ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΕΞΑΜΗΝΙΑΙΑ ΕΡΓΑΣΙΑ

ΡΟΗ Λ – ΕΞΑΜΗΝΟ 9ο
ΑΚ. ΕΤΟΣ 2020 - 2021
ΠΡΟΧΩΡΗΜΕΝΑ ΘΕΜΑΤΑ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

Χρήση του Apache Spark στις Βάσεις Δεδομένων

Αραπίδης Μάνος, ΑΜ: 03116071
Σταυρόπουλος Βασίλης, ΑΜ: 03116039

Μέρος 1ο

Ζητούμενο 1ο

Εκτελούμε τις παρακάτω εντολές από το απομακρυσμένο μηχάνημα από τον χρήστη master

```
wget http://www.cslab.ntua.gr/courses/atds/movie_data.tar.gz

tar -xvf movie_data.tar.gz

hadoop fs -put movie_data hdfs://master:9000/movie_data
```

Η πρώτη κατεβάζει το συμπιεσμένο αρχείο.
Η δεύτερη το αποσυμπιέζει και η τρίτη το φορτώνει στο hadoop.

Ζητούμενο 2ο

Εκτελούμε το παρακάτω πρόγραμμα για να μετατρέψουμε τα αρχεία csv σε parquet μορφή και να τα φορτώσουμε στο hadoop.

```
#!/bin/python3

from pyspark.sql import SparkSession
spark =
SparkSession.builder.master("local").appName("parquet_example").getOrCr
eate()
df = spark.read.csv('hdfs://master:9000/movie_data/movies.csv', header
= False)
df.write.parquet('hdfs://master:9000/movie_data/movies.parquet')



df = spark.read.csv('hdfs://master:9000/movie_data/movie_genres.csv',
header = False)
df.write.parquet('hdfs://master:9000/movie_data/movie_genres.parquet')

df = spark.read.csv('hdfs://master:9000/movie_data/ratings.csv', header
= False)
df.write.parquet('hdfs://master:9000/movie_data/ratings.parquet')
```

Ζητούμενο 3ο

Οι ψευδοκώδικες για κάθε ένα από τα πέντε ζητούμενα queries είναι:

Q1

```
map (movie):
     if year!=”” or cost!=0 or revenue!=0:
         profit = (revenue - cost)*100/cost
         emit(year,(title,profit)

reduce (key,records):
      max = 0
      for each record in records:
            if (max<profit)
                  max = profit
      emit(max)

map (key,movie):
      emit(key,title)
```

Q2

```
map reduce for some

map(movie):
      emit(movie_id,(rating,1))

reduce(key,movies):
      athroisma_rating = 0
      sinolo_kritikwn = 0
      for each movie in movies:
            athroisma_rating = rating
            sinolo_kritikwn = 1
      emit(athroisma_rating,sinolo_kritikwn)

map(key,record):
      if(rating>3.0):
          total = athroisma_rating/sinolo_kritikwn
          emit(key,total)

map(movie):
      emit(1,1)

reduce(key,records):
      sum = 0
      for record in records:
            sum+=1
      emit(sum)

---------------------
map reduce for total

map(movie):
      emit(movie_id,1)

reduce(key,movies):
      emit(0)

map(movie):
      emit(1,1)

reduce(key,records):
      sum = 0
      for record in records:
            sum+=1
emit(sum)
```

Q3

```
map(movie):
    emit(movie_id,rating)

map(movie):
    emit(movie_id,genre)

join-genres-rating = join(genres_t,rating_t)
-----------
map-reduce on join-genres-rating ( result saved as mean_rating)

map(key,movie):
      emit(genre,rating,1)

reduce(key,movies):
      athroisma_rating = 0
      sinolo_tainiwn = 1
      for movie in movies:
            athroisma_rating += rating
            sinolo_tainiwn += 1
      emit(athroisma_rating,sinolo_tainiwn)

map(key,value):
      result = sum0/sum1
      emit(key,result)

--------------
map-reduce on join-genres-rating ( result saved as
distinct_genres_movies_count)

map(movie):
      emit(genre,movie_id)

distinct()

map(key,movie):
      emit(key,1)

reduce(key,movies):
      sinolo_tainiwn = 0
      for movie in movies:
            sinolo_tainiwn += 1
      emit(sinolo_tainiwn)
```

```
----------

join(mean_rating,distinct_genres_movies_count)

map(key,values):
      emit(key,mean_rating,total_count)
```

