ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΒΑΣΕΩΝ ΓΝΩΣΕΩΝ ΚΑΙ ΔΕΔΟΜΕΝΩΝ

Προχωρημένα Θέματα Βάσεων Δεδομένων

Ακ. έτος 2022-2023, 9ο Εξάμηνο

Εξαμηνιαία Εργασία

Ομάδα 92

Κωνσταντίνος Σιδέρης, Α.Μ.:03118134

Χριστίνα Διαμαντή, Α.Μ.:03118203

Περιεχόμενα

1 GitHub Link

https://github.com/kon-si/ntua_atds

2 Προαπαιτούμενες Υποδομές

Χρησιμοποιούμε δύο Virtual Machines, τον Master και τον Slave. Κάθε VM διαθέτει 4 CPU cores, 8GB μνήμη RAM, κύρια μνήμη 30GB και διευθύνσεις public IPv6. Επιπρόσθετα, ο Master διαθέτει public IPv4. Επικοινωνούν μέσω τοπικού δικτύου IPv4. Τρέχουν λειτουργικό σύστημα Ubuntu 18.04.6 LTS.

2.1 Οι Τεχνολογίες που χρησιμοποιήθηκαν

- OpenJDK 1.8.0_352
- Apache Hadoop 3.3.4
- Apache Spark 3.3.1
- Scala 2.12.17
- SBT 1.8.2

Τα αναλυτικά βήματα για τις απαραίτητες εγκαταστάσεις και ρυθμίσεις περιγράφονται με λεπτομέρεια στο αρχείο README.md[1] του φακέλου setup του GitHub repository που δίνεται ως σύνδεσμος στην αρχή της παρούσας αναφοράς.

2.1.1 Εγκατάσταση HDFS

Το Hadoop HDFS χρησιμοποιείται συχνά με το Apache Spark για επεξεργασία μεγάλων δεδομένων, επειδή το Spark μπορεί να αξιοποιήσει το HDFS για αποθήκευση και ανάκτηση δεδομένων. Το HDFS παρέχει επεκτάσιμη, αξιόπιστη και κατανεμημένη αποθήκευση για μεγάλα δεδομένα και το Spark μπορεί να επεξεργάζεται αυτά τα δεδομένα παράλληλα, καθιστώντας το κατάλληλο για επεξεργασία δεδομένων μεγάλης κλίμακας. Αυτός ο συνδυασμός επιτρέπει την αποτελεσματική επεξεργασία μεγάλων δεδομένων και βοηθά επίσης στον χειρισμό αστοχιών στο cluster.
Συνοπτικά τα βήματα που χρειάστηκε να ακολουθήσουμε για την εγκατάσταση του Apache Hadoop είναι τα εξής:
1. Εγκατάσταση της JAVA στα μηχανήματα.
2. Δημιουργία ζεύγους δημοσίου/ιδιωτικού κλειδιού SSH στον Master και αντιγραφή του στον Slave, έτσι ώστε ο πρώτος να μπορεί να συνδεθεί με ασφάλεια στον δεύτερο χωρίς κωδικό πρόσβασης.
3. Εγκατάσταση stable version του Apache Hadoop στα μηχανήματα.
4. Ρύθμιση των Μεταβλητών Περιβάλλοντος για το Hadoop στο αρχείο .bashrc .
5. Κατάλληλη επεξεργασία των απαραίτητων configuration files.
6. Εκτέλεση των κατάλληλων εντολών για format και έπειτα εκκίνηση του HDFS.

[1] Οδηγίες Εγκατάστασης Spark Hadoop: https://github.com/kon-si/ntua_atds/blob/master/setup/README.md

2.1.2 Εγκατάσταση Yarn

Το Apache Hadoop Yarn είναι η τεχνολογία διαχείρισης πόρων και προγραμματισμού εργασιών (resource manager and job scheduler) του Apache Hadoop. Το Yarn είναι υπεύθυνο για την κατανομή πόρων συστήματος στις διάφορες εφαρμογές που εκτελούνται σε ένα Hadoop cluster και τον προγραμματισμό εργασιών που θα εκτελεστούν σε διαφορερικά cluster nodes. Το Yarn εγκαθίσταται στο σύστημα by default με την εγκατάσταση του Apache Hadoop, επομένως δεν χρειάζεται πρόσθετη εγκατάσταση, χρειάζεται όμως η ρύθμιση μερικών παραμέτρων σχετικές με τη χρήση του yarn και ορισμένες ρυθμίσεις σχετικές με τη διαχείριση μνήμης.

2.1.3 Εγκατάσταση Spark

Το Apache Spark είναι ένα open-source κατανεμημένο υπολογιστικό σύστημα που χρησιμοποιείται για επεξεργασία και ανάλυση μεγάλων δεδομένων. Έχει σχεδιαστεί για να χειρίζεται εργασίες επεξεργασίας δεδομένων μεγάλης κλίμακας με γρήγορο και αποτελεσματικό τρόπο και παρέχει έναν αριθμό API υψηλού επιπέδου για επεξεργασία δεδομένων, machine learning και επεξεργασία γραφημάτων.
Το Spark θα εγκατασταθεί στο Hadoop Cluster, επομένως τα βήματα εγκατάστασης θα πρέπει να γίνουν μετά την επιτυχή εγκατάσταση του Apache Hadoop και την πραγματοποίηση των απαραίτητων ρυθμίσεων Yarn.

2.1.4 Εγκατάσταση Scala

Η έκδοση 3.3.1 του Spark απαιτεί έκδοση Scala 2.12/2.13. Εγκαθιστούμε την έκδοση 2.12.17 της Scala.
Επιλέξαμε να εργαστούμε χρησιμοποιώντας Scala (εναντι της Python) προκειμένου αφενός να εξοικειωθούμε με μια νέα γλώσσα προγραμματισμού και αφετέρου για λόγους καλύτερης χρονικής απόδοσης των ερωτημάτων. Η Scala είναι ταχύτερη από την Python όσον αφορά την επεξεργασία μεγάλων δεδομένων, επειδή είναι μια statically typed γλώσσα και εκτελείται στο Java Virtual Machine (JVM). Το Spark είναι γραμμένο σε Scala, επομένως όταν χρησιμοποιείται η Scala, ο κώδικας επεξεργασίας δεδομένων μπορεί να εκτελείται εγγενώς στο JVM και να εκμεταλλεύεται τις βελτιστοποιήσεις του JVM.

