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

ΧΡΗΣΗ ΤΟΥ APACHE SPARK ΣΤΙΣ ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ

Λώλης Ηλίας 03116661

Ευαγγελία Ρεντίφη – 03116164

Public IP: 83.212.79.7

Μέρος 1ο

Ζητούμενο 2: Παράδειγμα μετατροπής αρχείου σε Parquet έχουμε στο αρχείο “code/csv to parquet.py”

Ζητούμενο 3: Τα screenshots των απαντήσεων των MapReduce βρίσκεται στον φάκελο “output/ScreenshotsMapReduce”.

Τα screenshots των απαντήσεων των SQL για CSV αρχεία βρίσκεται στον φάκελο “output/ScreenshotsSQL- csv”.

Τα screenshots των απαντήσεων των SQL για parquet αρχεία βρίσκεται στον φάκελο “output/ScreenshotsSQL- parquet”.

Οι κώδικες βρίσκονται στον φάκελο code.

Ζητούμενο 4:

Στο ζητούμενο αυτό παραθέτουμε τους χρόνους που προέκυψαν από τις υλοποιήσεις των queries, του ζητουμένου 3 της εργασίας, με RDD API ,Spark SQL-csv και Spark SQL-parquet

	RDD API	**Spark SQL-csv**	**Spark SQL-parquet**
Q1	0.2837886810302734	7.485328435897827	8.86049222946167
Q2	81.59077215194702	34.27328062057495	17.733304977416992
Q3	48,92343424342341	26.7877995967865	22.592684507369995
Q4	14,14324235158749	12.509453773498535	11.395089864730835
Q5	376.8594615459442	113.65680289268494	228.7125129699707

Με βάση τις πληροφορίες που παρατίθενται στο ζητούμενο 2 (όπου ζητούνταν η μετατροπή των αρχείων από csv μορφή σε parquet) αναμέναμε ότι η εκτέλεση των

queries θα ήταν πιο αποδοτική επι των αρχείων parquet.Σύμφωνα με τις μετρήσεις που έγιναν έχουμε:

Q1: Παρατηρούμε ότι το RDD API υπερτερεί έναντι των άλλων 2 υλοποιήσεων που έχουν παρόμοιους χρόνους (διαφορά κοντά στο 1 sec).

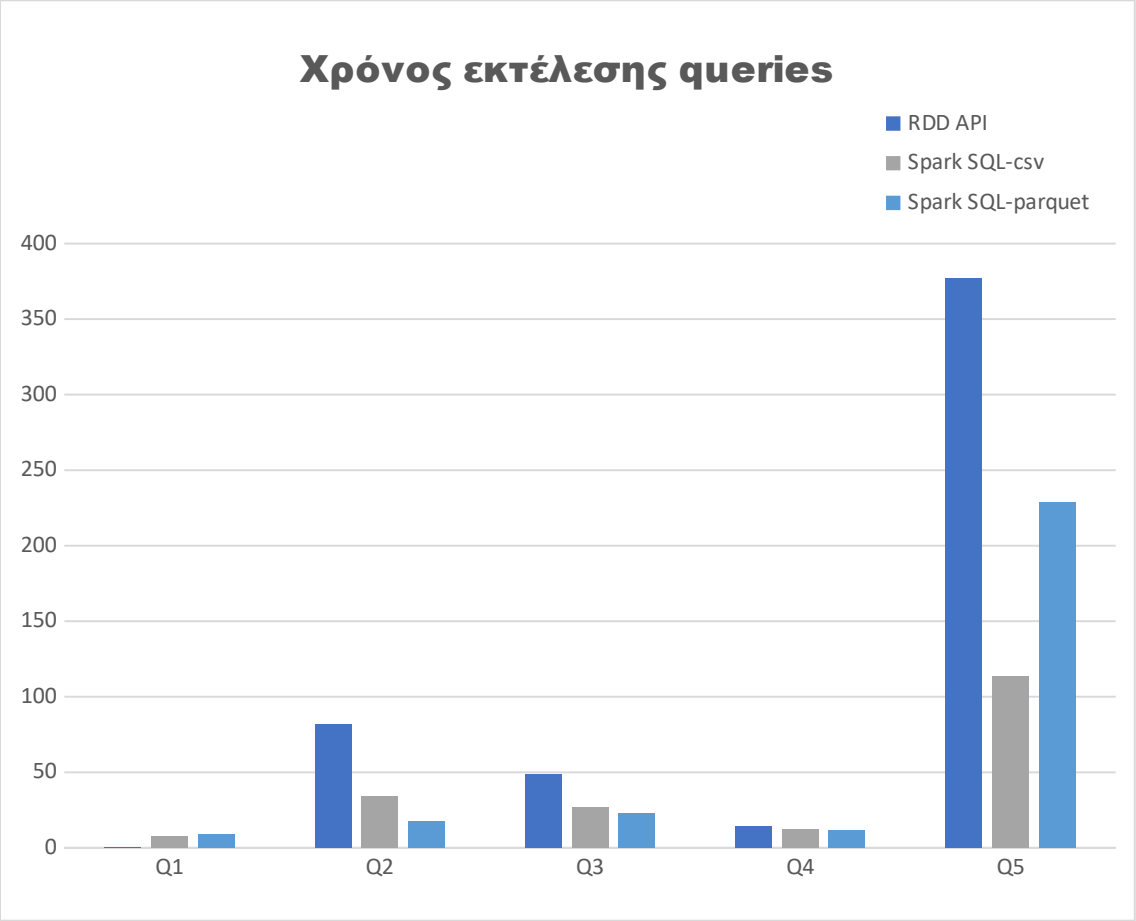
Q2: Η υλοποίηση σε Spark SQL με χρήση parquet αρχείων υπερτερεί έναντι των άλλων 2 και μάλιστα η υλοποίηση με RDD API φαίνεται να είναι σημαντικά πιο χρονοβόρα.

