Δημήτρης Κελέσης A.M.: 03115037

Σύντομη εισαγωγή

Στην εργασία αυτή θα χρησιμοποιήσουμε αλγόριθμους και τεχνικές MapReduce ώστε να επεξεργαστούμε μαζικά μεγάλες ποσότητες δεδομένων. Για να το πετύχουμε αυτό έχουμε στην διάθεση μας πόρους στον Ωκεανό, ειδικότερα διαθέτουμε 2 VMs και καθένα από τα οποία έχει 2 πυρήνες, 2GB RAM και 30GB hard drive (μνήμη), Για να αξιοποιηθούν κατάλληλα οι πόροι αυτοί χρειάστηκε να τους συνδέσουμε σε ένα ιδιωτικό δίκτυο και σε αυτό να εγκαταστήσουμε το Hadoop και το Spark. Πιο συγκεκριμένα θα γίνει χρήση του hdfs για την αποθήκευση του μεγάλου όγκου δεδομένων κατανεμημένα ώστε να κάνουμε πιο εύκολους και ταχύτερους τους υπολογισμούς φέρνοντας τους κοντά στον εκάστοτε Worker που θα αναλάβει να τους εκτελέσει. Ταυτόχρονα χρησιμοποιούμε το Spark για να γράψουμε κώδικα για εργασίες MapReduce που θα εκτελεστούν στα δεδομένα εισόδου και στα αποτελέσματα αυτών. Η γλώσσα προγραμματισμού που επελέγη για την συγκεκριμένη εργασία είναι η Python και ειδικότερα το module Pyspark που αυτή διαθέτει. Δεν έγινε χρήση της Scala πάνω στην οποία έχει γραφτεί το Spark γιατί η εξοικείωση με τη γλώσσα αυτή είναι περιορισμένη και οι πράξεις πινάκων ως ενέργειες μπορούν να γίνουν αποδοτικότερα με χρήση Python λόγω της πληθώρας των βιβλιοθηκών της. Στην εκπόνηση της εργασίας παρουσιάστηκαν προβλήματα σε ότι αφορά πιθανώς κακόβουλο λογισμικό καθώς η εργασία είχε εκπονηθεί σε cluster στην διάρκεια των Χριστουγέννων και ότι επιχειρήθηκε η ανάκτηση των αποτελεσμάτων αυτής τον Φεβρουάριο παρατηρήσαμε ότι το σύστημα δεν δούλευε σωστά και συγκεκριμένα δεν μπορούσε να τρέξει κανενός είδους πρόγραμμα αλλά ούτε και να μας δώσει τα ζητούμενα αποτελέσματα που είχε ήδη αποθηκεύσει. Συνέπεια αυτού ήταν η καταστροφή των εικονικών μηχανών και η δημιουργία τους ξανά ώστε να μπορέσουμε να τρέξουμε τον κώδικα μας. Τέλος να σημειωθεί ότι σε αυτό βοήθησε το γεγονός ότι ο κώδικας τρέχει σε 8 λεπτά μόνο και άρα ήταν δυνατή η επανεκτέλεση του άμεσα.

Θέμα **3**ο**:** Ομαδοποίηση με χρήση **k-means**

Επιλέξαμε το θέμα σχετικά με την πραγματοποίηση αλγορίθμου μηχανικής μάθησης για ομαδοποίηση δεδομένων. Για τον αλγόριθμο k-means η υλοποίηση που ακολουθήθηκε ήταν σύμφωνη με την προτεινομένη από την εκφώνηση. Εν τάχει η διαδικασία ήταν η εξής: Ανέβασμα των αρχείων στο hdfs, Διάβασμα αυτών από το πρόγραμμα, Απόρριψη μηδενικών συντεταγμένων, Εκτέλεση k-means, Εγγραφή αποτελέσματος σε τοπικό αρχείο, Τοποθέτηση αρχείου στο hdfs, Ανάκτηση αρχείου.

Όπως αναφέρθηκε παρατηρήσαμε ότι τα δεδομένα περιέχουν άκυρες εγγραφές καθώς αρκετές από αυτές έχουν μηδενικά ως συντεταγμένες καθώς όπως αναφέρεται αυτό οφείλεται σε λάθος του χειριστή του συστήματος ή του οδηγού ταξί που δεν είχε ενεργοποιήσει το gps. Οι εγγραφές αυτές είναι άσχετες και άκυρες σε σχέση με το σύστημα μας και το μόνο που