2.1.5 Εγκατάσταση SBT

Το SBT είναι open-source εργαλείο δημιουργίας εφαρμογών για Scala και Java. Το χρησιμοποιούμε ώστε να μετατρέψουμε τον κώδικα Scala σε εκτελέσιμα αρχεία JAR.

3 Εκκαθάριση Δεδομένων

Το σύνολο δεδομένων που χρησιμοποιείται για την εργασία είναι το TLC Trip Record Data.Πιο συγκεκριμένα, χρησιμοποιήσαμε δεδομένα από την Yellow Taxi Trip Records για τους μήνες Ιανουάριο έως Ιούνιο του 2022. Παρ'όλ'αυτά παρατηρήσαμε ότι στα δεδομένα μας υπήρχαν μερικά tuples που αφορούσαν ημερομηνίες εκτός του χρονικού πλαισίου που μας ενδιαφέρει. Γι' αυτό το λόγο φιλτράρουμε τα δεδομένα ώστε να πραγματοποιούνται αναζητήσεις μόνο σε χρήσιμα tuples. Στο ίδιο πνεύμα, πραγματοποιήθηκε έλεγχος ανά column για το πλήθος των πιθανών NULL τιμών για τα χαρακτηριστικά που ελέγχονται στα ζητούμενα Queries. Ο έλεγχος έγινε αφενός για να έχουμε καλύτερη εποπτεία επί της ποιότητας των δεδομένων και αφετέρου ώστε, σε περίπτωση εμφάνισης tuples με σημαντικά κενά τιμών, αυτά να διαγραφούν καθώς δεν θα είχε νόημα, από άποψης απόδοσης, η συμπερίληψή τους στις αναζητήσεις που θα ακολουθήσουν.

Ελέγχθηκαν οι στήλες: Tolls_amount, Tip_amount, Total_amount, Trip_distance, PULocationID, DOLocationID και Passenger_count.
Εκ των προαναφερθέντων μόνο η στήλη Passenger_count παρουσίασε μη μηδενικό πλήθος NULL τιμών (671,271 NULL values στις 19,798,361 σειρές). Αυτό δικαιολογείται από το γεγονός ότι το συγκεκριμένο χαρακτηριστικό δεν καταγράφεται αυτόματα αλλά απαιτεί να περαστεί στο σύστημα από τον ίδιο τον οδηγό. Παρά το πλήθος των NULL στην προκειμένη περίπτωση, επειδή αφορούν το συγκεκριμένο χαρακτηριστικό και θα χάνονταν σημαντικά δεδομένα σε πιθανή διαγραφή των σειρών εμφάνισης των κενών, δεν έγινε κάποιο περαιτέρω φιλτράρισμα των δεδομένων.

4 Εκτέλεση Queries

4.1 Δημιουργία εκτελέσιμων JAR

Για την σωστή λειτουργία του το SBT απαιτεί την αποθήκευση του κώδικα με συγκεκριμένη δομή φακέλων. Μέσα στον φάκελο που είναι εγκατεστημένο το Spark δημιουργούμε το αρχείο build.sbt με το οποίο θα κάνουμε compile την εφαρμογή μας και τοποθετούμε τον κώδικα μας στον φάκελο src/main/scala/. Για την δημιουργία του εκτελέσιμου εκτελούμε την εντολή `sbt package` [2].

4.2 Λήψη Μετρήσεων

Εκτελούμε τα queries στο Yarn cluster χρησιμοποιώντας το εργαλείο spark-submit. Για να αντισταθμίσουμε το data skew που παρουσιάζεται κατά την εκτέλεση Spark applications οι μετρήσεις των χρόνων εκτέλεσης του κάθε query λαμβάνονται από τον μέσο όρο 20 εκτελέσεων.

4.3 Ζητούμενο 1

Τα βήματα εγκατάστασης του HDFS και του Spark περιγράφονται αναλυτικά στο αρχείο README.md[3] του φακέλου setup του GitHub repository που δίνεται ως σύνδεσμος στην αρχή της παρούσας αναφοράς και γι' αυτό δεν θα αναλυθούν εδώ.
Ο κώδικας Scala για την δημιουργία των ζητούμενων Dataframes και RDDs φαίνεται παρακάτω:

```
/*Create Dataframes*/
val lookupDF =
    sqlContext.read.schema(lookupSchema).options(Map("header"->"true"))
    .csv("hdfs://192.168.0.1:9000/user/user/lookup/taxi+_zone_lookup.csv")

val recordsDF = spark.read.parquet("clean_records/records.parquet")

/*Create RDDs*/
val recordsRDD = recordsDF.rdd
val lookupRDD = lookupDF.rdd
```

4.4 Ζητούμενο 2

Τα Q1 και Q2 εκτελέσθηκαν με χρήση του DataFrame/ SQL API. Παρακάτω φαίνεται η λεκτική περιγραφή και ο κώδικας για το κάθε Query καθώς και τα αποτελέσματα που αυτά έφεραν.

