Προχωρημένα Θέματα Βάσεων Δεδομένων
Εξαμηνιαία Εργασία

Βεκράκης Εμμανουήλ - 03116068
Μπέτζελος Χρήστος - 03116067

Μέρος 1ο

Στο μέρος αυτό κληθήκαμε να υλοποιήσουμε πέντε ερωτήματα στα RDD και SQL APIs του Spark SQL. Αρχικά παρουσιάζεται ο ψευδοκώδικας των ερωτημάτων αυτών σε MapReduce.

Q1:

```
map(_, record from movies.csv):
     if (get_year(date) >= 2000):
          gains = 100*(revenues - cost) / cost
          emit(year, (gains, values))

reduce(year, list of values):
     max_value = max(values, key=gains)
     emit(year, max_value)
```

Q2:

```
map1(_, record from ratings.csv):
     emit(user_id, rating)

reduce1(user_id, list of ratings):
     emit(user_id, avg(list of ratings))

map2(_, users):
     emit(0, (user_id, average))

reduce2(_, list of tuples):
     total = len(list of users)
     good = 0
     for t in tuples:
          if (average > 3):
               good += 1
     emit(0, 100*good/total)
```

Q3:

```
map1(_, record from movie_genres.csv joined with ratings.csv):
     emit(movie_id, (rating, genres))

reduce1(movie_id, list of tuples):
     sum = 0
     for t in tuples:
          sum += t[0]
     average_movie_rating = sum / len(tuples)
     emit(movie_id, (average_movie_rating, genres))

map2(movie_id, (average_movie_rating, genres)):
     for genre in genres:
          emit(genre, average_movie_rating)

reduce2(genre, list of average_movie_ratings):
     emit(genre, avg(average_movie_ratings))
```

Q4:

```
map(_, values from movies.csv joined with movie_genres.csv):
     # Lustrum meant five-year period in Ancient Rome
     if (genre == 'Drama'):
          emit(get_lustrum(year), word_length(summary))

reduce(lustrum, list of summary_lengths):
     emit(year, average(summary_lengths))
```