Q4

```
map on movies

map(movie):
      quinq = quinq(year)
      summary_length = len(summary)
      emit(movie_id,(quinq,summary_length),1))

---------
map on genres

map(record):
      if(genre==Drama):
          emit(movie_id,genre)

----
joined = join(movies_t,genres_t)

----
map reduce on joined

map(key,movie):
      emit(quinq,(summary_length,1)

reduce(key,movies):
      sinonliko_mikos_summaries = 0
      sinolo_tainiwn = 0
      for movie in movies:
            sinonliko_mikos_summaries = summary_length
            sinolo_tainiwn = 1
      emit(sinonliko_mikos_summaries,sinolo_tainiwn)

map(key,values):
      result = sinonliko_mikos_summaries/sinolo_tainiwn
      emit(key,result)
```

Q5

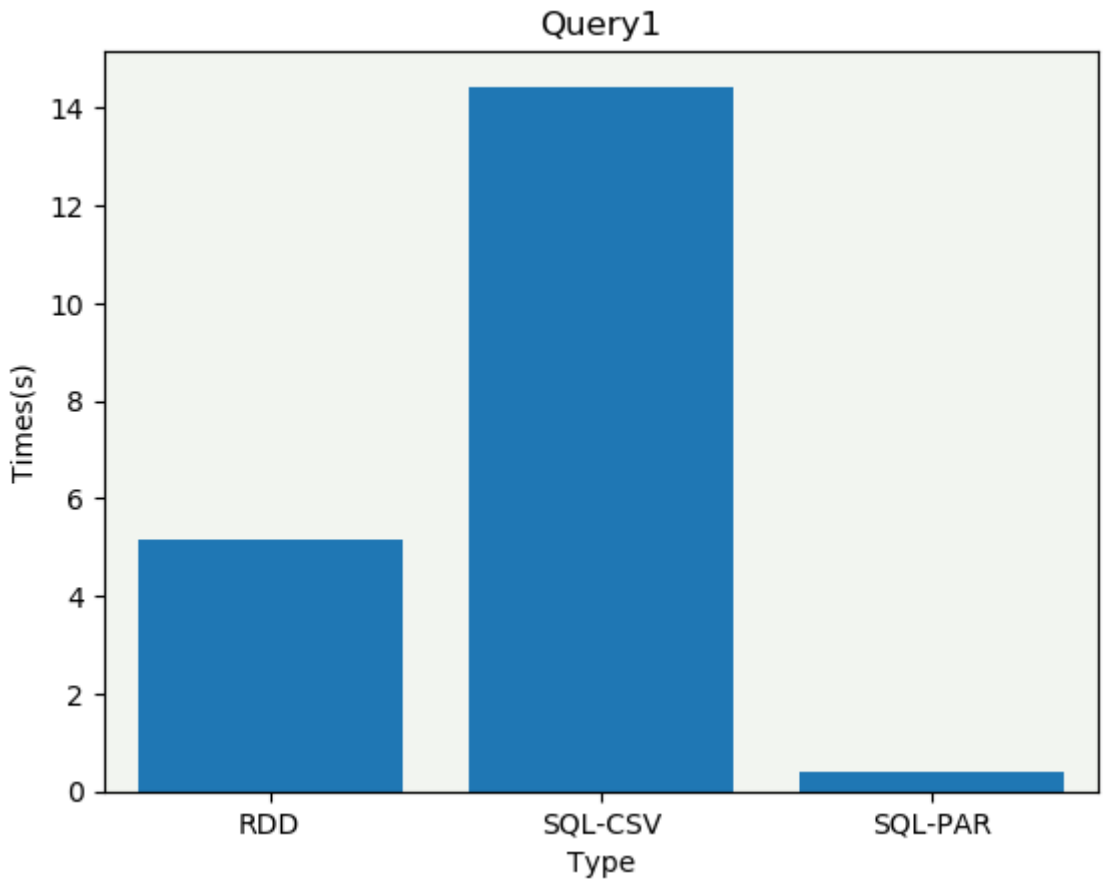
```
map(x):     #rating
    emit(movie_id, user_id, rating)

map(x):     #movie_genre
    emit((movie_id, genre))

map(x):     #movie_popularity_title
    emit(movie_id,popularity,title)

join(rating,movie_genre)        #rating_genre

map(x):
    emit(movie_id,(genre, user_id,rating))

join(rating_genre,movie_popularity_title)

map(x):     #rat_gen_pop
    emit((genre,user_i),(movie_id,rating,popularity,title))

map(x):     #count_users
    emit((genre,user_id),1)

reduce(key,values):
    count=0
    for value in values:
        count+=value
    emit(key,count)

map(x):
    emit(genre,(user_id,count))

reduce(key,values):
    max_count=0
    for value in values:
        max_count=max(max_count,count)
    emit max_count

distict()

map(x):     #count_users
    emit((genre,user_id),count)

join(rat_gen_pop, count_users)
```

```
map(x):     #join_ratings_genres_movies_count
    emit(genre,(user_id,movie_id,rating,popularity,title,count))


reduce(key,values):
    max_rating=values[0]
    for value in values:
        max_rating=max((max_rating, value), key=rating)
    emit max_rating

reduce(key, values):        #max_rating
    max_popularity=values[0]
    for value in values:
        max_popularity=max((max_popularity, value), key=popularity)
    emit max_popularity

---------

reduce(key,values):
    min_rating=values[0]
    for value in values:
        min_rating=max((min_rating, value), key=rating)
    emit min_rating

reduce(key, values):        #min_rating
    min_popularity=values[0]
    for value in values:
        min_popularity=max((min_popularity, value), key=popularity)
    emit min_popularity


join(min_rating,max_rating):
    emit(genre,user_id,max_title,max_rating,min_title,min_rating,count)
```