[2] Spark Self-Contained Apps: https://spark.apache.org/docs/latest/quick-start.html#self-contained-applications

[3] Οδηγίες Εγκατάστασης Spark Hadoop: https://github.com/kon-si/ntua_atds/blob/master/setup/README.md

4.4.1 Q1

Να βρεθεί η διαδρομή με το μεγαλύτερο φιλοδώρημα (tip) τον Μάρτιο και σημείο άφιξης το ”Battery Park”.

```
recordsDF.filter(month(col("tpep_pickup_datetime"))===3)
             .join(lookupDF,
                recordsDF("DOLocationID").equalTo(lookupDF("LocationID")))
             .filter(col("Zone").equalTo("Battery Park"))
             .withColumnRenamed("Zone","DOZone")
             .drop("LocationID","Borough","service_zone")
             .orderBy(desc("Tip_amount"))
             .limit(1)
             .show()
```

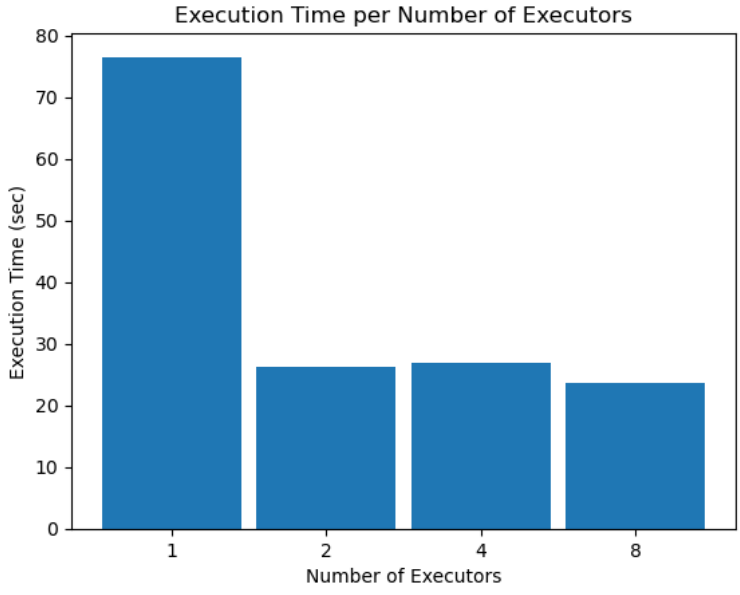
Αποτελέσματα Q1:

VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance
2	2022-03-17 12:27:47	2022-03-17 12:27:58	1.0	0.0
RatecodeID	**store_and_fwd_flag**	**PULocationID**	**DOLocationID**	**payment_type**
1.0	N	12	12	1
fare_amount	**extra**	**mta_tax**	**tip_amount**	**tolls_amount**
2.5	0.0	0.5	40.0	0.0
improvement_surchange	**total_amount**	**congestion_surcharge**	**airport_fee**	**DOZone**
0.3	45.8	2.5	0.0	Battery Park

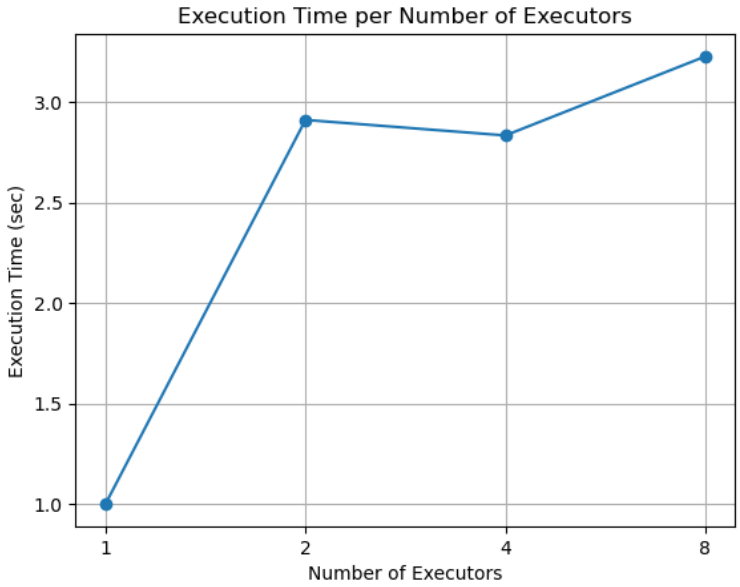
Χρόνοι Εκτέλεσης για το Q1:

Executors	1	2	4	8
Time (sec)	76.53	26.28	26.99	23.70

Διαγράμματα για το Q1:



Σχήμα 1: Διάγραμμα Χρόνων Εκτελέσεων Q1



Σχήμα 2: Διάγραμμα Speedup Q1

Σχόλια: Παρατηρούμε ότι με την αύξηση των executors ο χρόνος εκτέλεσης μειώνεται. Αυτό είναι αναμενόμενο καθώς, στο Q1 εκτελούμε join μεταξύ των records και των lookup zones το οποίο επωφελείται από παράλληλη επεξεργασία δεδομένων. Με την αύξηση των executors από 2 σε 4 και 8 δεν παρατηρούμε ιδιαίτερη βελτίωση της επίδοσης καθώς, οι πόροι του συστήματος, συμπεριλαμβανομένου της RAM και της cache, διαμοιράζονται μεταξύ των executors. Ως αποτέλεσμα η βελτίωση της επίδοσης λόγω του παραλληλισμού αντισταθμίζεται από τα αυξημένα memory access που πρέπει να γίνουν αφού η cache, λόγω του μειωμένου μεγέθους της, περιέχει λιγότερες εγγραφές του πίνακα.

4.4.2 Q2

Να βρεθεί, για κάθε μήνα, η διαδρομή με το υψηλότερο ποσό στα διόδια. Αγνοήστε μηδενικά ποσά.

```
recordsDF.groupBy(month(col("tpep_pickup_datetime")).as("Month"))
        .agg(max("Tolls_amount").as("Max_Tolls_amount"))
        .orderBy("Month")
        .show()
```

