Αναφορά εξαμηνιαίας εργασίας στο μάθημα "Προχωρημένα Θέματα Βάσεων Δεδομένων"

Λεωνίδας Αβδελάς, 03113182

Πίνακας Περιεχομένων

Μεθοδολογία που ακολουθήθηκε

Για την εκπόνηση της εργασίας, χρησιμοποιήσαμε τα παρακάτω περιβάλλοντα και εργαλεία:

- Apache Hadoop 2.7.6
- OpenJDK 1.8.0_222
- Apache Spark 2.3
- Anaconda 2019.07 και πιο συγκεκριμένα
    - Python 3.7
    - Jupyter 4.5.0
    - PySpark
    - findSpark 1.3.0

Το cluster στήθηκε σε 2 Virtual Machines του okeanos και δημιουργήθηκε ένας jupyter public server για την εύκολη διαχείρηση του jupyter notebook. Η γλώσσα Python προτιμήθηκε λόγω της οικιότητας του συγγραφέα με αυτήν, καθώς και την ευκολία της όσον αφορά prototyping και μικρά snippets κώδικα. Σε επίπεδο προγράμματος, έγινε χρήση των Spark RDDs ως μοντέλων αποθήκευσης δεδομένων και αυτού του API για την επίλυση των προβλημάτων. Αυτό σήμαινει ότι η διαδικασία και οι συναρτήσεις που χρησιμοποιήσαμε είχαν ένα επίπεδο abstraction πάνω από το MapReduce, γεγονός που μας έδινε περισσότερες δυνατότητες για το πώς να επεξεργαστούμε τα δεδομένα.

Ο τελικός κώδικας δεν είναι optimized, καθώς το readability προτιμήθηκε από τον μικρότερο αριθμό jobs. Η χρήση διαδοχικών map θα μπορούσε να αντικατασταθεί από μια πιο πολύπλοκη συνάρτηση map, όπως φαίνεται στα παραδείγματα παραπάνω.

Το hdfs site με τα αποτελέσματα μπορεί να βρεθεί εδώ.
Εδώ μπορείτε να δείτε όλα τα jobs που έτρεξαν.

1) Αναλυτική επεξεργασία δεδομένων

α) Μέση διάρκεια διαδρομής ανα ώρα εκκινησης

Ο ψευδοκώδικας σε MapReduce της εργασίας αυτής είναι:

```
map(key, value):
// key: trip_id;
// value: trip_data - array [start_time, end_time, start_lat, start_long, end_lat, end_long, cost]
        for each start_time s_t, end_time e_t in value:
                s_t_stripped = strip_time(s_t)
                e_t_stripped = strip_time(e_t)
                emit(s_t_stripped.hour, e_t_stripped - s_t_stripped)
reduce(key, values):
// key: an hour; value: an iterator over durations
        result = 0
        cnt = value.length
        for each count v in value:
                result += v
        emit(key, result/cnt)
```

β) Μέγιστο ποσό σε μια διαδρομή ανα εταιρία ταξί

Για να επιλύσουμε αυτό το πρόβλημα θα χρειαστούμε δύο βήματα MapReduce.

Αρχικά θα περάσουμε από τον mapper όλα τα δεδομένα, δηλαδή και τις πληροφορίες των ταξί και των εταιριών και θα ενώσουμε τα δύο dataset με βάση το id της διαδρομής που είναι μοναδικό για κάθε διαδρομή. Αυτό θα γίνει αυτόματα από τον combiner και το reduce βήμα απλά θα βγάλει ταυτοτικά αποτελέσματα.

Για το δεύτερο βήμα ο mapper θα αλλάξει το κλειδί στον αριθμό της εταιρίας και στο βήμα reduce, θα βρεί το μέγιστο μεταξύ των κοστών για αυτή την εταιρία.

Ο ψευδοκώδικας σε MapReduce της εργασίας αυτής είναι:

```
map(key, value):
```

```
// key: trip_id;
// value: trip_data - array [start_time, end_time, start_lat, start_long, end_lat, end_long, cost]
or // vendor
        if value is array
                emit(key, cost)
        else
                emit(key, vendor)

reduce(key, values):
// key: trip_id; value: an iterator over cost and vendor
        emit(key, values)
```

```
map(key, value):
// key: trip_id;
// value: array [cost, vendor]
        emit(vendor, cost)

reduce(key, values):
// key: vendor; value: an iterator costs
        result = 0
        for each cost c in value:
                result = max(c, result)
        emit(key, result)
```

2) Machine Learning - εκτέλεση k-means με fixed k

Για την επίλυση της άσκησης χρησιμοποιήσαμε την έτοιμη βιβλιοθήκη του spark για machine learning, καθώς αυτή μας εξασφαλίζει ταχύτερη εκτέλεση.

