

**ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ**

**Δ.Π.Μ.Σ ΕΠΙΣΤΗΜΗ ΔΕΔΟΜΕΝΩΝ & ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ**

**Ακαδημαϊκό Έτος 2018-19**

**Εξαμηνιαία εργασία στη Διαχείριση Δεδομένων Μεγάλης Κλίμακας**

**Μανδηλαράς Νικηφόρος A.M.: 03400022**

**Παραδοτέα 22/9/19**

# 1 Εισαγωγή

Η ραγδαία αύξηση των διαθέσιμων δεδομένων τα τελευταία χρόνια οδήγησε στη δημιουργία νέων αρχιτεκτονικών προκειμένου να καταστεί εφικτή η αποτελεσματική αποθήκευση και διαχείριση αυτών. Η όλη αυτή προσπάθεια βασίστηκε σε κατανεμημένα συστήματα τα οποία έχουν τη δυνατότητα αποτελεσματικής κλιμάκωσης, μέσω της προσθήκης επιπλέον κόμβων (οριζόντια κλιμάκωση). Έτσι δημιουργήθηκαν μια σειρά από frameworks προκειμένου να διευκολύνουν τον προγραμματισμό σε κατανεμημένο περιβάλλον, μεταξύ των οποίων και το Apache Spark το οποίο και χρησιμοποιούμε στην παρούσα εργασία. Το Apache Spark προσφέρει βελτιστοποιήσεις σε σχέση με το προγενέστερο MapReduce με κυρίαρχη τη διατήρηση ενδιάμεσων αποτελεσμάτων στην κύρια μνήμη αποφεύγοντας τις εγγραφές στο δίσκο.

Σκοπός της συγκεκριμένης άσκησης είναι η εξοικείωση με τις προγραμματιστικές διεπαφές του Spark (RDDs, Dataframes) καθώς και με τον τρόπο λειτουργίας του ίδιου του framework (υλοποιήσεις join). Να αναφέρουμε ότι για την υλοποίησή μας χρησιμοποιήσαμε το Java API του Spark.

# 2 Οργάνωση και προετοιμασία

### 2.1 Οργάνωση Κώδικα

Με βάση την εκφώνηση της άσκησης επιλέχθηκε να υλοποιηθούν τρεις κύριες κλάσεις μία για κάθε ζητούμενο. Ακολουθεί μία συνοπτική περιγραφή όλων αυτών:

|  |  |
| --- | --- |
| SparkTaxiMapReduce | Στην κλάση αυτή περιέχεται η υλοποίηση των ερωτημάτων με χρήση του RDD API. |
| SparkTaxiSQL | Αντίστοιχα με την παραπάνω αλλά για το SQL API. |
| SparkTaxiJoin | Εδώ βρίσκεται η υλοποίηση του 2ου θέματος σχετικά με την απόδοση των διαφορετικών Join αλγορίθμων. |
| SparkTaxiCreateParquet | Υπεύθυνη για τη δημιουργία των parquet files. |
| SparkBS | Factory class για τη δημιουργία και παραμετροποίηση του Entry Point στο Spark ανάλογα με το ερώτημα που εκτελείται. |
| Functions | Περιέχονται διάφορες συναρτήσεις που χρησιμοποιούνται απ’ όλες τις υλοποιήσεις π.χ. haversine. |
| TaxiSchema | Περιέχει τα ονόματα των στηλών που είτε υπήρχαν είτε δημιουργήθηκαν. |
| Settings | Η κλάση αυτή αναλαμβάνει τη διαχείριση των command line παραμέτρων με τη βοήθεια του framework JCommander. |
| Constants | Εδώ εμπεριέχονται διάφορες σταθερές, όπως η ακτίνα της Γης. |

### 2.2 Υποδομή

Για τις ανάγκες της άσκησης δημιουργήθηκε στον Ωκεανό ένα Hadoop-Spark cluster δύο κόμβων σύμφωνα με τις οδηγίες που δόθηκαν. Κατόπιν τα δεδομένα καθώς και τα jar αρχεία κώδικα μεταφέρθηκαν εκεί με χρήση του FileZilla. Η εκτέλεση και όλες οι μετρήσεις που παρουσιάζονται παρακάτω έλαβαν χώρα στο συγκεκριμένο cluster.

