ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΡΟΧΩΡΗΜΕΝΑ ΘΕΜΑΤΑ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

Ακ.΄Ετος 2019-2020

Εξαμηνιαία Εργασία

Ξενίας Δημήτριος
03115084

1.Περίληψη

Σκοπός της εργασίας είναι η χρήση εργαλείων κατανεμημένων συστημάτων για την επεξεργασία δεδομένων που είναι αποθηκευμένα σε κατανεμημένους κόμβους και έχουν αρκετά μεγάλο μέγεθος (Big Data). Για τον σκοπό χρησιμοποιούνται 2 εικονικά μηχανήματα (Vms) που διατίθενται από τον Okeanos-Knossos και προσομοιώνουν ένα δίκτυο 2 κόμβων που επικοινωνούν μεταξύ τους. Επίσης για την προσομοίωση του κατανεμημένου δικτύου χρησιμοποιείται το αρχείο φακέλων HDFS για την κατανεμημένη αποθήκευση και το εργαλείο Spark για την επεξεργασία των δεδομένων.

Τα δεδομένα που χρησιμοποιούνται είναι πραγματικά και αφορούν σε διαδρομές taxi στην Νέα Υόρκη και περιέχουν 13 εκατομμύρια διαδρομές, που πραγματοποιήθηκαν το Μάρτιο του 2015. Από τα τρία θέματα της εργασίας επιλέχθηκε το 1ο κατά σειρά με τίτλο: "Υλοποίηση SQL ερωτημάτων για αναλυτική επεξεργασία δεδομένων".

Για τον προγραμματισμό με το εργαλείο του Spark, χρησιμοποιείται η Python μέσω του PySpark interface και τρέχονται αρχεία .py μέσω της εντολής *spark-submit test.py.*

2.Υλοποίηση SQL ερωτημάτων για αναλυτική επεξεργασία δεδομένων

Τα δεδομένα αρχικά αποθηκεύονται στο HDFS ως csv αρχεία. Το πρόβλημα που καλούμαστε να αντιμετωπίσουμε είναι ο υπολογισμός των ερωτημάτων του Πίνακα1 με δύο διαφορετικούς τρόπους:

- Γράφοντας MapReduce κώδικα χρησιμοποιώντας το RDD API του Spark
- Χρησιμοποιώντας SparkSQL και το DataFrame API

Πίνακας 1

ID	Query

Q1	Ποια είναι η μέση διάρκεια διαδρομής (σε λεπτά) ανά ώρα έναρξης της διαδρομής; Ταξινομείστε το αποτέλεσμα με βάση την ώρα έναρξης σε αύξουσα σειρά.
	Ενδεικτικά αποτελέσματα:
	
	`HourOfDay AverageTripDuration`
	`00 13.035268962938562`
	`01 15.24545454540119`

Q2	Ποιες είναι οι 5 πιο γρήγορες κούρσες που έγιναν μετά τις 10 Μαρτίου και σε ποιους vendors ανήκουν;

2.1. Φόρτωση αρχείων csv στο HDFS

Η φόρτωση των αρχείων csv στο HDFS γίνεται με πολύ εύκολο τρόπο μέσω της εντολής
user@master:~/data$ hadoop fs -put yellow_tripdata_1m.csv hdfs://master:9000/.
user@master:~/data$ hadoop fs -put yellow_tripvendors_1m.csv hdfs://master:9000/.

Έτσι λοιπόν ο master ως NamedNode δεσμεύει resources στο cluster ώστε να αποθηκεύσει κατανεμημένα τα δεδομένα στο δίκτυο με τους DataNodes (στην περίπτωση μας ένας μόνος DataNode) και κρατάει meta-data για το cluster, καθώς και μια λίστα με τα blocks των αρχείων και το πού αυτά βρίσκονται.

2.2 Υλοποίηση και τρέξιμο των ερωτημάτων του Πίνακα 1

2.2.1 Με χρήση Mapreduce

Στην περίπτωση αυτή δουλεύουμε με το RDD API του Spark και την χρήση λογικής MapReduce παρόμοια με αυτή του Hadoop. Η υλοποίηση τρέχει απευθείας πάνω στα αρχεία κειμένου τα οποία μέσω της εντολής

yellow_tripdata_1m = sc.textFile("hdfs://master:9000/yellow_tripdata_1m.csv")

φορτώνονται ως RDD (Resilient Distributed Dataset). Είναι ουσιαστικά η θεμελιώδης δομή δεδομένων με την οποία λειτουργεί το Spark και είναι το πρώτο πράγμα που πρέπει να γίνει για να μπορούμε να κάνουμε MapReduce πάνω στα δεδομένα. Έχοντας λοιπόν τα τα δεδομένα μας σε RDD χρησιμοποιούμε την map συνάρτηση για να φτιάξουμε τα δεδομένα σε τούπλες (key,value).