Q3: Και εδώ η υλοποίηση με Spark-SQL parquet φαίνεται να υπερτερεί των άλλων δυο υλοποιήσεων. Ωστόσο η διαφορά με την Spark-SQL parquet δεν είναι τρομακτικά μεγάλη (της τάξης των 4 sec περίπου)

Q4: Εδώ η υλοποίηση με Spark-SQL parquet υπερτερεί των άλλων δυο υλοποιήσεων

Q5: Σε αυτό το query η ιδανικότερη υλοποίηση είναι με Spark SQL-csv ,ενώ και πάλι κατά την εκτέλεση της RDD API εκδοχής είχαμε slave drop οπότε ο χρόνος δεν είναι ακριβής.

Συμπερασματικά θα λέγαμε ότι με Spark SQL(και ειδικότερα με input parquet αρχεία) έχουμε καλύτερους χρόνους εκτέλεσης στις περιπτώσεις ομαδοποίησης και συνένωσης(group by ,join κλπ).Ωστόσο στην περίπτωση που απαιτείται η δημιουργία μιας νέας στήλης έπειτα από πράξεις πάνω στα δεδομένα (στήλη Profit στην περίπτωση του Q1) το RDD API είναι ταχύτερο(ανάλογα συμπεράσματα επισημαίνονται και σε σχετικές δημοσιεύσεις)



Ψευδοκώδικες MapReduce

Q1:

```
# Gia kathe entry sto table movies kaloume tin map
def get_kerdos(movie):
   return (movie['esoda']  - movie['kostos_paragogis']   ) * 100 /
movie['kostos_paragogis']

map(key, movie):
   if(movie['kostos_paragogis'] >0 and value['esoda']  > 0 ):
      year_in_string = split( movie['date'] , '-') [0]
      movie['year'] = int(year_in_string)
      movie['kerdos'] = get_kerdos(movie)
      if(movie['year'] > 2000):
         emit( movie['year'], movie )

# ara edw exoume san key to year kai movies_array oles tis tainies me xrono = year
reduce(year, movies_array):
   highest_movie = movies_array[0]
   foreach movie in movies_array:
      if(movie['kerdos'] > highest_movie['kerdos']):
         highest_movie = movie
   emit(highest_movie)
```