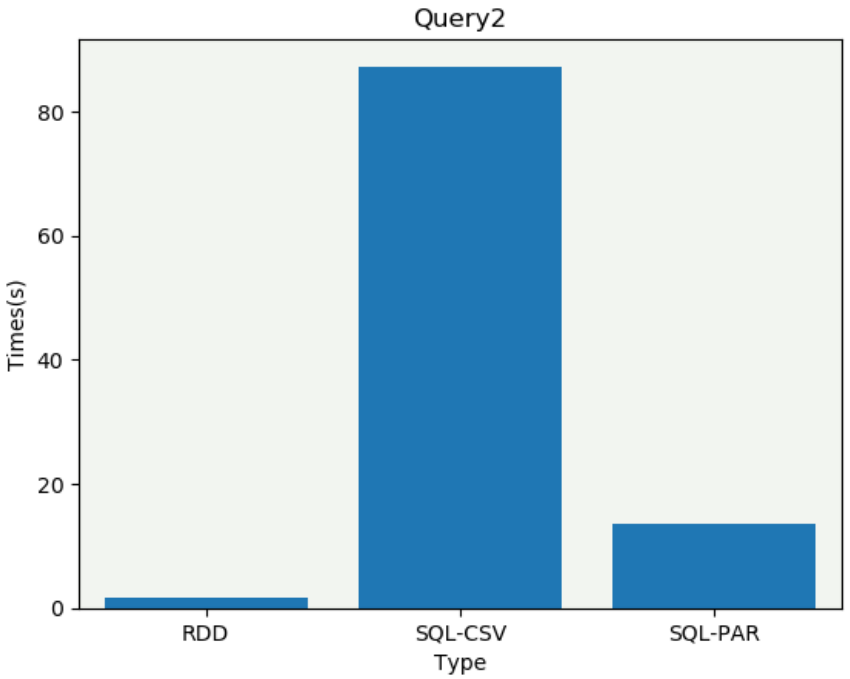
Ζητούμενο 4ο

Τα διαγράμματα με τους χρόνους εκτέλεσης για κάθε query είναι:

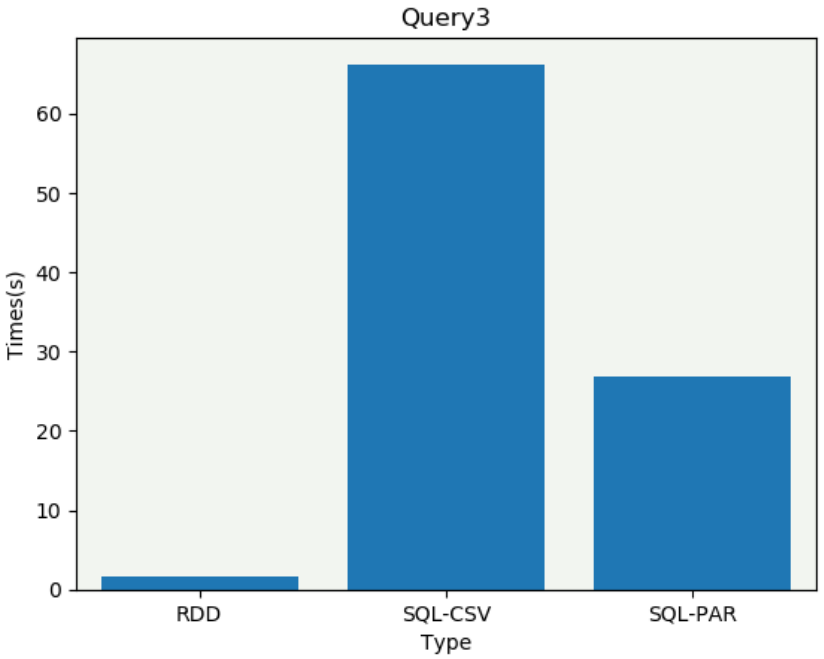
Q1



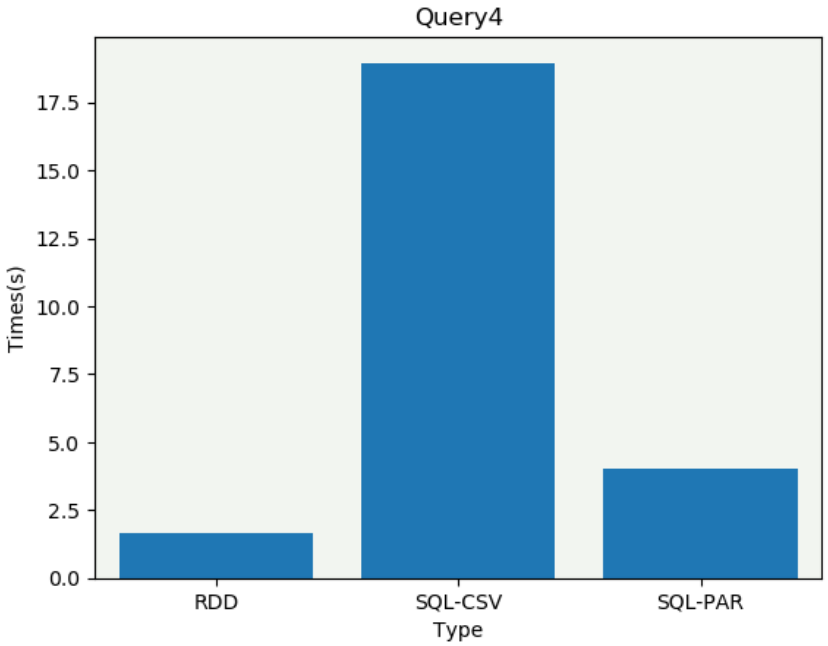
Q2


Query2
Times(s)
0
20
40
60
80
RDD
SQL-CSV
SQL-PAR
Type

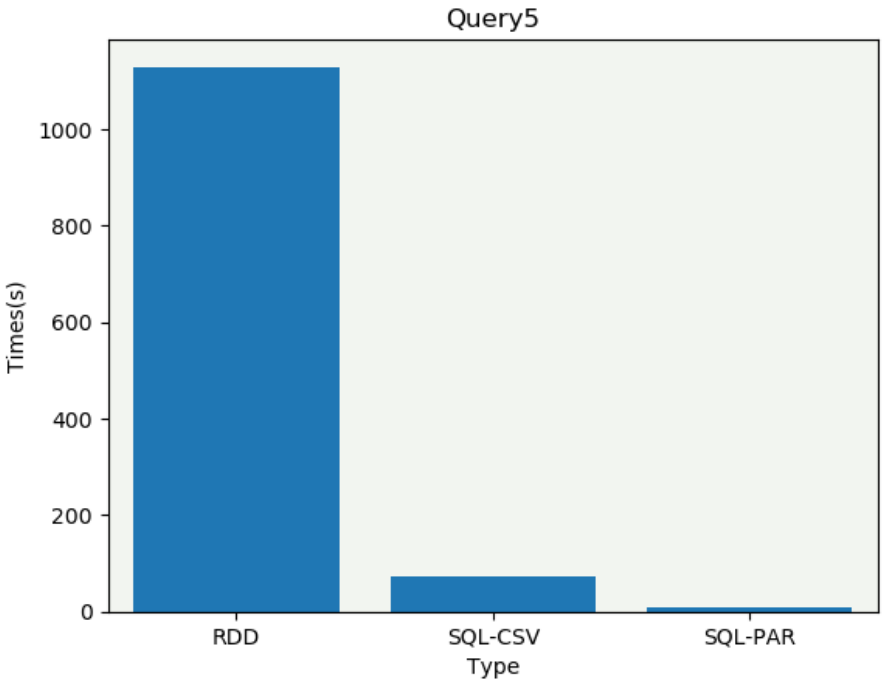
Q3


Query3
Times(s)
0
10
20
30
40
50
60
RDD
SQL-CSV
SQL-PAR
Type

Q4


Query4
Times(s)
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
RDD
SQL-CSV
SQL-PAR
Type