Για το πρώτο ερώτημα, αρκεί για κάθε γραμμή δεδομένων να επιστρέφουμε την ακριβή ώρα εκκίνησης της κούρσας και τη χρονική διάρκεια. Στη συνέχεια κάνουμε μια ομαδοποίηση με βάση την κοινή ώρα εκκίνησης και κρατάμε τη μέσο όρο των τιμών τους. Έτσι έχουμε ως έξοδο μια λίστα με (“ώρα έναρξης κούρσας”,”μέση διάρκεια”) που είναι το ζητούμενο της άσκησης και απλά με μια ταξινόμηση ως προς τον χρόνο εκκίνησης, παίρνουμε το ακόλουθο αποτέλεσμα:

```
[(u'00', 14.017793737362242), (u'01', 13.975069898907156), (u'02', 13.035635592676691), (u'03', 13.322282520526898), (u'04', 13.799857931121982), (u'05', 13.2755
83221175415), (u'06', 12.487420237563251), (u'07', 13.395006418527382), (u'08', 14.62750454336785), (u'09', 14.670106419765608), (u'10', 14.65793916969826), (u'1
1', 14.93582122190557), (u'12', 15.130881322885703), (u'13', 15.553918733195145), (u'14', 16.523138380789458), (u'15', 30.223498632126194), (u'16', 17.2130720693
7467), (u'17', 16.51082565440858), (u'18', 15.29045374860121), (u'19', 14.221208805985762), (u'20', 13.575899636364236), (u'21', 13.510855327392878), (u'22', 14.
231797625637382), (u'23', 13.958471125232714)]
```

Να σημειωθεί ότι αυτό το Job χωρίστηκε σε 27 tasks για το tripdata, αφού τόσα είναι και τα partitions του αρχείου rdd που δημιουργήθηκαν από το .

```
[(u'292058461054', (2551.4213051640336, u'2')), (u'352188042892', (460.8606932244732, u'2')), (u'369367777958', (592.3224002302029, u'2')), (u'292057869323', (1275.272126778
2613, u'2')), (u'317828383378', (592.4587726259158, u'2'))]
```

Για το δεύτερο ερώτημα, αρχικά χρειάστηκε να φιλτράρουμε το RDD με τα αρχικά δεδομένα ώστε να κρατήσουμε μόνο αυτά που έχουν ημερομηνία έναρξης κούρσας μετά τις 10 Μαρτίου. Στη συνέχεια, πρέπει να κάνουμε map ώστε να επιτρέψουμε μια τούπλα (key,value) με key το id της κούρσας και value την ταχύτητα. Έτσι εύκολα με μια ταξινόμηση παίρνουμε τους 5 πιο γρήγορους και ύστερα κάνουμε συνένωση με το RDD των yellow_tripvendors_1m ώστε να πάρουμε το τελικό ζητούμενο. Να σημειωθεί ότι έγινε και μια διαλογή δεδομένων ούτως ώστε οι συντεταγμένες που δίνονται να είναι μέσα στη Νέα Υόρκη. Το αποτέλεσμα είναι το εξής.

Βλέπουμε λοιπόν το αποτέλεσμα της συνένωσης των δυο RDDs. Η χρήσιμη πληροφορία είναι η δεύτερη τούπλα μέσα στις 5 τούπλες της λίστας. Εκεί φαίνεται το αποτέλεσμα με την μορφή (“ταχύτητα”,”vendor”) για τις 5 πιο γρήγορες κούρσες

2.2.1 Με χρήση SparkSQL σε DataFrames

Εδώ χρησιμοποιούμε μια άλλη μορφή απεικόνισης των δεδομένων, τα Dataframes. Είναι ο τύπος δεδομένων με τα οποία το framework SparkSQL μπορεί να εκτελέσει τα SQL ερωτήματα, είτε με κανονική SQL γλώσσα, είτε μέσω DataFrame Operations. Τα αρχεία φορτώνονται τώρα μέσω ενός SQLContext και από csv γίνονται Dataframes μέσω της εντολής:

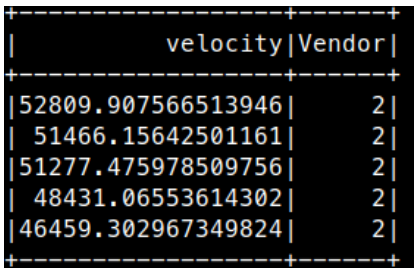
df_tripdata = sqlContext.read.csv("hdfs://master:9000/yellow_tripdata_1m.csv")

Στην συνέχεια με την χρήση DataFrame Operations που υπάρχουν στο documentation της Spark καταλήγουμε στα εξής αποτελέσματα:

Query 1:

```
+-----+------------------+
|start|     avg(duration)|
+-----+------------------+
|   00|14.014732218495665|
|   01|13.971026902993223|
|   02|13.031247770806884|
|   03|12.381880976344284|
|   04|13.799096847846748|
|   05|13.274742036787808|
|   06|12.487023011525073|
|   07| 13.39500641852737|
|   08|14.626919176562351|
|   09|14.668773991637426|
|   10|14.657253383754934|
|   11| 14.93487388589339|
|   12|  15.1296123774549|
|   13|15.552627873251884|
|   14|  16.5220609090768|
|   15|30.211548179083874|
|   16|17.211297781030606|
|   17|16.509717117907787|
|   18|15.289086069641007|
|   19|14.219330238513898|
|   20|13.574330702255747|
|   21|13.509544933658109|
|   22| 14.23033952575753|
|   23|13.956139401824828|
+-----+------------------+
```

Query 2:



velocity	Vendor
52809.907566513946	2
51466.15642501161	2
51277.475978509756	2
48431.06553614302	2
46459.302967349824	2

ενώ χωρίστηκαν σε 14 tasks αυτή τη φορά για το tripdata καθώς τα Dataframes λαμβάνουν λιγότερο χώρο στην μνήμη και έτσι χρειάζονται λιγότερα tasks.

2.3 Μετατρέψτε τα αρχεία κειμένου σε αρχεία Parquet. Στη συνέχεια φορτώστε τα Parquet αρχεία ως Dataframes και εκτελέστε το υποερώτημα 2b. Πόσος χρόνος χρειάζεται για τη μετατροπή των αρχείων;

Αντί να τρέξουμε τώρα τα ερωτήματά μας απευθείας πάνω στα csv αρχεία, μπορούμε να μετατρέψουμε πρώτα το dataset σε μια ειδική μορφή που:

- Έχει μικρότερο αποτύπωμα στη μνήμη και στον δίσκο και άρα βελτιστοποιεί το I/O (input/output) μειώνοντας τον χρόνο εκτέλεσης.
- Διατηρεί επιπλέον πληροφορία, όπως στατιστικά πάνω στο dataset, τα οποία βοηθούν στην πιο αποτελεσματική επεξεργασία του. Για παράδειγμα, αν ψάχνω σε ένα σύνολο δεδομένων τις τιμές που είναι μεγαλύτερες από 100 και σε κάθε block του dataset έχω πληροφορία για το ποια είναι η min και ποια η max τιμή, τότε μπορώ να παρακάμψω την επεξεργασία των blocks με max τιμή < 100 γλιτώνοντας έτσι χρόνο επεξεργασίας.

Το ειδικό format που χρησιμοποιούμε για να επιτύχουμε τα παραπάνω είναι το Apache Parquet. Όταν φορτώνουμε έναν πίνακα σε Parquet, αυτός μετατρέπεται κι αποθηκεύεται σε ένα columnar format που βελτιστοποιεί το I/O και τη χρήση της μνήμης κι έχει τα χαρακτηριστικά που αναφέραμε.

Για το γράψιμο ενός αρχείου parquet χρησιμοποιείται η εντολή

df_tripdata.write.parquet("hdfs://master:9000/yellow_tripdata_1m.parquet")
df_tripdata.write.parquet("hdfs://master:9000/yellow_tripvendors_1m.parquet")

όπου παίρνει ένα rdd και το γράφει σε parquet.
Ο χρόνος που χρειάζεται για τις εντολές αυτές είναι:

```
---yellow_tripdata_1m.parquet write in 61.8045909405 seconds ---
```

```
---yellow_tripvendors_1m.parquet write in 14.7228150368 seconds ---
```