Q2:

```
# trexoume tin map gia kathe entry tou rating.csv
map(key, value):
   emit(value['user_id'], [ value['rating'], 1  ]])

# edw apla briskoume to median rating tou kathe xristi kai filtraroume na einai > 3.0
reduce(user_id, array):
   total_ratings_sum = 0
   total_ratings = 0
   foreach element in array:
      total_ratings_sum + element['rating']
      total_ratings++
   average = total_ratings_sum/total_ratings
   if(average > 3):
      emit(user_id, average)
# theoroume pws to apotelesma tou parapanw reduce apothikeuete se mia lista me onoma
user_with_median_bigger_than_3
# dld i metabliti user_with_median_bigger_than_3 e;inai mia lista tis morfis ( (user_id,average),...,
(user_id,average) )
# O arithmos twn users p exoun median rating > 3 einai to lenfth tis parapanw listas
user_with_median_bigger_than_3 = length(user_with_median_bigger_than_3)

# twra arkei na brw ton sunoliko arithmo twn xristwn
# i parakatw map kaleite gia oles tis touples tou arxeiou ratings.csv
map(null,rating)
   emit(rating['userId'])

# kaloume tin distinct kai meta count wste na broume ta distinct user id
# dld ton arithmo olwn twn xristwn
number_of_users = ratings.distinct().count()
```

result = user_with_median_bigger_than_3/ number_of_users

Q3:

```
# gia kathe toupla tou ratings kaloume tin parakatw map
map(movieId, value : a value from ratings):
   emit(movieId, value)
# Briskw gia kathe tainia ton Meso Oro tis
reduce(movieId, valuesArray):
   sum = 0
   numberOfReviews = 0
   foreach value in valuesArray:
      sum += value['rating']
      numberOfRevies++
   movie_average_rating = sum / numberOfReviews
   emit(movieId,  value + {'average_rating'  : movie_average_rating}  )

# ta parapanw emits apothikeuontai stin metavliti movies_with_average_rating
# ara exoume ena sinolika touples tis morfis movie_id, movieValues opoy ta movieValues periexoun
# movieName,...,popularity kai to average_rating

# movieGenres einai oi touples tou movie_genres.csv
movies_with_average_with_genre = movies_with_average_rating.join(movieGeres)

# Kaloume tin parakatw map gia kathe toupla tou
map(null, touple):
   emit(touple['genre'], touple)

reduce(genre, touples):
   sum = 0
   numberOfMovies = 0
   foreach touple in touples:
      sum = touple['average_rating']
      numberOfMovies++
   genre_average_rating = sum / numberOfMovies
   emit(genre, genre_average_rating,numberOfMovies)
```

Q4:

```
get_year(date):
   return int(date.spit("-")[0])

dates = [[2000,2004],[2005,2009], [2010,2014], [2015,2019] ]
# dates = [  [2010,2014], [2015,2019] ]
for i in dates:
   date_start = i[0]
   date_end = i[1]

   # estw variable genres oles oi touples tou arxeiou movie_genres.csv
   # kaloume tin parakatw map gia kathe toupla tou genres
   map(null, genre):
      if(genre=='Drama')
         emit(genre['id'],genre)

   # ara me tin parapanw map filtrarame kai exoume mono touples tou genre
   # poy einai Drama.
```

```
    # estw variable movies oles oi touples tou arxeiou movies.csv
    # kaloume tin parakatw map gia kathe toupla tou arxeiou movies.csv
    # kanoume fitler i tainia na exei mi keno description kai to date tis na einai anamesa
    # apo date_start kai date_end, dld tis xronies p eksetazoume
    map(null, movie):
        if(movie['description'] != ""  and get_year(movie['date']) >= date_start and
get_year(movie['date']) <= (date_end))
            emit(movie['id'], movie)

    # ara kratame sto variable movies oles tis touples p den exoun keno description kai to date tous einai
    # anamesa sto date_start kai date_end

    only_drama_movies_within_date = genres.join(movies)

    # ara sto variable only_drama_movies_within_date exoume to join twn genres kai movies
    # gia kathe toupla tou only_drama_movies_within_date kaloume tin parakatw map

    map(null, value):
        emit(value['genre'] , length(  split( value['description'],',') ))

    reduce(genre, lengthsOfMovieDescriptions):
        totalLengthSum = 0;
        numberOfLengths = 0
        foreach length in lengthsOfMovieDescriptions:
            totalLengthSum += length
            numberOfLengths++
        average = totalLengthSum/numberOfLengths
        emit(genre,average)
```

