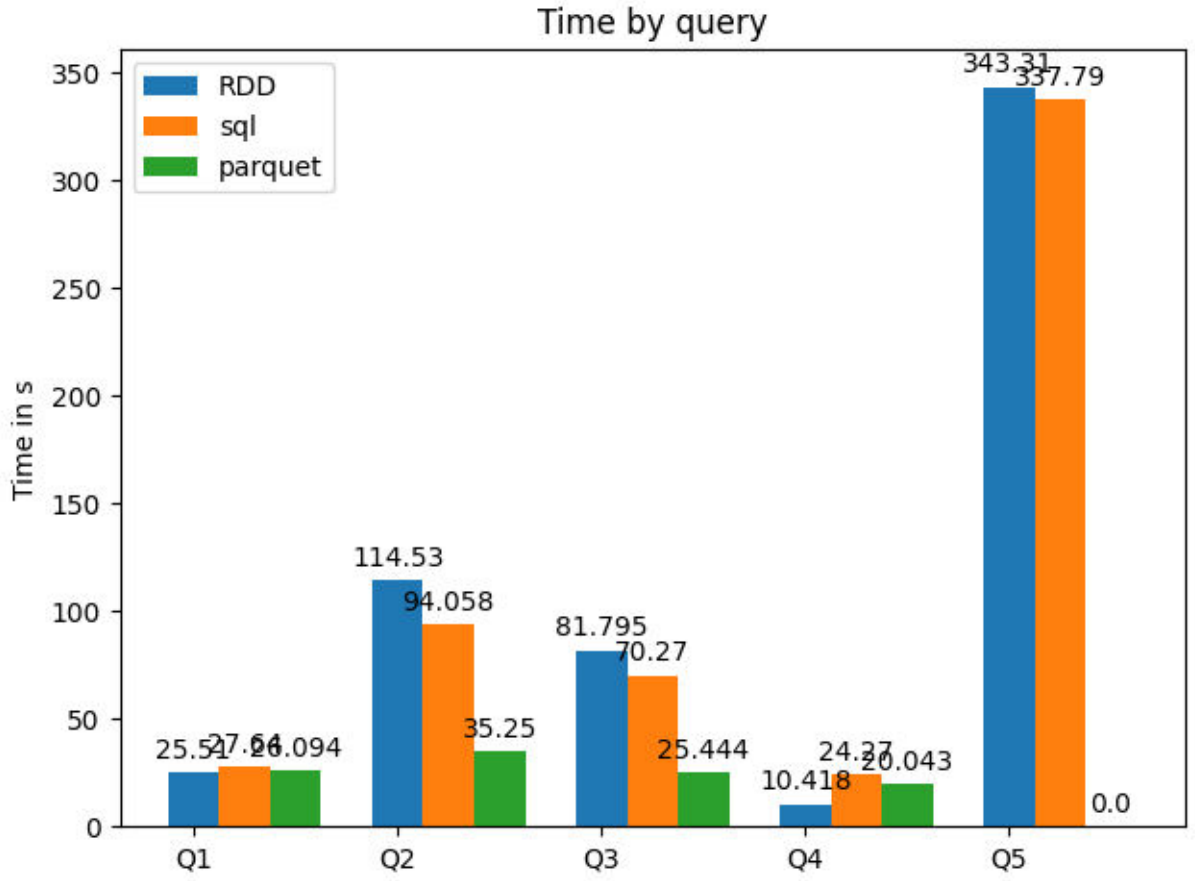
```
        emit(Category&User,(values_list,count));
    }
```

#6

```
    Map(key,values_list){ // (Rate,Title,Popularity,Reviews) element of list
        for i in values_list:
            key2 = key[0]; //Category as key
            value2 = (key[1],i[0],i[1],i[2],i[3]); (Rate,Title,Popularity,Reviews) as
value
            emit(key2, value2);
    }
```

#7

```
    ReduceByKey(Category, values_list){
        max = 0;
        for i in values_list{
            if i[3]>max → max = i[3];
        }
        emit(Category,values_list);
    }
```