### 2.3 Εκκαθάριση Δεδομένων

Τα δεδομένα μας αποτελούνταν από 13 εκατομμύρια εγγραφές που αντιστοιχούσαν σε διαδρομές ταξί στην περιοχή της Νέας Υόρκης. Παρατηρήθηκε ότι στα δεδομένα εμπεριέχονταν προβληματικές τιμές και έγιναν ενέργειες για την εκκαθάρισή τους. Συγκεκριμένα απομακρύνθηκαν εγγραφές που είχαν ώρα περάτωσης μικρότερη, ίση από ώρα έναρξης καθώς και εγγραφές με συντεταγμένες εκτός της γεωγραφικής περιοχής της πολιτείας της Νέας Υόρκης ή με ίδιες συντεταγμένες έναρξης και λήξης διαδρομής.

Παρ’ όλα αυτά κάποιες εγγραφές παρέμειναν προβληματικές εμφανίζοντας εξωπραγματικές τιμές στο 2ο και 3ο SQL ερώτημα, όπου στο μεν πρώτο τα κόστη των διαδρομών είναι τεράστια ενώ στο δεύτερο φαίνεται να διανύονται τεράστιες αποστάσεις σε πολύ μικρούς χρόνους με αποτέλεσμα να επιτυγχάνονται διαστημικές ταχύτητες στις κούρσες. Επιλέχθηκε να μην προστεθούν επιπλέον πιο αυστηροί κανόνες ώστε να αποφύγουμε να αποκοπούν και υγιείς μετρήσεις.

# 3 Θέμα 1ο

### 3.1 HDFS

Πρώτο βήμα αποτελεί η τοποθέτηση των αρχείων στο hdfs του cluster από το filesystem του edge node όπου είχαν μεταφερθεί. Στο hdfs δεν υπάρχει η έννοια του current directory, συνεπώς για την τοποθέτηση των αρχείων πρώτα χρειάστηκε να δημιουργήσουμε ένα. Οι εντολές που χρησιμοποιήσαμε ήταν η εξής:

|  |
| --- |
| hdfs dfs –mkdir /user  hdfs dfs –put <filename> /user |

### 3.2 Υλοποίηση ερωτημάτων

### 3.2.1 MapReduce

Αφού διαβάσουμε τα αρχεία κειμένου προβαίνουμε στην εκκαθάριση των εγγραφών τους με τον τρόπο που περιγράψαμε προηγουμένως. Παρουσιάζουμε στη συνέχεια τις ενέργειες που χρειάστηκε να κάνουμε για κάθε ερώτημα.

* Στο πρώτο query η εγγραφή διασπάται σε επιμέρους πεδία και από το πεδίο έναρξης της διαδρομής εξάγεται η ώρα, η οποία αποτελεί και το κλειδί του νέου PairRDD που προκύπτει ως αποτέλεσμα της πρώτης αυτής Map φάσης. Το αντίστοιχο value είναι η διάρκεια της διαδρομής όπως υπολογίστηκε από την συνάρτηση timeDiff. Στη συνέχεια οι εγγραφές με ίδιο κλειδί συγκεντρώνονται σε groups με χρήση της groupByKey και για κάθε group υπολογίζεται η μέση διάρκεια. Το τελικό αποτέλεσμα παρουσιάζεται αφού έχει γίνει και χρήση της sortByKey ώστε να προβληθούν οι ώρες σε μια λογική σειρά.
* Για το δεύτερο ερώτημα πραγματοποιούμε παρόμοια διαδικασία με πριν ώστε να δημιουργήσουμε ένα RDD με key το id της εγγραφής προκειμένου στη συνέχεια να πραγματοποιήσουμε join με το δεύτερο αρχείο στο οποίο επίσης έχουμε εφαρμόσει την ίδια διαδικασία. Από εκεί και πέρα ξεκινά μια νέα φάση Map Reduce όπου αρχικά για κάθε εγγραφή κρατάμε ως key τον vendor και ως value το κόστος αυτής. Στο reduceByKey κρατάμε απλά την μέγιστη τιμή.
* Για το τρίτο query ξεκινάμε από το ενοποιημένο RDD που δημιουργήσαμε προηγουμένως, διασπάμε τα διάφορα πεδία και τα διατηρούμε σε λίστα. Φιλτράρουμε το RDD ώστε να κρατήσουμε τα δεδομένα που έπονται της 10ης Μαρτίου. Τέλος έχουμε εκ νέου μια Map φάση όπου υπολογίζεται η διάρκεια, η απόσταση με χρήση της συνάρτησης haversine και από αυτά τα επιμέρους μεγέθη, εν τέλει και η ταχύτητα σε χιλιόμετρα ανά ώρα. Ως αποτέλεσμα παρουσιάζονται οι 5 πιο γρήγορες διαδρομές με τη βοήθεια της συνάρτησης takeOrdered.