Q5:

```
# estw movies oi touples tou arxeiou movies.csv
# estw movie_genres oi touples tou arxeiou movie_genres.csv
# estw ratings oi touples tou arxeiou ratings.csv

# stin metavliti movie_genres apothikeuoume to join twn movies kai genres
# mia toupla sto movies mporei na antistoixei se N touples sto movie_genres,
# afou mporei na exei N genres
movie_genres = movies.join(genres)

# sto movie_genres to kanoume join me to ratings, wste gia kathe toupla tou movie_genres
# na exoume M touples sto genres_with_best_reviewer_with_number_of_reviews an i tainia auti
# exei M reviews
ratings_movie_genres = ratings.join(movie_genres)

# epomeno vima einai gia kathe user na brw gia kathe genre ton arithmo ton reviews p exei kanei
# se auto to sigkekrimeno genre

# gia kathe toupla tis ratings_movie_genres kalw tin parakatw map
map(null, value):
    emit( (value['userId'] , value['genre']) , 1 )

reduce( (userId, genre ) , listApoAssoi ):
    sum = 0
    foreach asso in listApoAssoi:
```

```
        sum = sum + asso;
    emit( genre, [userId,numberOfReviews] )

# ta emits tou parapanw reduce apothikeuontai stin metavliti
genres_with_best_reviewer_with_number_of_reviews
# i metavliti genres_with_best_reviewer_with_number_of_reviews, exei gia kathe user kai gia kathe
genre
# ton arithmo twn reviews p exei kanei.
# dld exei px (('71', 'Crime'), 2)  kai (('176', 'Mystery'), 4)
# sunepws kanoume reduce me key to genre kai krataw to userId p exei ton megalitero arithmo reviews

reduce (genre, arrayValues):
    userWithMostReviews = arrayValues[0]
    foreach value in arrayValues:
        if(value['numberOfReviews'] > userWithMostReviews['numberOfReviews']):
            userWithMostReviews = value
    emit(genre, value)

# sunepws i metavliti genres_with_best_reviewer_with_number_of_reviews
# periexei toses touples oses kai o arithmos twn genre
# kai touples tis morfis (genre, [userId, numberOfRatings]).
# dld gia kathe genre, exoume brei ton user p exei kanei ta perissotera reviews
# kai ton arithmo ton reviews.

# Recap: i metavliti genres_with_best_reviewer_with_number_of_reviews exei touples:
# (genre, [userId,numberOfReviews]),
# dld gia kathe genre, exw ton User p exei kanei ta perissotera reviews alla kai ton arithmo ton
reviews.

######
# Opote exw brei ton kalutero reviewer gia kathe genre, menei na mathw gia auton ton user, tin
agapimeni tou
# kai tin ligotero agapimeni tou tainia

only_unique_users = genres_with_best_reviewer_with_number_of_reviews.distinct({column :
'userId'})
# I metavliti only_unique_users, periexei ola ta distinct userId apo tin metavliti
genres_with_best_reviewer_with_number_of_reviews
# to kanw auto, giati mono se autous tous xristes me endiaferei na brw agapimeni kai ligotero
agapimeni tainia

# estw metavliti ratings me oles tis touples tou arxeiou ratings.csv
# den thelw oles tis touples tou ratings, arkei na kratisw mono ta ratings twn xristwn p me endiaferei,
dld
# twn xristwn p vriskotnai sto only_unique_users, giati mono gia autous thelw na mathw tin pio
agapimeni kai ligotero agapimeni tainia

ratings_of_only_the_best_reviewers = only_unique_users.join(ratings)
# sunepws i metavliti ratings_of_only_the_best_reviewers periexei gia ola ta ratings, twn xristwn p me
endiafrei na mathw
# tin pio/ligotero agapimeni tainia

# uparxei omws o periorismos oti gia ton kathe xristi, i pio agapimeni tou tainia tha prepei na anoikei
sto genre
# pou o idios exei bgei san o kaluteros reviewers. Sunepws, exoume ola ta reviews twn xristwn p me
endiafrei na mathw
# ara gia auta ta reviews tha paw kai tha kanw join me ton pinaka genres wste na mathw to genre p
```

```
anoikei i kathe tainia p anaferete to review

ratings_of_only_the_best_reviewers_with_movie_genre =
ratings_of_only_the_best_reviewers_with_movie_genre.join(genres)
# opote kataligoume se touples tis morfis -> movieId, [userId, genre, rating]
# theloume omws kai join me to movies.csv vste gia kathe tainia na mathoume kai to onoma tis kai tin
dimotikotita ths

ratings_of_only_the_best_reviewers_with_movie_genre =
ratings_of_only_the_best_reviewers_with_movie_genre.join(movies)
# exoume userId, movieId, genre, rating, popularity, movieName

# gia kathe toupla tis ratings_of_only_the_best_reviewers_with_movie_genre trexoume tin parakatw
map

map(null, value):
   emit( (value['userId'], value['genre']) , value )

reduce(  (userId, genre) , listOfAllMovieReviews  ):
   #opote apo tin listOfAllReviews, tha prepei na kratisoume to review p exei tin megaluteri
   pio_agapimeni = listOfAllMovieReviews[0]
   ligotero_agapimeni = listOfAllMovieReviews[0]
   foreach movie_review in listOfAllMovieReviews:
      if(movie_review['review'] > pio_agapimeni['review']):
         pio_agapimeni = movie_review
      else if (movie_review['review'] == pio_agapimeni['review']):
            if(movie_review['popularity'] > pio_agapimeni['popularity']):
               pio_agapimeni = movie_review

      if (ligotero_agapimeni['review'] < movie_review['review']):
         ligotero_agapimeni = movie_review
      else if (movie_review['review'] == ligotero_agapimeni['review']):
         if (movie_review['popularity'] > ligotero_agapimeni['popularity']):
            ligotero_agapimeni = movie_review

      emit(genre, userid ,pio_agapimeni, ligotero_agapimeni )

# ara exoume stin metavliti ratings_of_only_the_best_reviewers_with_movie_genre touples tis morfis
# genre, userId (me ta perissotera reviews), pio_agapimeni (tainia), ligotero_agapimeni
# alla den kseroume gia auta ta user_id, posa reviews ekanan sto kathe genre.
# auto to exw brei parapanw stin metavliti genres_with_best_reviewer_with_number_of_reviews
# ara kanw ratings_of_only_the_best_reviewers_with_movie_genre join me to
genres_with_best_reviewer_with_number_of_reviews
genres_with_best_reviewer_with_number_of_reviews.join(ratings_of_only_the_best_reviewers_with_
movie_genre)

#sunepws kataligw se touples tis morfis
# (genre, userId, numberOfReviewsInThatGenre, pio_agapimeni_tainia, bathmologia_pio_agapimenis,
ligotero_agapimeni_tainia, bathmologia_ligotero_agapimenis)
```