μπορούν να κάνουν είναι να οδηγήσουν σε εσφαλμένα κέντρα και λάθος αποτελέσματα. Για να το εξηγήσουμε αυτό ας φανταστούμε ότι υπάρχει μία μεγάλη μερίδα από μηδενικές εγγραφές, τότε το σύστημα θα πρέπει να υπολογίσει ένα από τα κέντρα να βρίσκεται πολύ κοντά στο (0, 0) τοποθεσία που είναι άκυρη καθώς αντιστοιχεί σε σημείο νοτιοανατολικά της Αφρικής (Null Island) και όχι στην Νέα Υόρκη. Από την άλλη πλευρά αν δεν είναι τόσες πολλές αυτές οι εγγραφές επηρεάζουν αρνητικά το αποτέλεσμα στην ανανέωση των κέντρων καθώς όταν διαιρούμε με το πλήθος των σημείων που έχουν ανατεθεί σε κάθε κέντρο θα αναγκαστούμε να διαιρέσουμε με περισσότερα σημεία χωρίς η ποσότητα του αριθμητή να αυξάνεται αναλόγως, λόγω των μηδενικών.

Διαβάσαμε για αρχή τα αρχεία από το hdfs με χρήση των κατάλληλων συναρτήσεων map, filter ώστε για κάθε γραμμή να απομονώσουμε τα στοιχεία της που αντιστοιχούν στις γεωγραφικές συντεταγμένες και να τα μετατρέψουμε σε μορφή επιθυμητή για την συνέχεια του προγράμματος (floats). Η filter χρησιμοποιήθηκε για να ελέγξουμε αν το γινόμενο των παραπάνω συντεταγμένων ήταν διάφορο του μηδενός για τους λόγους που εξηγήσαμε. Στη συνέχεια κάναμε DataFrame τα δεδομένα μας και ξεκινήσαμε να υλοποιούμε τον αλγόριθμο. Να σημειωθεί ότι για ευκολότερο χειρισμό χρησιμοποιήσαμε ένα λεξικό που αντιστοιχούσε όλα τα κέντρα στα αντίστοιχα labels τους, δηλαδή 0, 1,..,4. Δημιουργήθηκε συνάρτηση που επιστρέφει τα 5 πρώτα στοιχεία από το DataFrame στην είσοδο της ώστε να μπορέσουμε να εξάγουμε τις τιμές αρχικοποίησης του αλγορίθμου όπως αναφέρεται στην εκφώνηση. Στη συνέχεια υλοποιήθηκε η συνάρτηση εύρεσης απόστασης πάνω σε σφαίρα που προσεγγίζει ικανοποιητικά την πραγματική απόσταση δύο σημείων πάνω στη Γη, θα μπορούσαμε να χρησιμοποιήσουμε άμεσα Ευκλείδεια απόσταση καθώς τα σημεία βρίσκονται πολύ κοντά εφόσον μιλάμε για τοποθεσίες στην ίδια πόλη και άρα η καμπυλότητα του χώρου έχει αμελητέα επίδραση. Η παραπάνω συνάρτηση Haversine Distance υλοποιήθηκε όπως αναγράφεται σε σχετική βιβλιογραφία με τις απαραίτητες μετατροπές των δεδομένων εισόδου σε rads και εν συνεχεία τον υπολογισμό των ζητούμενων ενδιάμεσων και τελικών τιμών που υποδηλώνουν την απόσταση.

Για την ανάθεση ετικετών σε κάθε σημείο που υποδηλώνουν σε ποιο κέντρο ανήκει το συγκεκριμένο σημείο αρχικά επιχειρήσαμε να το κάνουμε με εμφωλευμένο map δηλαδή στο map που υπολογίζει για κάθε σημείο όλες τις αποστάσεις από τα 5 κέντρα και αναθέτει ως ετικέτα το κέντρο με την μικρότερη απόσταση να κάναμε κατανεμημένα αυτών τον υπολογισμό των 5 αποστάσεων αλλά κάτι τέτοιο δεν κατέστη δυνατό τεχνικά από την pyspark. Συνεπώς για κάθε σημείο υπολογίζουμε την απόσταση του από τα 5 κέντρα και αναθέτουμε ως ετικέτα αυτού την ετικέτα του κέντρου που βρίσκεται πιο κοντά του.

Τέλος το αποτέλεσμα γράφεται σε μορφή string σε κατάλληλη μεταβλητή που αρχικοποιείται κενή στην αρχή. Κάθε φορά που θέλουμε να γράψουμε κάτι καλούμε αντίστοιχη συνάρτηση που δημιουργήσαμε ώστε οι εγγραφές να είναι ομοιόμορφες και να έχουν κατάλληλη στοίχιση.