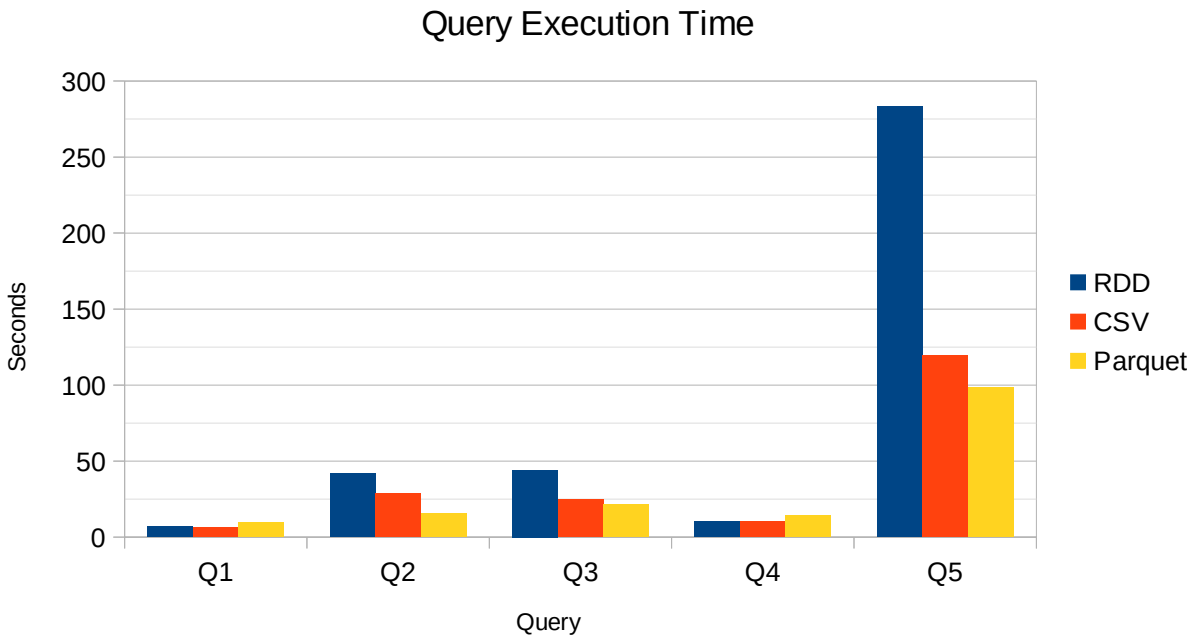
Q5:

```
map1(_, values from movie_genres.csv joined with ratings.csv):
     emit((genre, user_id), 1)

reduce1((genre, user_id), list of 1s (ratings count)):
     emit((genre, user_id), len(list of 1s))

map2((genre, user_id), ratingsCount):
     emit(genre, (user_id, ratingsCount))

reduce2(genre, (user_id, ratingsCount)):
     maxRatingsCount = max(ratingsCount)
     maxUser = user_id with maxRatingsCount
     emit((genre, maxUser), maxRatingCount)
     # We assume emissions get stored in table "table1"
```

```
# In order to limit below size, we can filter ratings only for the
# users that show up in table1
map3(_, values from movie_genres join movies join ratings):
     emit((genre, user_id), (movie_id, title, rating, popularity))

reduce3((genre, user_id), list(movie_id, title, rating, popularity)):
     mostFavMovie = movie_id with max(rating) and max(popularity)
     leastFavMovie = movie_id with min(rating) and max(popularity)

     maxRating = rating of mostFavMovie
     minRating = rating of leastFavMovie

     emit((genre, user_id), (mostFavMovie, maxRating, leastFavMovie,
                             minRating))
     # We assume emissions get stored in table "table2"

map4(_, values from table1 joined with table2):
     emit(0, (genre, user_id, mostFavMovie, maxRating, leastFavMovie,
               minRating))

reduce4(0, list of tuples):
     sort list by genre
     for t in list of tuples:
          output(t)
```

Ζητούμενο 4

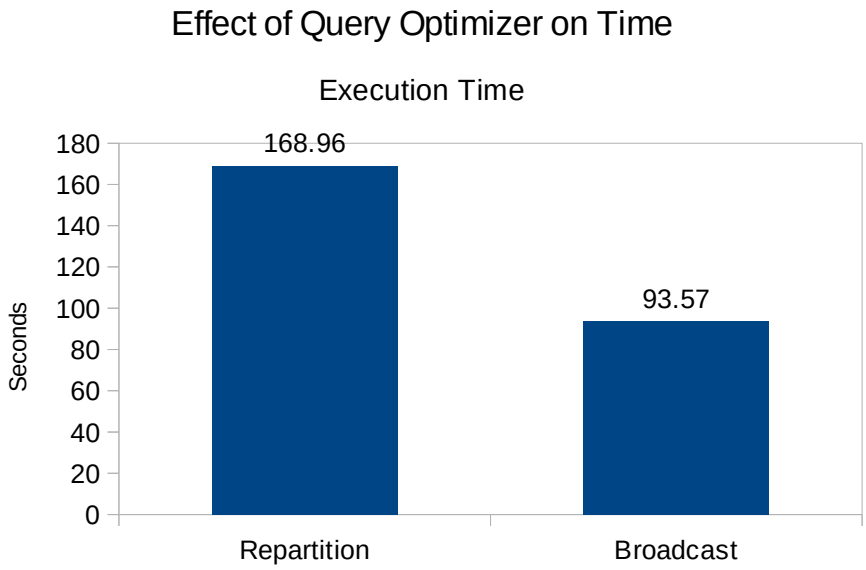
Εκτελέσαμε τα ερωτήματα που ζητούνται χρησιμοποιώντας το RDD API και το Spark SQL (για CSV και για Parquet είσοδο) και παρακάτω φαίνονται τα αποτελέσματα ως προς τον χρόνο εκτέλεσης της κάθε υλοποίησης. Παρατηρούμε ότι στα περισσότερα ερωτήματα, ο χρόνος εκτέλεσης του SQL με το Parquet είναι μικρότερος από ο,τι αυτός με την είσοδο σε CSV. Ο λόγος που δεν συμβαίνει αυτό στην περίπτωση του Q1 και του Q4 είναι διότι τα ερωτήματα αυτά δεν εκμεταλλεύονται την επιπλέον πληροφορία που παρέχεται από τα Parquet αρχεία. Ο χρόνος των RDDs είναι μεγαλύτερος από τα SQL ερωτήματα με εξαίρεση πάλι τα Q1 και Q4, ενώ όπως είναι αναμενόμενο, το Q5 απαιτεί πολύ μεγαλύτερο χρόνο σε όλες τις εκτελέσεις, από τα υπόλοιπα queries. Τα αποτελέσματα προέκυψαν πανομοιότυπα για όλες τις εκτελέσεις και βρίσκονται στον φάκελο output του παραδοτέου. Το infer schema δεν χρησιμοποιείται γιατί, όπως παρατηρήθηκε, δεν είναι απαραίτητο για την εξαγωγή συμπερασμάτων σχετικά με τον τύπο των δεδομένων.