Μέρος 2ο

Ζητούμενο 1: Η υλοποίηση βρίσκεται στο αρχείο code/broadcast.py

Ζητούμενο 2: Η υλοποίηση βρίσκεται στο αρχείο code/repartition.py

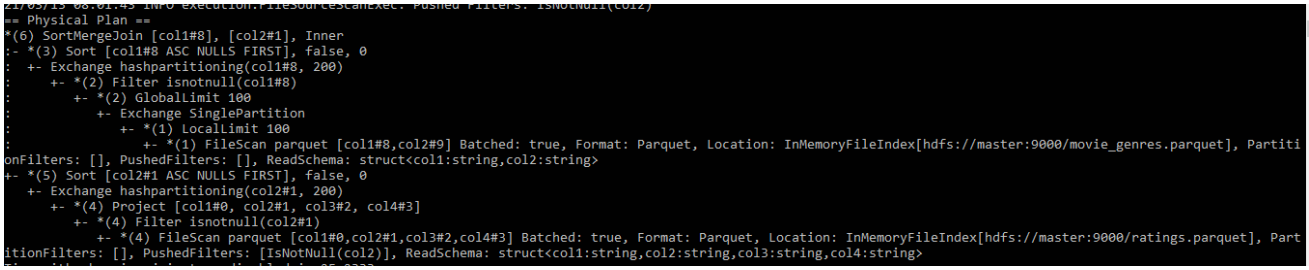
Ζητούμενο 3:

Στο συγκεκριμένο ζητούμενο απομονώσαμε 100 γραμμές του πίνακα movie genres σε ένα άλλο CSV και κάναμε join με τον πίνακα ratings τόσο με broadcast join όσο και με repartition join. Οι χρόνοι εκτέλεσης βρέθηκαν 358.14084219932556 sec και 507.2322027683258 sec αντιστοίχως.