Q5



Παρατηρούμε ότι και στις πέντε περιπτώσεις η υλοποίηση του query με sql με είσοδο parquet αρχείο απαιτεί αρκετά λιγότερο χρόνο από ότι η ίδια υλοποίηση sql με είσοδο αρχείο csv. Αυτό ήταν αναμενόμενο, αφού τα αρχεία parquet έχουν αρκετά μικρότερο memory footprint (άρα μειώνεται δραστικά ο απαιτούμενος χρόνος για την φόρτωση) και διατηρούν στατιστικά πάνω στα δεδομένα τα οποία οδηγούν σε ταχύτερη επεξεργασίας τους από το κατανεμημένο σύστημα. Δεν χρησιμοποιούμε infer schema για τα parquet αρχεία καθώς αναγνωρίζει τους τύπους των στηλών. Επιπλέον, μπορούμε να προσθέσουμε δικό μας schema εξ αρχής στην μετατροπή από csv σε parquet το οποίο διατηρείται.

Μέρος 2ο

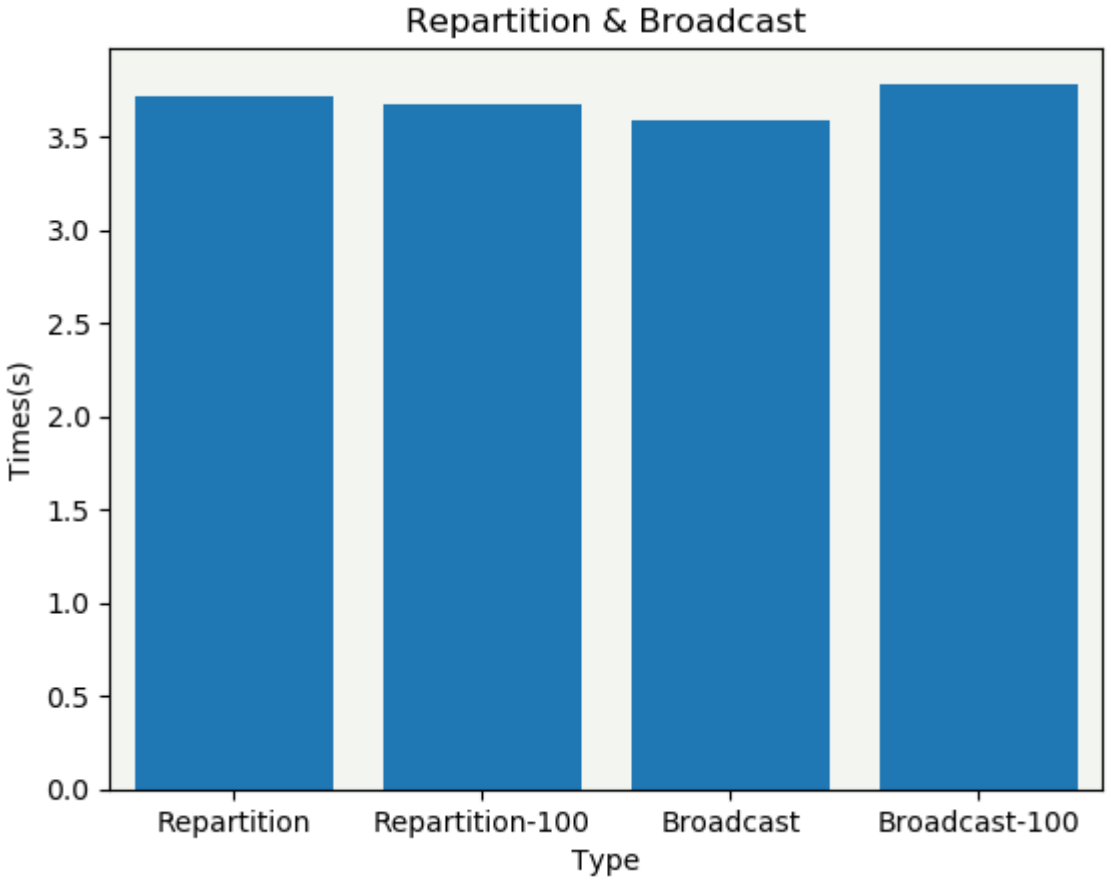
Ζητούμενο 1ο

Ο κώδικας υλοποίησης του repartition join βρίσκεται στο αρχείο repartition.py.