Ο Ψευτοκώδικας Map Reduce για τα τρία ερωτήματα που αναλύθηκαν συμπεριλαμβάνεται στα παραδοτέα αρχεία της εργασίας.

### 3.2.2 SparkSQL

Πρώτα απ’ όλα διαβάζουμε σε dataframe τα δύο αρχεία με βάση το schema που έχουμε προσδιορίσει. Ακολουθεί η εκκαθάριση των δεδομένων καθώς και η συνένωση των dataframes που αντιστοιχούν στα δύο αρχεία.

* Για το πρώτο query δημιουργούμε μία νέα στήλη με την ώρα έναρξης της κάθε διαδρομής, στη συνέχεια πραγματοποιούμε groupBy με βάση αυτή και παίρνουμε τη μέση διάρκεια των διαδρομών για κάθε ώρα έναρξης. Τέλος ταξινομούμε το αποτέλεσμα με βάση ώρα έναρξης.
* Στο δεύτερο ερώτημα πραγματοποιούμε αυτή τη φορά groupBy με το id του vendor και για κάθε τέτοιο id κρατάμε απλά την εγγραφή με το μέγιστο κόστος.
* Για το τρίτο query αρχικά φιλτράρουμε όλες τις εγγραφές κρατώντας αυτές που έλαβαν χώρα μετά τις 10 Μαρτίου. Στη συνέχεια δημιουργούμε δύο νέες στήλες μία όπου υπολογίζουμε την απόσταση από τις συντεταγμένες με βάση τη συνάρτηση haversine και μία δεύτερη όπου από την απόσταση και την διάρκεια του ταξιδιού υπολογίζουμε την μέση ταχύτητα.

### 3.3 Parquet

Για την μετατροπή σε parquet αρχεία εκτελούμε την main της κλάσης SparkTaxiCreateParquet. Για την εκτέλεση με parquet προσθέτουμε την παράμετρο –useParque, η οποία έχει ως αποτέλεσμα να διαβαστούν τα parquet ως dataset και από εκεί και πέρα η εκτέλεση ακολουθεί την πορεία που αναφέραμε προηγουμένως.

### 3.4 Μετρήσεις

Ακολουθούν οι χρόνοι εκτέλεσης σε δευτερόλεπτα για όλα τα queries και για όλες τις υλοποιήσεις. Στο σημείο αυτό να αναφέρουμε πως οι υλοποιήσεις έγιναν θεωρώντας τη διαδικασία υποβολής ερωτημάτων ως ολότητα. Δηλαδή για παράδειγμα σκοπός ήταν το join operation να εκτελεστεί μία φορά με όλα τα απαραίτητα πεδία ώστε το αποτέλεσμα αυτού να μπορεί να χρησιμοποιηθεί και από τα δύο ερωτήματα που το χρειάζονται. Το ενδιάμεσο αυτό αποτέλεσμα γινόταν cache. Εν τέλει όμως παρατηρήσαμε πως η διαδικασία του explicit cache ήταν χρονικά πιο κοστοβόρα.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| API/Queries | 1st Query | 2nd Query | 3rd Query | Total Time |
| Map Reduce | 41,37 | 66,7 | 82,52 | 192 |
| SQL | 61,18 | 75,2 | 65,45 | 216 |
| SQL Parquet | 7,51 | 18,72 | 14,69 | 58 |