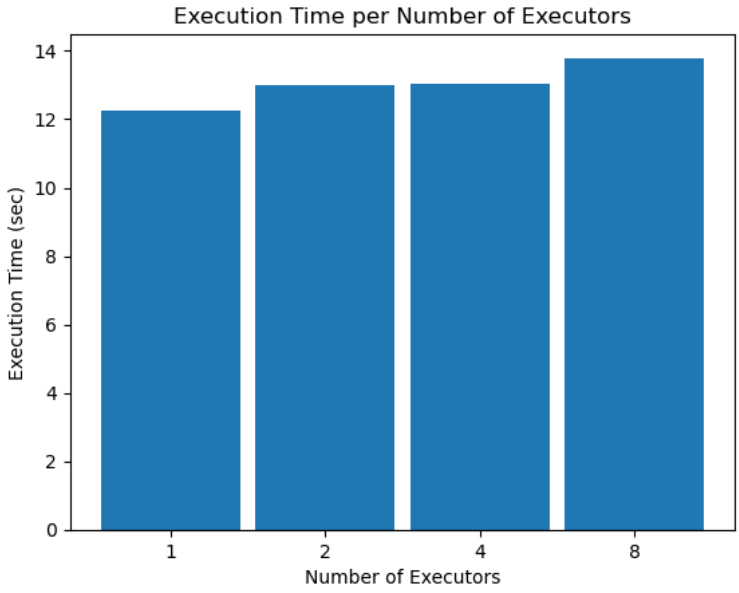
Αποτελέσματα Q2:

Month	Max_Tolls_Amount
1	193.3
2	95.0
3	235.7
4	911.87
5	813.75
6	800.09

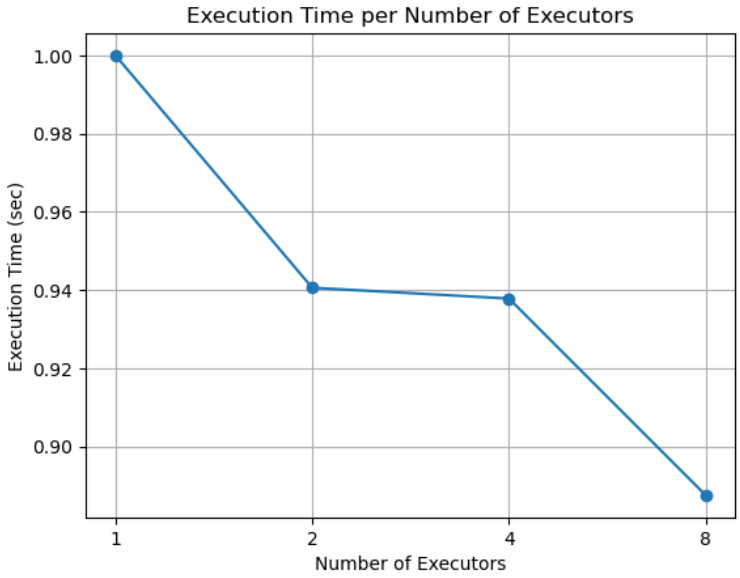
Χρόνοι Εκτέλεσης για το Q2:

Executors	1	2	4	8
Time (sec)	12.24	13.01	13.05	13.79

Διαγράμματα για το Q2:



Σχήμα 3: Διάγραμμα Χρόνων Εκτελέσεων Q2



Σχήμα 4: Διάγραμμα Speedup Q2

Σχόλια: Παρατηρούμε ότι η αύξηση των executors επιφέρει μείωση της επίδοσης. Το παραπάνω query διατρέχει όλα records ώστε να γίνει ομαδοποίηση με βάση τον μήνα συνεπώς, ευνοείται από μεγάλο και συνεχόμενο χώρο μνήμης. Με την αύξηση των executors ο χώρος μνήμης κατακερματίζεται οπότε προκαλείται χρονικό overhead, λόγω των αυξημένων προσβάσεων μνήμης και της

επικοινωνίας μεταξύ των executors που εργόζονται σε διαφορετικά μέρη του DataFrame, το οποίο δεν αντισταθμίζεται από τον παραλληλισμό.

4.5 Ζητούμενο 3

Το Q3 εκτελέσθηκε με χρήση του RDD API και του DataFrame/ SQL API. Παρακάτω φαίνεται η λεκτική περιγραφή και ο κώδικας για το Query καθώς και τα αποτελέσματα που αυτό έφερε.

Να βρεθεί, ανά 15 ημέρες, ο μέσος όρος της απόστασης και του κόστους για όλες τις διαδρομές με σημείο αναχώρησης διαφορετικό από το σημείο άφιξης.

4.5.1 Q3: RDD API

```
def tperiod(x:Int) : String = {
  if (x <= 15)
    "{2022-01-01 00:00:00, 2022-01-16 00:00:00}"
  else if (x <= 30)
    "{2022-01-16 00:00:00, 2022-01-31 00:00:00}"
  else if (x <= 45)
    "{2022-01-31 00:00:00, 2022-02-15 00:00:00}"
  else if (x <= 60)
    "{2022-02-15 00:00:00, 2022-03-02 00:00:00}"
  else if (x <= 75)
    "{2022-03-02 00:00:00, 2022-03-17 00:00:00}"
  else if (x <= 90)
    "{2022-03-17 00:00:00, 2022-04-01 01:00:00}"
  else if (x <= 105)
    "{2022-04-01 01:00:00, 2022-04-16 01:00:00}"
  else if (x <= 120)
    "{2022-04-16 01:00:00, 2022-05-01 01:00:00}"
  else if (x <= 135)
    "{2022-05-01 01:00:00, 2022-05-16 01:00:00}"
  else if (x <= 150)
    "{2022-05-16 01:00:00, 2022-05-31 01:00:00}"
  else if (x <= 165)
    "{2022-05-31 01:00:00, 2022-06-15 01:00:00}"
  else if (x <= 180)
    "{2022-06-15 01:00:00, 2022-06-30 01:00:00}"
  else
    "{2022-06-30 01:00:00, 2022-07-15 01:00:00}"
  }

recordsRDD.filter(T => T.getAs[Long]("PULocationID") !=
    T.getAs[Long]("DOLocationID"))
  .map(T => (
    tperiod(dateFormat.format(T.getTimestamp(1)).toInt)
    ,
    (T.getAs[Double]("trip_distance"), T.getAs[Double]("total_amount")))
  )
  .aggregateByKey(((0.0, 0.0), (0.0, 0.0)))
  (
```

```
  (C, V:(Double, Double)) => ((C._1._1 + V._1, C._1._2 + 1.0), (C._2._1 +
     V._2, C._2._2 + 1.0)),
  (C1, C2) => ((C1._1._1 + C2._1._1, C1._1._2 + C2._1._2), (C1._2._1 +
     C2._2._1, C1._2._2 + C2._2._2))
)
.map(T => (T._1, T._2._1._1 / T._2._1._2, T._2._2._1 / T._2._2._2))
.sortBy(T => T._1)
.collect()
.foreach(println)
```