Ζητούμενο 2ο

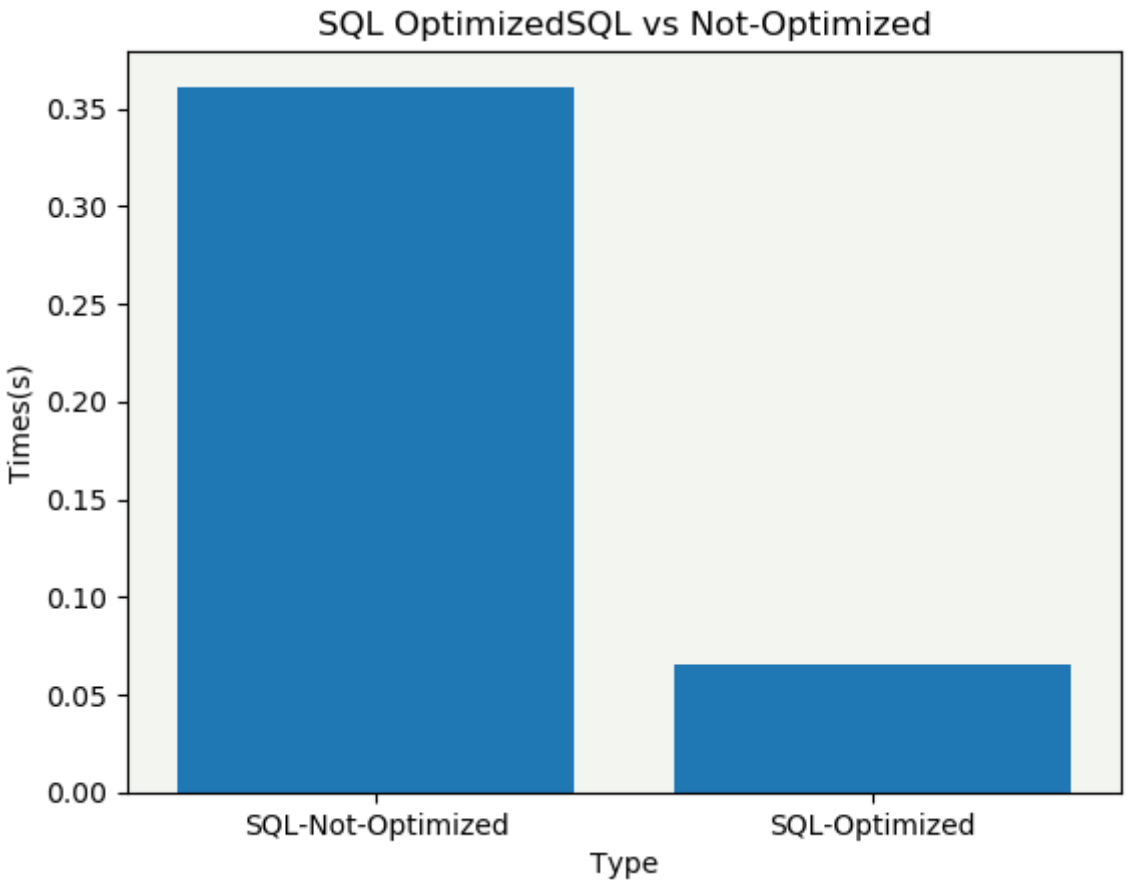
Ο κώδικας υλοποίησης του broadcast join βρίσκεται στο αρχείο broadcast.py.

Ζητούμενο 3ο



Παρατηρούμε ότι πετυχαίνουμε καλύτερο χρόνο με το broadcast join στην γενική περίπτωση. Ωστόσο, τα αποτελέσματα είναι αρκετά κοντινά για όλες τις περιπτώσεις.

Ζητούμενο 4ο



Στην περίπτωση που κάνουμε enable την βελτιστοποίηση παρατηρούμε ότι πραγματοποιείται broadcast join, ενώ στην αντίθετη περίπτωση γίνεται repartition join. Αυτό μπορούμε να το εντοπίσουμε από τα logs που προκύπτουν και συγκεκριμένα κοιτάζοντας το physical plan.