Σε ότι αφορά την υλοποίηση αφού διαβάσαμε και βρήκαμε τα 5 αρχικά κέντρα εισερχόμαστε σε μία επαναληπτική διαδικασία για τις 3 ζητούμενες επαναλήψεις. Για κάθε σημείο με χρήση MapReduce υπολογίζουμε κατανεμημένα την ετικέτα του και στη συνέχεια ομαδοποιούμε με βάση αυτή. Τέλος αθροίζουμε όλες τις συντεταγμένες και βρίσκουμε τον μέσο όρο τους που αποτελεί τα νέα κέντρα για την επόμενη επανάληψη, για κάθε ετικέτα. Όλα τα παραπάνω τα

γράφουμε στην μεταβλητή αποτελέσματος ως strings. Τέλος η μεταβλητή αυτή γράφεται σε αρχείο εξόδου το οποίο τοποθετείται στο hdfs και στη συνέχεια το ανακτάμε για να μπορέσουμε να το παρουσιάσουμε.

Ψευδοκώδικας

Παρακάτω θα παραθέσουμε τον ψευδοκώδικα για τα προγράμματα MapReduce που χρησιμοποιήσαμε με τα αντίστοιχα keys, values που το κάθε ένα από αυτά θα βγάζει.

Ο ακόλουθος κώδικας πραγματοποιεί την ανάγνωση ανά γραμμή, την χωρίζει και εφαρμόζοντας την συνθήκη για x, y διάφορα του μηδενός παράγει τα αποδεκτά σημεία. Key κάποιο id και value η γραμμή εισόδου

```
map(key, value):
    line = value.split(',')
    x = float(line[3])
    y = float(line[4])
    if(x * y != 0):
        emit(null, (x ,y))
```

Ο ακόλουθος κώδικας παίρνει σαν είσοδο κάποιο σημείο και εκτελεί πάνω του την συνάρτηση για να βρει την ετικέτα του (για να το κάνει αυτό υπολογίζει την ελάχιστη απόσταση από τα ήδη υπάρχοντα κέντρα και για την ελάχιστη απόσταση αναθέτει την σχετική ετικέτα). Στην έξοδο του δίνει ως κλειδί την ετικέτα και ως τιμή το σημείο ακολουθούμενο από μία μονάδα ώστε να βοηθήσει στην εύρεση του πλήθους των σημείων στην φάση Reduce. Στη συνέχεια στη φάση Reduce αφού έχουμε ομαδοποιήσει τα δεδομένα ανά ετικέτα, δηλαδή ανά κέντρο αθροίζουμε τις συντεταγμένες τους και τις διαιρούμε με το πλήθος τους ώστε να προκύψουν τα νέα κέντρα ως ο μέσος όρος των σημείων που ανήκουν στα παλιά κέντρα, όπως υποδεικνύει ο αλγόριθμος.

```
map(key, value):
    label = get_labels_for_each_point(value, centroids)
    emit(label, (value, 1))
```

```
reduce(key, value):
    sum_x, sum_y = 0, 0
    count = 0
    for i in value:
        sum_x += i[0][0]
        sum_y += i[0][1]
        count += i[1]
    new_x = sum_x / count
```

```
        new_y = sum_y / count
        emit(key, (new_x, new_y))
```

Σημειώνουμε ότι ο κώδικας λόγω τεχνικών περιορισμών του Pyspark πραγματοποιεί το παραπάνω Reduce σε δύο βήματα. Στο πρώτο υπολογίζει τα αθροίσματα και το counts και στο δεύτερο με ένα ακόμη Map κάνει την διαίρεση και βρίσκει τα νέα κέντρα. Θεωρητικά το παραπάνω είναι ισοδύναμο και μας επιτρέπει να γλυτώσουμε μία πράξη Map.

Τα παραπάνω εκτελούνται 3 φορές όπως ζητείται και προκύπτουν τα αποτελέσματα που φαίνονται στο αντίστοιχο αρχείο. Δίνουμε ενδεικτικά τα τελικά αποτελέσματα:

```
Centrer IDs     Coordinates
    1           (-74.00242408772402, 40.73170488692425)
    2           (-73.83747289194628, 40.716328670235036)
    3           (-73.99479622767157, 40.713172559751065)
    4           (-73.98812567171869, 40.74591444317248)
    5           (-73.96851068547537, 40.77206591259295)
```

Link to hdfs:
http://83.212.74.139:50070/explorer.html/

Στο παραπάνω link φαίνεται το hdfs και στις αντίστοιχες ενότητες βρίσκουμε τα αρχεία εισόδου και τα final results. Δοκιμάσαμε να κάνουμε Download από το GUI αλλά αυτό δεν επιτρέπεται λόγω της IP του slave μηχανήματος. Τυχόν λάθος μορφή του result στο hdfs ίσως να οφείλεται στο γεγονός ότι πρώτα το δημιουργούμε τοπικά και στη συνέχεια το τοποθετούμε στο hdfs.

Σημείωση: Όλα τα αποτελέσματα μαζί με τα log-files και τον πηγαίο κώδικα .py βρίσκονται σε αντίστοιχο φάκελο εντός του .zip.