Γράφημα 1: Χρόνοι εκτέλεσης για κάθε query και ανά υλοποίηση

Γράφημα 2: Συνολικός χρόνος εκτέλεσης και των 3 ερωτημάτων για κάθε υλοποίηση

Τέλος, όσο αφορά τους χρόνους δημιουργίας των parquet files αυτοί ήταν 85,04 s για το μεγάλο fact αρχείο και 8,5 s για το lookup όπου περιέχονταν τα ids των πωλητών.

### 3.5 Σχολιασμός

Από τα παραπάνω γραφήματα ξεχωρίζει η επίδραση στην απόδοση των parquet αρχείων καθώς ο χρόνος εκτέλεσης είναι υποτετραπλάσιος σε σχέση με την αντίστοιχη εκτέλεση σε txt. Μάλιστα ο χρόνος εκτέλεσης παραμένει μικρότερος ακόμα αν συμπεριλάβουμε σε αυτόν και το κόστος της μετατροπής σε parquet. Γίνεται ξεκάθαρο λοιπόν, πως ακόμα και για την πιο απλή διαδικασία είναι προς όφελος μας να μετατρέψουμε τα δεδομένα μας σε parquet.

Απ’ την άλλη οι χρόνοι εκτέλεσης μεταξύ MapReduce και SQL API είναι παρεμφερείς, με το MapReduce να έχει ένα μικρό προβάδισμα. Αυτό μπορεί να εξηγείται από το γεγονός πως το Spark SQL είναι API υψηλότερου επιπέδου, αν και η χρήση του συνίσταται αφού παρέχει βελτιστοποιήσεις στον τρόπο εκτέλεσης των stages που θέτει ο χρήστης. Σε κάθε περίπτωση η χρήση του SparkSQL είναι ανώτερη τόσο από άποψη ευκολίας στον προγραμματισμό, όσο όμως και από άποψη απόδοσης όταν συνδυάζεται με parquet αρχεία.

# 4 Θέμα 2ο

### 4.1 Broadcast Join

Χρησιμοποιώντας το explain plan όπως αναφέρεται στην υπόδειξη βλέπουμε πως το είδος του Join που επιλέχθηκε από τον optimizer ήταν Broadcast Join. Αυτό το είδος join επιλέγεται όταν συμμετέχει ένας μεγάλος πίνακας που περιέχει την κύρια πληροφορία και ένας μικρότερος lookup πίνακας. Η ανάγκη για τέτοια joins είναι πολύ συχνή σε βάσεις με σχεσιακά σχήματα καθώς τα δεδομένα βρίσκονται σε κανονικοποιημένη μορφή προκειμένου να αποφεύγεται η αναπαραγωγή πληροφορίας.

Κατά το Broadcast Join ο μικρότερος lookup πίνακας μεταφέρεται σε όλους τους κόμβους και στη συνέχεια πραγματοποιείται join μεταξύ του τμήματος του fact table που διαθέτει ο κάθε κόμβος με το αντίγραφο του lookup που έλαβε. Το join αυτό προτιμάται ώστε να αποφευχθεί ένα shuffle operation στο μεγάλο πίνακα το οποίο θα δημιουργούσε μεγάλη επιβάρυνση στους χρόνους εκτέλεσης. Το Spark στις default του ρυθμίσεις θέτει τα 10 ΜΒ ως το μέγιστο μέγεθος του μικρότερου πίνακα για το οποίο μπορεί να επιλεγεί το Broadcast Join.