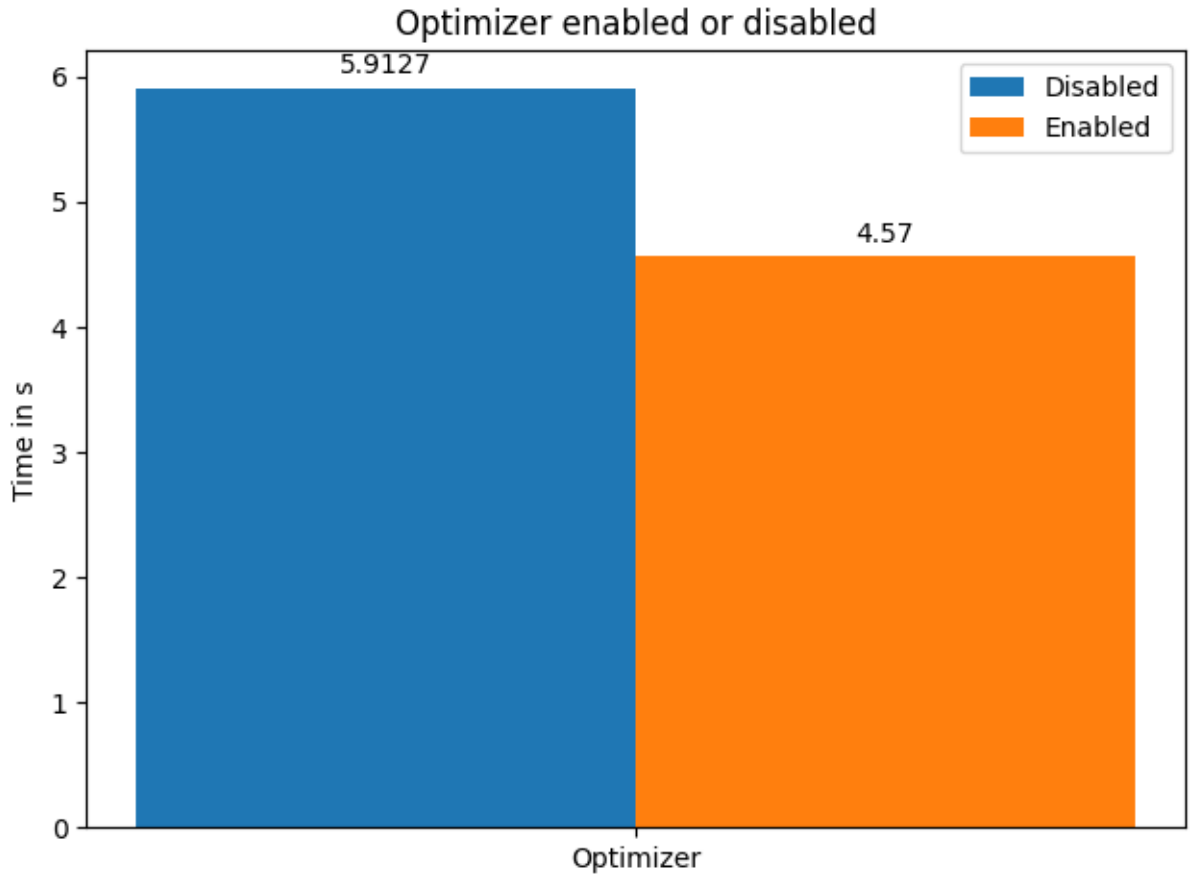


Στο ζητούμενο 4 του πρώτου μέρους ζητούνται τα διαγράμματα εκτέλεσης των τριών προγραμμάτων σε rdd, sql (csv και parquet). Παραπάνω φαίνεται το σχετικό διάγραμμα για κάθε query.

ΔΕΥΤΕΡΟ ΜΕΡΟΣ

Στο δεύτερο μέρος της εργασίας μας ζητούνται οι υλοποιήσεις των broadcast join και repartition join, καθώς και ένα παράδειγμα εκτέλεσης για κάθε ένα από αυτά (ζητούμενο 1,2,3 του 2ου μέρους). Ωστόσο, καθώς δεν έβρισκα τρόπο να κάνω αυτού του είδους join δίχως να χρησιμοποιήσω την μέθοδο join για την ένωση μεταξύ των 2 RDD δεν κατάφερνα να κάνω τις υλοποιήσεις που ζητούνταν (και τουλάχιστον στα μάτια μου θα είχαν ένα κάποιο νόημα όσον αφορά την πολυπλοκότητα.

Στο τέταρτο ζητούμενο μας ζητούνταν να συμπληρώσουμε κάποια κομμάτια κώδικα, τα οποία ήταν απαραίτητα προκειμένου να τρέξει το query ενεργοποιώντας ή απενεργοποιώντας τον βελτιστοποιητή. Οι χρόνοι εκτέλεσης χωρίς και με τον βελτιστοποιητή φαίνονται στο παρακάτω ραβδόγραμμα.



Ο βελτιστοποιητής λαμβάνει υπόψιν το μέγεθος των δεδομένων και αλλάζει την σειρά με την οποία κάνει τις πράξεις. Εάν ο ένας πίνακας είναι μικρός και ο άλλος μεγάλος τότε κάνει broadcast join αλλιώς κάνει repartition join.