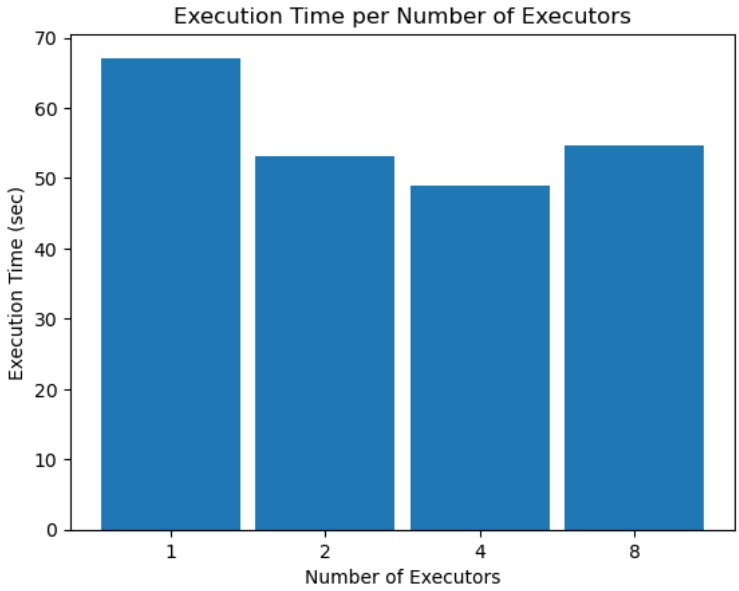
Αποτελέσματα Q3 RDD:

time_period | avg(trip_distance) | avg(total_amount)
(2022-01-01 00:00:00, 2022-01-16 00:00:00,5.576410377852007,19.903702637879007)
(2022-01-16 00:00:00, 2022-01-31 00:00:00,4.804840472309411,19.03660791389491)
(2022-01-31 00:00:00, 2022-02-15 00:00:00,5.950485844928086,19.553891327978455)
(2022-02-15 00:00:00, 2022-03-02 00:00:00,6.1857672125677,20.17207809365826)
(2022-03-02 00:00:00, 2022-03-17 00:00:00,6.60698631990843,20.692357713183547)
(2022-03-17 00:00:00, 2022-04-01 01:00:00,5.532999252101388,21.121659489582353)
(2022-04-01 01:00:00, 2022-04-16 01:00:00,5.679323077938378,21.5155590945829)
(2022-04-16 01:00:00, 2022-05-01 01:00:00,5.800344707646387,21.428088376188708)
(2022-05-01 01:00:00, 2022-05-16 01:00:00,6.249697852127191,21.921570348909174)
(2022-05-16 01:00:00, 2022-05-31 01:00:00,7.999063222469341,22.806499070421843)
(2022-05-31 01:00:00, 2022-06-15 01:00:00,6.378971191608972,22.452110839872283)
(2022-06-15 01:00:00, 2022-06-30 01:00:00,6.153370128239474,22.352167683521646)
(2022-06-30 01:00:00, 2022-07-15 01:00:00,5.79851745131752,22.026305990619665)

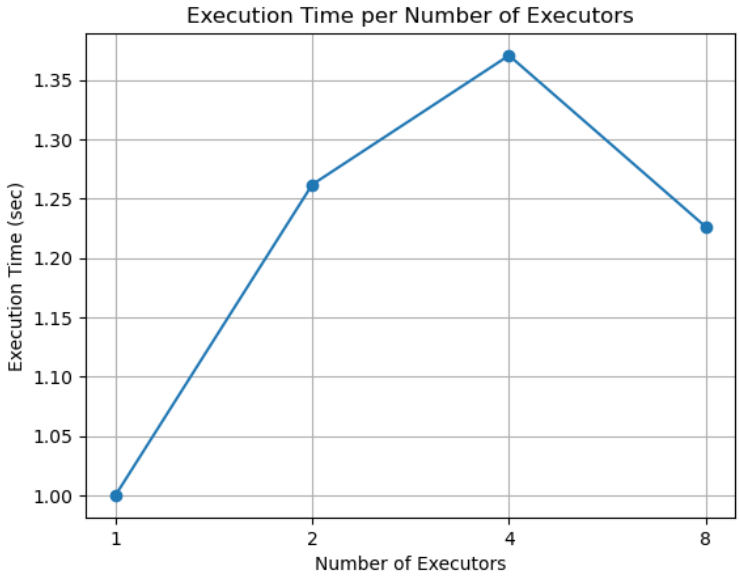
Χρόνοι Εκτέλεσης για το Q3 RDD:

Executors	1	2	4	8
Time (sec)	67.10	53.18	48.96	54.72

Διαγράμματα για το Q3 RDD:



Σχήμα 5: Διάγραμμα Χρόνων Εκτελέσεων Q3 RDD



Σχήμα 6: Διάγραμμα Speedup Q3 RDD

Σχόλια:Η αύξηση των executors επιφέρει βελτίωση της επίδοσης καθώς, τα δεδομένα ομαδοποιούνται ανά 15 μέρες και κάθε ομάδα επεξεργάζεται παράλληλα. Η αύξηση από 4 σε 8 executors έχει ως αποτέλεσμα μικρή αύξηση του χρόνου. Αυτό οφείλεται στο γεγονός ότι η μνήμη κατακερματίζεται ανάμεσα στους executors με αποτέλεσμα να μην χωράει ολόκληρο το παράθυρο 15 ημερών στην cache και να αυξάνεται το overhead από τις προσβάσεις μνήμης.