### 4.2 SortMergeJoin

Για να αποτρέψουμε το βελτιστοποιητή να επιλέξει το συγκεκριμένο είδος join θέτουμε το configuration «spark.sql.autoBroadcastJoinThreshold» ίσο με -1 κατά τη δημιουργία του SparkSession, κάτι που απενεργοποιεί εντελώς τη δυνατότητα για Broadcast joins. Συμβουλευόμενοι και πάλι το explain plan βλέπουμε πως πλέον επιλέγεται η υλοποίηση SortMergeJoin.

Το SortMergeJoin πρώτα ταξινομεί τις εγγραφές στα partitions των δύο πινάκων ώστε να διευκολυνθεί μετέπειτα κατά την αναζήτηση των αντίστοιχων εγγραφών. Το join αυτό επιλέγεται καθώς το είδος του κλειδιού στην δικιά μας περίπτωση είναι τέτοιο που επιτρέπει την ταξινόμηση. Μάλιστα παρατηρούμε ότι τα δεδομένα στη μορφή που μας δίνονται και στα δύο αρχεία είναι ήδη ταξινομημένα με βάση το id κάτι που σημαίνει ότι το κομμάτι του sort που είναι και ακριβό υπολογιστικά θα μπορούσε να αποφευχθεί. Προϋπόθεση για την αποδοτική λειτουργία του Join αυτού είναι τα partitions να έχουν κοινά κλειδιά, ώστε να αποφευχθούν shuffle operations.

Από την έκδοση 2.3 του Spark το SortMergeJoin προτιμάται έναντι του ShuffleHashJoin και ο λόγος για αυτό είναι ότι το δεύτερο απαιτεί το hash table που δημιουργείται να χωρά στη μνήμη τη στιγμή που το πρώτο μπορεί να επεκταθεί και στο δίσκο. Στη συνέχεια ακολουθεί η σύγκριση των δύο εκτελέσεων.

### 4.3 Χρόνοι Εκτέλεσης

Εν τέλει για την πρώτη υλοποίηση με το BroadcastJoin ο χρόνος που απαιτήθηκε για τη συνένωση ήταν 18 δεύτερα ενώ για την δεύτερη 24. Όπως αναμέναμε ο χρόνος με χρήση broadcast join είναι μικρότερος. Δεδομένου ότι δεν έχουμε λάβει κάποια μέριμνα για το πώς τα αρχεία χωρίζονται σε partitions κατά την τοποθέτησή τους στο HDFS δεν είναι δυνατόν να αποφευχθεί στην περίπτωση του SortMergeJoin ένα shuffle operation κάτι που αυξάνει τους χρόνους. Το συγκεκριμένο αποτέλεσμα καταδεικνύει και την χρησιμότητα του query optimizer που διαθέτει το Apache Spark, ο οποίος του επιτρέπει να ανταποκρίνεται με τον καλύτερο δυνατό τρόπο στις διάφορες συνθήκες και δεδομένα που μπορεί να του παρουσιαστούν.

# 5 Προβλήματα

Κατά τη διάρκεια των δοκιμών στον Ωκεανό αντιμετωπίσαμε προβλήματα που σχετίζονταν με την περιορισμένη διαθέσιμη μνήμη των μηχανημάτων. Συγκεκριμένα σποραδικά οι executors ξέφευγαν από συγκεκριμένα όρια μνήμης που θέτει το Spark κάτι που είχε ως αποτέλεσμα τον τερματισμό τους από τον resource manager και την εκκίνηση νέου executor. Ως εκ τούτου υπήρχε επιβάρυνση στους χρόνους εκτέλεσης. Αυτή η συμπεριφορά εμφανίστηκε στην υλοποίηση με το Map Reduce API και στους executors που έτρεχαν στο master node καθώς αυτός ο κόμβος ήταν επιβαρυμένος και με τον drive.

Επιπλέον, η έλλειψη μνήμης είχε ως αποτέλεσμα την αδυναμία λειτουργίας του History Server καθώς η χρήση του επιδείνωνε το πρόβλημα απώλειας executors στον Master Node. Αυτή η εξέλιξη δυσκόλεψε πολύ το monitoring των εφαρμογών και την καταγραφή των μετρήσεων οι οποίες εξήχθησαν εν τέλει από τα logs.