Παρ' όλα αυτά παρακάτω είναι οι MapReduce εργασίες που θα έπρεπε να γίνουν για να εκτελεστεί το αντίστοιχο πρόγραμμα με μόνο MapReduce. Για να γίνει αυτό, θα θεωρήσουμε ότι έχουμε τον πίνακα με τα αρχικά centroids διαθέσιμο σε όλους τους mappers. Ακόμα θεωρούμε ότι έχουμε απομονώσει τα δεδομένα των επιβιβάσεων των πελατών, το οποίο μπορεί να γίνει κρατώντας μόνο δύο στήλες από το αρχικό αρχείο, με ένα mapper, χωρίς περαιτέρω επεξεργασία.

```
k = 5
```

```
MAX_ITERATIONS = 3

Centroids = get_k_first_points(points, k)

eucl_dist(lat_1, long_1, lat_2, long_2):
        return sqrt((lat_1 - lat_2)^2 + (long_1 - long_2)^2)

map(key, value):
// key: start_lat
// value: start_long
        closer = 1
        dist = eucl_dist(start_lat, start_long, centroids[0][0], centroids[0][1])
        for each centroind c in centroids:
                if eucl_dist(start_lat, start_long, c[0], c[1]) < dist
                        closer = c.number
                        dist =  eucl_dist(start_lat, start_long, c[0], c[1])
                emit(closer, start_lat, start_long)

reduce(key, values):
// key: centroid; value: an iterator of start_lats and an iterator of start_longs
        new_cent_lat = avg(start_lats)
        new_cent_long = avg(start_longs)
        emit(key, new_cent_lat, new_cent_long)
```

Από τα αποτελέσματα του reducer, θα ανανεώσουμε την τιμή κάθε centroid. Έπειτα, ανάλογα με τον αριθμό των ITERATIONS, οι ίδιοι mappers και reducers θα τρέξουν για MAX_ITERATIONS - i φορές ανάλογα με την φορά στην οποία βρισκόμαστε.

3) Γράφοι - PageRank computation

Αρχικά για να δομήσουμε τα δεδομένα, χρησιμοποιήσαμε έναν mapper και μετά ένα reducer. Ο στόχος ήταν από γραμμές αρχείου/συμβολοσειρές να δημιουργήσουμε μια λίστα από σελίδες, τα outbound links της και το αρχικό PageRank της. Ο combiner μας εξασφαλίζει ότι θα δημιουρηθεί η λίστα των outbound links.

Ο ψευδοκώδικας σε MapReduce της εργασίας αυτής είναι:

```
map(key, value):
// key: none
// value: string of “Page1 Page2” showing outbound links
        page1, page2 = value.split(‘\t’)
        emit(page1, page2)
reduce(key, values):
// key: source_page; value: an iterator over destination_pages
        emit(key, result, 0.5)
```

Ο αλγόριθμος του PageRank δίνεται από την συνάρτηση:

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

Το σημαντικό σημείο για να φτιάξουμε το MapReduce για αυτή την εργασία είναι το γεγονός ότι κάθε outbound link συνεισφέρει στην σελίδα στην οποία είναι inbound ποσότητα ίση με το PageRank της σελίδας δια το πλήθος των outbound links της. Δηλαδή μια σελίδα με PageRank 0.5 και 5 outbound links θα προσφέρει 0.1 σε κάθε σελίδα που είναι inbound link. Αρα δημιουργώντας ζεύγη (page, inbound_rank) για κάθε link σε κάθε σελίδα, με ένα reduce μετά μπορούμε να βρούμε το PageRank της.

Ο ψευδοκώδικας σε MapReduce της εργασίας αυτής είναι:

```
map(key, value):
// key: outbound_page
// value: iterator of the list of pages that that key has outbound links to, pagerank of key
        pr_contribution = lenght(value[0])/pagerank_of_key
        for each page p in value[0]:
                emit(p, pr_contribution)

reduce(key, values):
// key: page; value: an iterator over pagerank contributions
        summation  = 0
        for contribution c in value
                summation += contribution
        pagerank = 1-d/N + d * summation
        emit(page, pagerank)
```

Θεωρούμε τα N και d γνωστά.

Αν θέλουμε να τρέξουμε παραπάνω από μια φορές της εργασία για να φτάσουμε σε σύγκλιση, το μόνο που αλλάζει είναι ότι θα χρειαστεί ένας mapper να ανανεώνει το pagerank της κάθε σελίδας που έχουμε από το πρώτο reduce.

4) Γραμμική Άλγεβρα - Πολλαπλασιασμός Πινάκων

Για αυτή την άσκηση, χρησιμοποιήσαμε δύο τρόπους επίλυσης. Ο πρώτος ήταν μέσω των κατανεμημένων πινάκων που έχει το Spark. Καθώς αυτός δεν έχει ξεκάθαρα βήματα MapReduce, χρησιμοποιήσαμε και έναν δεύτερο που τα βήματα που γίνονται είναι πιο ξεκάθαρα. Η διαδικασία αποτελείται απο δύο εργασίες MapReduce.

Ο ψευδοκώδικας σε MapReduce της εργασίας αυτής είναι:

```
map(key, value):
// key: the array of origin (A or B)
// value: tuple of (i or j, j or k, mij or mjk)
        if key is A
                emit(j, (key, i, mij)
        else
                emit(j, (key, k, mjk)

reduce(key, values):
// key: j; value: an iterator over is and ks and their mijs/mjks
        for i in value[0]
                for k in value[1]
                        emit((i,k), mij * mik)
```

Ο ψευδοκώδικας σε MapReduce της δεύτερης εργασίας είναι:

```
map(key, value):
// key:(i, k)
// value: the value mij*mik
        emit(key, value)
reduce(key, values):
// key: (i,k); value: an iterator mij * mjk for certain j
        sum = 0
        for i in value
                sum +=  i
        emit(key , sum)
```

Έτσι στο τέλος παίρνουμε την τιμή για κάθε στοιχείο του πίνακα.

Πηγές

- Learning Spark: Lightning-Fast Big Data Analysis
- Mining of Massive Datasets
- PageRank Spark Example