4.5.2 Q3: DataFrame API

```
recordsDF.filter(col("PULocationID") =!= col("DOLocationID"))
  .groupBy(window(col("tpep_pickup_datetime"), "15 days", "15 days", "70
     hours").as("time_period"))
  .avg("trip_distance", "total_amount")
  .orderBy("time_period")
  .show(false)
```

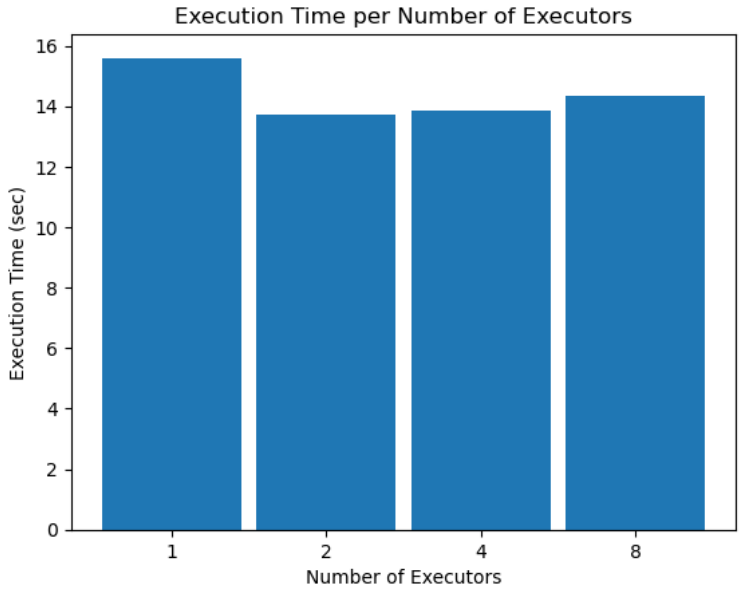
Αποτελέσματα Q3 DataFrame:

time_period	avg(trip_distance)	avg(total_amount)
2022-01-01 00:00:00, 2022-01-16 00:00:00	5.576410377852007	19.903702637879007
2022-01-16 00:00:00, 2022-01-31 00:00:00	4.804840472309411	19.03660791389491
2022-01-31 00:00:00, 2022-02-15 00:00:00	5.950485844928086	19.553891327978455
2022-02-15 00:00:00, 2022-03-02 00:00:00	6.1857672125677	20.17207809365826
2022-03-02 00:00:00, 2022-03-17 00:00:00	6.60698631990843	20.692357713183547
2022-03-17 00:00:00, 2022-04-01 01:00:00	5.524788048396609	21.118287307889744
2022-04-01 01:00:00, 2022-04-16 01:00:00	5.679221475787281	21.513246092852082
2022-04-16 01:00:00, 2022-05-01 01:00:00	5.800096624033662	21.431010174428504
2022-05-01 01:00:00, 2022-05-16 01:00:00	6.255316989977509	21.929327001976382
2022-05-16 01:00:00, 2022-05-31 01:00:00	8.000620246152229	22.80847294454238
2022-05-31 01:00:00, 2022-06-15 01:00:00	6.372734051706068	22.444346976981922
2022-06-15 01:00:00, 2022-06-30 01:00:00	6.154210472858656	22.352414801280606
2022-06-30 01:00:00, 2022-07-15 01:00:00	5.93465896170917	22.091536970956383

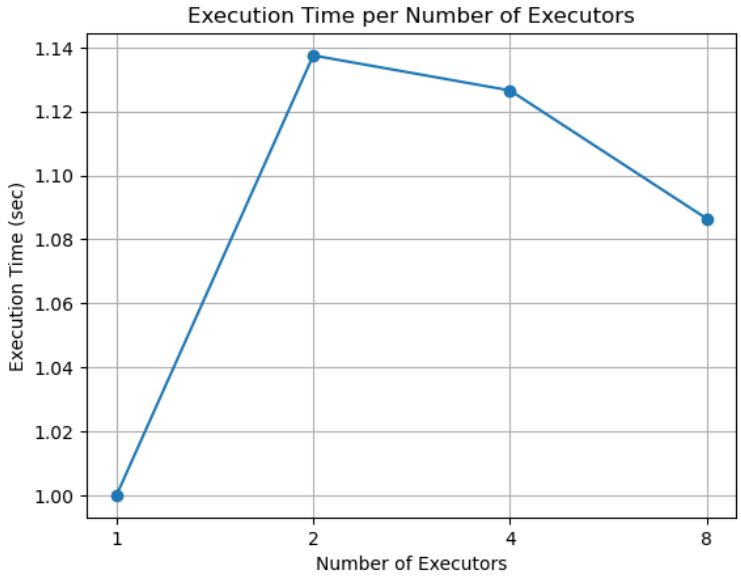
Χρόνοι Εκτέλεσης για το Q3 DF:

Executors	1	2	4	8
Time (sec)	15.59	13.71	13.84	14.35

Διαγράμματα για το Q3 DF:



Σχήμα 7: Διάγραμμα Χρόνων Εκτελέσεων Q3 DataFrame



Σχήμα 8: Διάγραμμα Speedup Q3 DataFrame

Σχόλια:Η επίδοση με χρήση του DataFrame API είναι σημαντικά βελτιωμένη σε σχέση με το query χρησιμοποιώντας το RDD API. Παρόλα αυτά, η τάση των χρόνων εκτέλεσης με την αύξηση των executors είναι ίδια με την τάση που παρατηρήθηκε με το RDD API καθώς η ομαδοποίηση των δεδομένων και ο διαχωρισμός των πόρων του συστήματος είναι ίδιος.

4.6 Ζητούμενο 4

Τα Q4 και Q5 εκτελέσθηκαν με χρήση του DataFrame/ SQL API. Παρακάτω φαίνεται η λεκτική περιγραφή και ο κώδικας για το κάθε Query καθώς και τα αποτελέσματα που αυτά έφεραν.