Κάτι τέτοιο ήταν λογικό και αναμενόμενο να συμβεί, καθώς εδώ έχουμε να κάνουμε join δυο πίνακες που το μέγεθος του ενός είναι συγκριτικά πολύ μεγαλύτερο από του άλλου. Όπως αναφέρεται και στο σχετικό paper ,που παρατίθεται με την εκφώνηση της άσκησης, στο broadcast join αντί να μετακινήσουμε και τους δυο πίνακες στο δίκτυο όπως συμβαίνει στα repartition-based joins,μπορούμε να κάνουμε broadcast στον μικρότερο πίνακα ,αποφεύγοντας έτσι το sorting και στους δυο πίνακες αλλά κυρίως την επιβάρυνση της μετακίνησης όλου του μεγαλύτερου πίνακα.

Ζητούμενο 4 :

Στο συγκεκριμένο ζητούμενο κληθήκαμε να συμπληρώσουμε καταλλήλως και να τρέξουμε το δοσμένο script , ώστε να εκτελέσουμε το query που περιέχεται σε αυτό με ή χωρίς την επιλογή του join από τον βελτιστοποιητή.
Κατά την απενεργοποίηση του βελτιστοποιητή ο χρόνος που χρειάστηκε για την εκτέλεση του query ήταν *25.2333 sec*. Το πλάνο εκτέλεσης όπως εμφανίζεται αποτυπώνεται στο παρακάτω στιγμιότυπο.



Όπως φαίνεται για το join των δυο πινάκων (δλδ των πρώτων 100 γραμμών από τον πίνακα movie genres και του πίνακα ratings ,ο οποίος έχει 26024289 εγγραφές) επιλέχθηκε ο αλγόριθμος **Sort Merge** .Ο αλγόριθμος αυτός συνοπτικά έχει ως εξής: Έστω ότι έχουμε δυο datasets, P και Q , με ορισμένες ιδιότητες το καθένα. Ο αλγόριθμος ταξινομεί και τα δύο datasets με βάση το χαρακτηριστικό στο οποίο θα γίνουν join και στη συνέχεια αναζητά κατάλληλες tuples,$p \in P$ και $q \in Q$,με ίδια τιμή στο join attribute,συγχωνεύοντας ουσιαστικά τα δύο σύνολα δεδομένων. Το βήμα ταξινόμησης ομαδοποιεί σε κάθε πίνακα όλες τις tuples με την ίδια τιμή στην join attribute και έτσι διευκολύνει στον εντοπισμό πλειάδων με την ίδια τιμή στην join attribute μεταξύ των δυο πινάκων. Αναλυτικότερα οι tuples σε ένα partition του P συγκρίνονται μόνο με τις tuples στο αντίστοιχο partition του Q .Ο ψευδοκώδικας του συγκεκριμένου αλγορίθμου φαίνεται παρακάτω:

```
p ∈ P;q ∈ Q;gq ∈ Q
while more tuples in inputs do
  while p.a < gq.b do
    advance p
  end while
  while p.a > gq.b do
    advance gq {a group might begin here}
  end while
  while p.a == gq.b do
    q = gq {mark group beginning}
    while p.a == q.b do
      Add ⟨p,q⟩ to the result
      Advance q
    end while
    Advance p {move forward}
  end while
  gq = q {candidate to begin next group}
end while
```

Σημειώνεται ότι προφανώς ο συγκεκριμένος αλγόριθμος θα δώσει καλύτερα αποτελέσματα(από άποψη χρόνου) αν τα δοσμένα datasets είναι ήδη ταξινομημένα ως προς το join attribute.
Στο δεδομένο παράδειγμα λοιπόν, όπως φαίνεται και στο πλάνο εκτέλεσης, ο πίνακας movie_genres γίνεται sort ως προς την col1 και ο πίνακας ratings ως προς την col2,ακολούθως γίνεται partition(-ομαδοποίηση) με βάση τις στήλες αυτές που αποτελούν το join key και στην συνέχεια γίνεται το merge των δύο πινάκων.