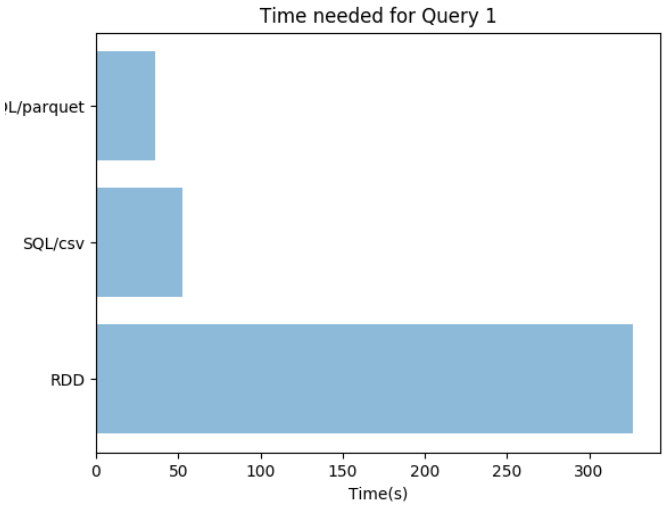
Τα αποτελέσματα των queries όπως είναι φυσικό είναι ίδια με το προηγουμένως, αλλά αυτή τη φορά χωρίζονται σε 4 tasks για το tripdata, αφού το parquet έχει μικρότερο αποτύπωμα στη μνήμη.

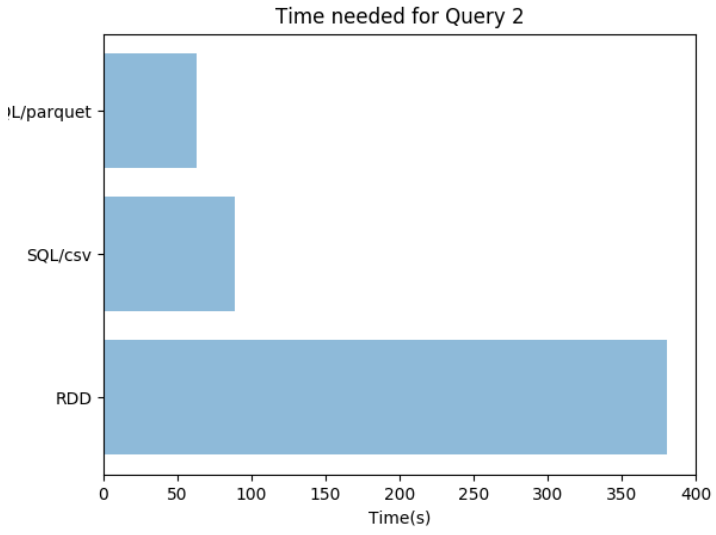
2.4 Για κάθε ερώτημα του Πίνακα 1, συγκρίνετε και φτιάξτε ένα διάγραμμα με τον χρόνο εκτέλεσης των παραπάνω διαφορετικών περιπτώσεων

Στην συνέχεια βλέπουμε το διάγραμμα με το χρόνο εκτέλεσης των περιπτώσεων:

a. SQL πάνω σε Parquet αρχεία. (Διευκρίνιση: Τα SQL ερωτήματα θα πρέπει να τρέξουν σε Dataframes που έχουν δημιουργηθεί είτε από αρχεία κειμένου είτε από αρχεία Parquet)
b. SQL πάνω σε csv αρχεία.
c. MR me RDD πάνω σε csv αρχεία.

για το Query 1 και Query 2 αντίστοιχα.





Παρατηρούμε ότι υπάρχει τρομερά μεγάλη και αισθητή διαφορά όταν δουλεύουμε με RDD και MapReduce, εν αντιθέσει με όταν δουλεύουμε με SQLSpark σε DataFrames. Αυτό οφείλεται στον διαφορετικό τρόπο που τα δεδομένα αποθηκεύονται και διαχειρίζονται από την Spark. Αντίθετα, μεταξύ parquet και csv σε Dataframes βλέπουμε μια πιο μικρή διαφορά και αυτή λόγω

των χαρακτηριστικών των parquet που αναφέρθηκαν παραπάνω. Έτσι λοιπόν βλέπουμε πόσο πιο γρήγορο είναι να δουλεύουμε με SQL Spark, η οποία μάλιστα είναι και πιο απλούστερη στο γράψιμο. Ουσιαστικά η όλη διαφορά είναι στα Tasks που δημιουργούνται από τον Task Manager. Όπως αναφέραμε, στο RDD MR δημιουργούνται 27 tasks, ενώ σε Dataframe από csv 14 tasks και σε parquet μόλις 4, καθώς σε κάθε μια περίπτωση τα δεδομένα αποθηκεύονται με διαφορετικό τρόπο.