4.6.1 Q4

Να βρεθούν οι τρεις μεγαλύτερες (top 3) ώρες αιχμής ανά ημέρα της εβδομάδος, εννοώντας τις ώρες της ημέρας με τον μεγαλύτερο αριθμό επιβατών σε μια κούρσα ταξί. Ο υπολογισμός αφορά όλους τους μήνες.

```
recordsDF.withColumn("day_of_week", dayofweek(col("tpep_pickup_datetime")))
          .withColumn("hour", hour(col("tpep_pickup_datetime")))
          .groupBy("day_of_week", "hour")
          .agg(avg(col("Passenger_count")).as("Avg_Passenger_Count"))
          .withColumn("rank",
             row_number().over(Window.partitionBy("day_of_week")
          .orderBy(col("Avg_Passenger_Count").desc)))
          .filter(col("rank") < 4)
          .show(50)
```

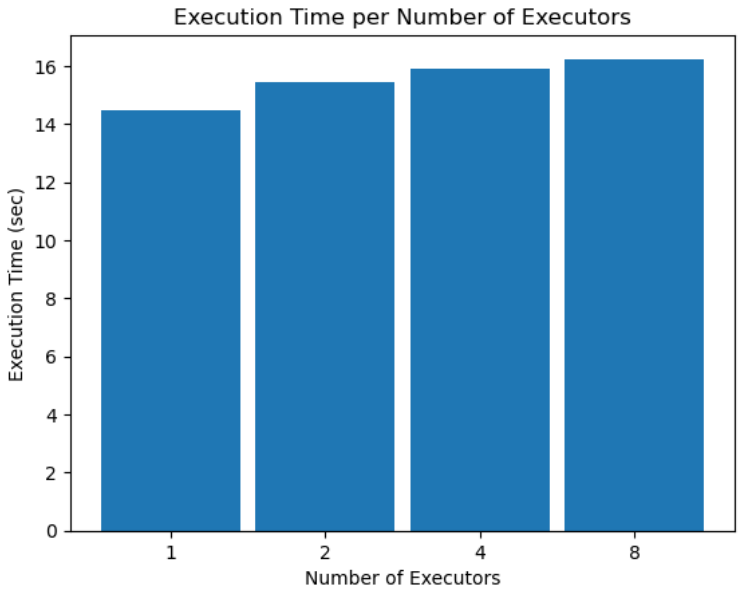
Αποτελέσματα Q4:

day_of_week	hour	Avg_Passenger_Count	rank
1	0	1.5299456507188562	1
1	1	1.527838567375201	2
1	2	1.5080726185191242	3
2	0	1.4679887711672552	1
2	1	1.4442867916810471	2
2	2	1.4231993989051486	3
3	0	1.4200313882151518	1
3	1	1.4175124740006593	2
3	2	1.4104520814693964	3
4	1	1.4088480212656305	1
4	0	1.4012291857176276	2
4	2	1.4011489645958584	3
5	23	1.405311916503088	1
5	1	1.4025550243918357	2
5	0	1.4010369554672855	3
6	23	1.475576918073731	1
6	22	1.444813976205668	2
6	2	1.4230581143524386	3
7	23	1.522606766277207	1
7	22	1.5068176194011382	2
7	0	1.4993154284898547	3

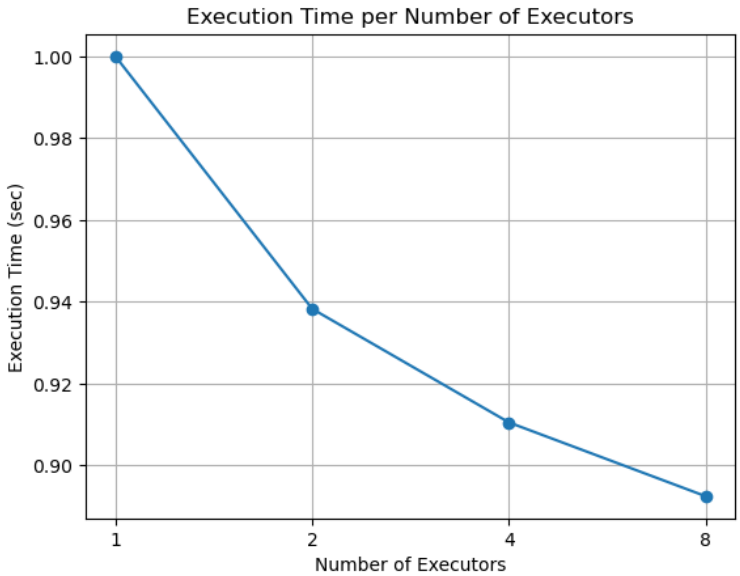
Χρόνοι Εκτέλεσης για το Q4:

Executors	1	2	4	8
Time (sec)	14.49	15.44	15.92	16.24

Διαγράμματα για το Q4:



Σχήμα 9: Διάγραμμα Χρόνων Εκτελέσεων Q4



Σχήμα 10: Διάγραμμα Speedup Q4

Σχόλια:Παρατηρούμε ανοδική τάση του χρόνου εκτέλεσης με την αύξηση των executors. Αυτό οφείλεται στο γεγονός ότι προσθέτουμε δύο νέες στήλες, την μέρα της εβδομάδας και την ώρα, στο DataFrame δηλαδή, διατρέχουμε όλα τα records. Αυτή η διεργασία δεν επωφελείται από τον

κατακερματισμό της RAM και της cache καθώς, αφού χωράνε λιγότερες εγγραφές στην cache απαιτούνται περισσότερες προσβάσεις μνήμης ώστε να διατρέξουμε τον πίνακα.