Μέρος 2ο

Ζητούμενο 3

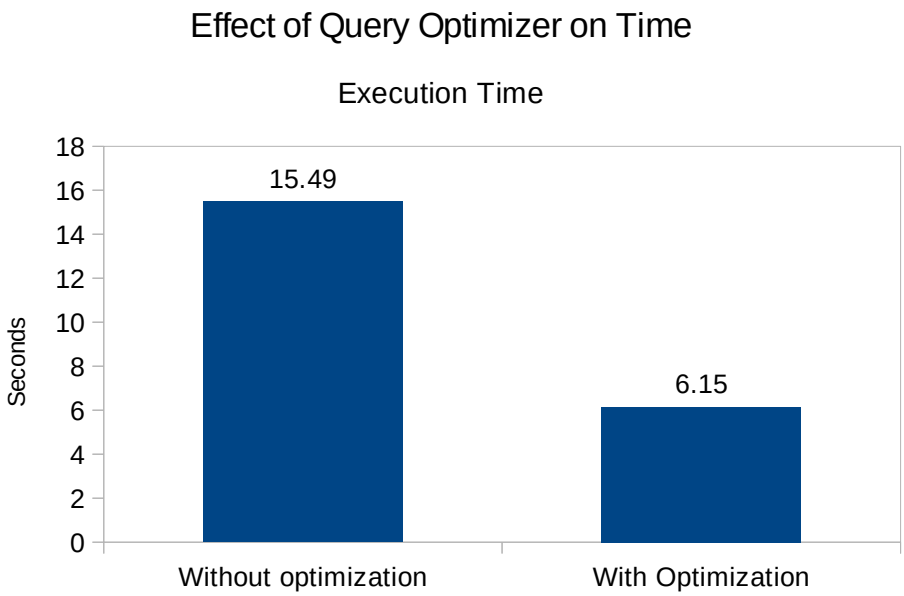
Απομονώσαμε 100 γραμμές του πίνακα movie genres και τις ενώσαμε με τον πίνακα ratings χρησιμοποιώντας τα δύο είδη ενώσεων που υλοποιήσαμε στα προηγούμενα ζητούμενα (Broadcast και Repartition Join). Στο παρακάτω διάγραμμα απεικονίζονται οι χρόνοι εκτέλεσης της καθεμίας υλοποίησης. Τα αποτελέσματα των joins προέκυψαν ίδια και για τις δύο περιπτώσεις.



Παρατηρούμε ότι το repartition join χρειάζεται πολύ περισσότερο χρόνο να εκτελεστεί, όπως ήταν αναμενόμενο. Σύμφωνα με τη θεωρία, το broadcast join υπερέχει σε περίπτωση που ο ένας πίνακας είναι πολύ μικρός και χωράει στην τοπική μνήμη κάθε κόμβου, αφού μειώνονται δραματικά οι μεταφορές δεδομένων μέσω του δικτύου.