Κατά την <u>ενεργοποίηση του βελτιστοποιητή</u> ο χρόνος που χρειάστηκε για την εκτέλεση του query ήταν *4.0618 sec*,σημαντικά μικρότερος δλδ σε σχέση με την προηγούμενη περίπτωση(όπως άλλωστε ήταν αναμενόμενο). Το πλάνο εκτέλεσης όπως εμφανίζεται αποτυπώνεται στο παρακάτω στιγμιότυπο.

```
== Physical Plan ==
*(3) BroadcastHashJoin [col1#8], [col2#1], Inner, BuildLeft
:- BroadcastExchange HashedRelationBroadcastMode(List(input[0, string, false]))
:  +- *(2) Filter isnotnull(col1#8)
:     +- *(2) GlobalLimit 100
:        +- Exchange SinglePartition
:           +- *(1) LocalLimit 100
:              +- *(1) FileScan parquet [col1#8,col2#9] Batched: true, Format: Parquet, Location: InMemoryFileIndex[hdfs://master:9000/movie_genres.parquet], PartitionF
ilters: [], PushedFilters: [], ReadSchema: struct<col1:string,col2:string>
+- *(3) Project [col1#0, col2#1, col3#2, col4#3]
   +- *(3) Filter isnotnull(col2#1)
      +- *(3) FileScan parquet [col1#0,col2#1,col3#2,col4#3] Batched: true, Format: Parquet, Location: InMemoryFileIndex[hdfs://master:9000/ratings.parquet], PartitionF
ilters: [], PushedFilters: [IsNotNull(col2)], ReadSchema: struct<col1:string,col2:string,col3:string,col4:string>
```

Όπως φαίνεται για το join των δυο πινάκων χρησιμοποιήθηκε ο αλγόριθμος **Broadcast Hash Join.** Ο αλγόριθμος αυτός είναι ιδανικός όταν πρόκειται να γίνουν join πίνακες , που το μέγεθος του ενός είναι συγκριτικά πολύ μικρότερο από το μέγεθος του άλλου και χαρακτηρίζεται ως map-phase only.Παρακάτω φαίνεται ο ψευδοκώδικας του συγκεκριμένου αλγορίθμου:

```
Init ()
  if R not exist in local storage then
      remotely retrieve R
      partition R into p chunks R_1..R_p
      save R_1..R_p to local storage

  if R < a split of L then
      H_R ← build a hash table from R_1..R_p
  else
      H_L1..H_Lp ← initialize p hash tables for L

Map (K: null, V: a record from an L split)
  if H_R exist then
      probe H_R with the join column extracted from V
      for each match r from H_R do
          emit (null, new_record(r, V))
  else
      add V to an H_Li hashing its join column

Close ()
  if H_R not exist then
      for each non-empty H_Li do
          load R_i in memory
          for each record r in R_i do
              probe H_Li with r's join column
              for each match l from H_Li do
                  emit (null, new_record(r, l))
```

Στην συνάρτηση init() ελέγχεται αν το R είναι ήδη αποθηκευμένο στο τοπικό file system. Αν όχι ανακτάται από το DFS, γίνεται partition με βάση το join key και έπειτα αποθηκεύεται στο τοπικό file system. Ο αλγόριθμος αποφασίζει δυναμικά αν θα «χτίσει» το hash table πάνω στο L ή στο R. Αν το R είναι μικρότερο από το split του L, όλα τα partitions του R φορτώνονται στην μνήμη ,στο hash table.Έπειτα η συνάρτηση map «εξάγει» το join key για κάθε tuple του L και το χρησιμοποιεί για αναζήτηση στο hash table.Για κάθε match γίνεται το κατάλληλο join. Στην

περίπτωση όμως που το μέγεθος του spit του L είναι μικρότερο από το μέγεθος του R,το join δεν γίνεται στην συνάρτηση map. Αρχικά στην init() το L γίνεται partition με τον ίδιο τρόπο που γίνεται το R(με βάση το Join key δλδ).Ακολούθως στην συνάρτηση close() τα partitions των R και L γίνονται join. Στην δική μας περίπτωση όπως βλέπουμε από το πλάνο εκτέλεσης (και όπως αναμενόταν σύμφωνα με την παραπάνω ανάλυση του αλγορίθμου),ο πίνακας ο οποίος γίνεται partition είναι ο movie_genres (εννοώντας τις 100 πρώτες γραμμές του) και ακολουθείται ο αλγόριθμος όπως περιγράφεται παραπάνω.
Συγκεντρωτικά οι χρόνοι εκτέλεσης των queries για κάθε περίπτωση φαίνονται στο παρακάτω ραβδόγραμμα.