4.6.2 Q5

Να βρεθούν οι κορυφαίες πέντε (top 5) ημέρες ανά μήνα στις οποίες οι κούρσες είχαν το μεγαλύτερο ποσοστό σε tip. Να βρεθούν οι κορυφαίες πέντε (top 5) ημέρες ανά μήνα στις οποίες οι κούρσες είχαν το μεγαλύτερο ποσοστό σε tip.

```
recordsDF.withColumn("tip_percentage",
   col("tip_amount") / col("total_amount") * 100)
 .withColumn("date", to_date(col("tpep_dropoff_datetime")))
 .groupBy("date")
 .avg("tip_percentage")
 .withColumn("row",
   row_number()
   .over(Window
     .partitionBy(month(col("date")))
     .orderBy(desc("avg(tip_percentage)"))
   )
 )
 .filter(col("row") <= 5)
 .select("date", "avg(tip_percentage)")
 .orderBy("date")
 .show(30)
```

Αποτελέσματα Q5:

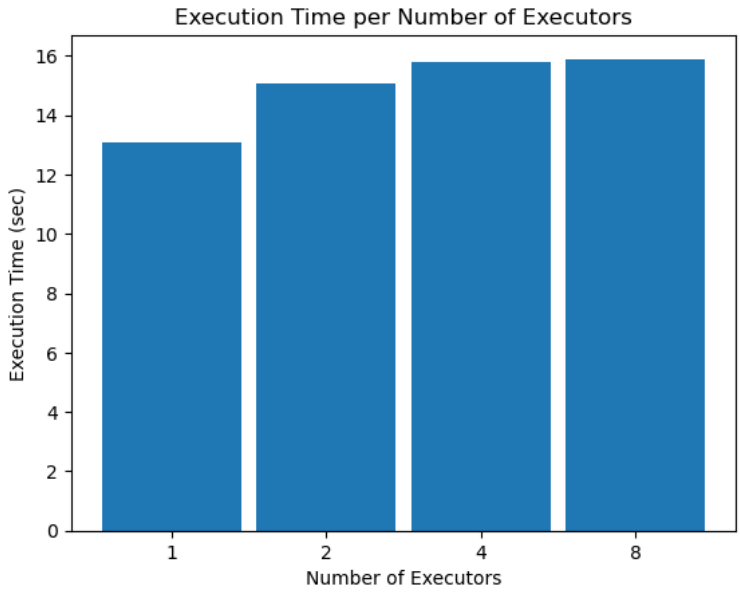
date	avg(tip_percentage)
2022-01-15	12.493406955902126
2022-01-22	12.506777958202651
2022-01-23	12.451826792945813
2022-01-29	13.263520115480489
2022-01-30	12.648425842313266
2022-02-04	12.443689932700892
2022-02-05	12.596319021084778
2022-02-06	12.521082372639468
2022-02-10	12.402413854262088
2022-02-13	12.485816604312875
2022-03-09	12.520434884013525
2022-03-10	12.451389826354061
2022-03-12	12.429288681713686
2022-03-30	12.433909344452855
2022-03-31	12.402137582534634
2022-04-01	12.323353156655472

2022-04-06	12.390087899760692
2022-04-07	12.434342508124175
2022-04-27	12.383722371356122
2022-04-28	12.356539174220126
2022-05-04	12.343572953447529
2022-05-12	12.376205852394001
2022-05-18	12.341170350917928
2022-05-19	12.317754279200157
2022-05-25	12.327037068438422
2022-06-08	12.316573275455053
2022-06-09	12.384192792190577
2022-06-15	12.308897250490785
2022-06-16	12.37477971189518
2022-06-23	12.323035352339327

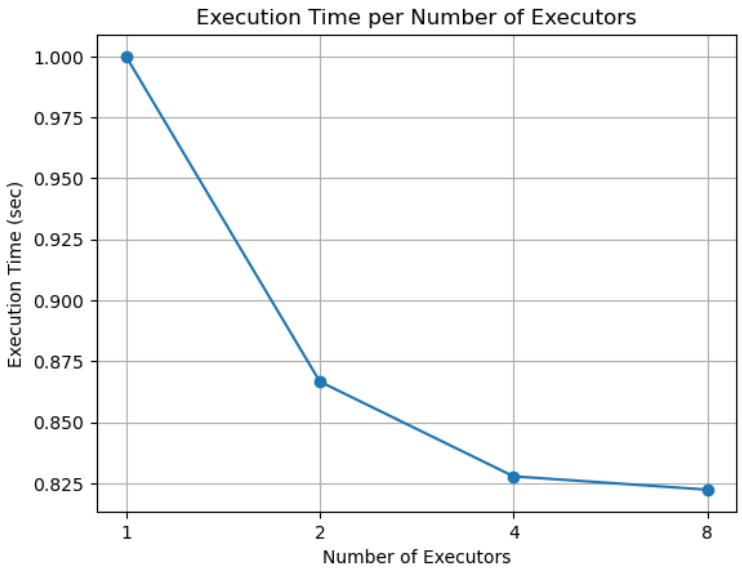
Χρόνοι Εκτέλεσης για το Q5:

Executors	1	2	4	8
Time (sec)	13.07	15.08	15.79	15.89

Διαγράμματα για το Q5:



Σχήμα 11: Διάγραμμα Χρόνων Εκτελέσεων Q5



Σχήμα 12: Διάγραμμα Speedup Q5

Σχόλια:Ομοίως με το query 5, παρατηρούμε πτωτική τάση του χρόνου εκτέλεσης με την αύξηση των executors. Αυτό οφείλεται στο γεγονός ότι προσθέτουμε δύο νέες στήλες, αυτή την φορά την ημερομηνία και το ποσοστό σε φιλοδώρημα, στο DataFrame. Όπως έχει προαναφερθεί, αυτή η διεργασία επωφελείται από μεγάλη και συνεχόμενη μνήμη. Η αύξηση των executors κατακερματίζει την RAM και την cache καθώς, με αποτέλεσμα την μείωση της επίδοσης.

4.7 Συμπεράσματα

Με βάση τα αποτελέσματα από τις εκτελέσεις των queries συμπεραίνουμε ότι το παρόν configuration συστήματος δεν είναι επαρκές για την πλήρη εκμετάλλευση παραλληλοποίησης. Ο ιδανικός αριθμός πυρήνων για κάθε executor (executor cores) είναι 5 και δεδομένου ότι ο κάθε κόμβος διαθέτει μόνο 4 πυρήνες είναι φανερό ότι το σύστημα μας υπολείπεται. Η διαθέσιμη μνήμη επίσης δεν ευνοεί την παραλληλοποίηση, δεδομένου ότι κάθε κόμβος διαθέτει 8 GB μνήμης RAM η οποία μοιράζεται μεταξύ των executors και του driver του Spark καθώς και του Hadoop.