Ζητούμενο 4

Στο ζητούμενο αυτό μελετήσαμε την επίδραση του βελτιστοποιητή ερωτημάτων του SparkSQL. Μέσω της επιλογής spark.sql.autoBroadcastJoinThreshold, ορίζουμε το μέγιστο μέγεθος που μπορεί να έχει ένας πίνακας για να γίνει broadcast join. Ορίζοντας την τιμή αυτή -1, ουσιαστικά απαγορεύουμε από τον optimizer να χρησιμοποιήσει broadcast join. Παρακάτω φαίνονται το ραβδοδιάγραμμα με τους χρόνους εκτέλεσης του δοθέντος query χωρίς και με τον βελτιστοποιητή ενεργοποιημένο, καθώς και τα πλάνα εκτέλεσης των δύο αυτών περιπτώσεων. Στα πλάνα εκτέλεσης φαίνεται ότι η ύπαρξη βελτιστοποιητή οδηγεί σε χρήση broadcast join.



Πλάνο εκτέλεσης χωρίς βελτιστοποίηση:

```
== Physical Plan ==
*(6) SortMergeJoin [_c0#8], [_c1#1], Inner
:- *(3) Sort [_c0#8 ASC NULLS FIRST], false, 0
:  +- Exchange hashpartitioning(_c0#8, 200)
:     +- *(2) Filter isnotnull(_c0#8)
:        +- *(2) GlobalLimit 100
:           +- Exchange SinglePartition
:              +- *(1) LocalLimit 100
:                 +- *(1) FileScan parquet [_c0#8,_c1#9] Batched: true, Format: Parquet, Location: InMemoryFileIndex[hdfs://master:9000/project/movie
_genres.parquet], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<_c0:string,_c1:string>
+- *(5) Sort [_c1#1 ASC NULLS FIRST], false, 0
   +- Exchange hashpartitioning(_c1#1, 200)
      +- *(4) Project [_c0#0, _c1#1, _c2#2, _c3#3]
         +- *(4) Filter isnotnull(_c1#1)
            +- *(4) FileScan parquet [_c0#0,_c1#1,_c2#2,_c3#3] Batched: true, Format: Parquet, Location: InMemoryFileIndex[hdfs://master:9000/project
/ratings.parquet], PartitionFilters: [], PushedFilters: [IsNotNull(_c1)], ReadSchema: struct<_c0:string,_c1:string,_c2:string,_c3:string>
Time with choosing join type disabled is 15.4929 sec.
```

Πλάνο εκτέλεσης με βελτιστοποίηση:

```
== Physical Plan ==
*(3) BroadcastHashJoin [_c0#8], [_c1#1], Inner, BuildLeft
:- BroadcastExchange HashedRelationBroadcastMode(List(input[0, string, false]))
:  +- *(2) Filter isnotnull(_c0#8)
:     +- *(2) GlobalLimit 100
:        +- Exchange SinglePartition
:           +- *(1) LocalLimit 100
:              +- *(1) FileScan parquet [_c0#8,_c1#9] Batched: true, Format: Parquet, Location: InMemoryFileIndex[hdfs://master:9000/project/movie_ge
nres.parquet], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<_c0:string,_c1:string>
+- *(3) Project [_c0#0, _c1#1, _c2#2, _c3#3]
   +- *(3) Filter isnotnull(_c1#1)
      +- *(3) FileScan parquet [_c0#0,_c1#1,_c2#2,_c3#3] Batched: true, Format: Parquet, Location: InMemoryFileIndex[hdfs://master:9000/project/ratin
gs.parquet], PartitionFilters: [], PushedFilters: [IsNotNull(_c1)], ReadSchema: struct<_c0:string,_c1:string,_c2:string,_c3:string>
Time with choosing join type enabled is 6.1506 sec.
```