Εθνικο Μετσοβιο Πολυτεχνειο

Ηλετρολογων Μηχανικων & Μηχανικων Υπολογιστων

Προχωρημένα Θέματα Βάσεων Δεδομένων

Αναφορά Άσκησης MapReduce

Δοντάς Σπυρίδων
9ο εξάμηνο
03114141

Τσιούρβας Αστέριος
9ο εξάμηνο
03114133

Αθήνα, Φεβρουάριος 2019

Περιεχόμενα

1 | Αναλυτική Επεξεργασία Δεδομένων

1.1 | Μεθοδολογία

1.1.1 | Ερώτημα Α

Το πρώτο ερώτημα ζητά να βρεθεί η μέση διάρκεια διαδρομής (σε λεπτά) ανά ώρα έναρξης της διαδρομής και να ταξινομιθεί το αποτέλεσμα με βάση την ώρα έναρξης σε αύξουσα σειρά. Για την επίλυση του ερωτήματος, διαβάζουμε τα δεδομένα απο τό αρχείο yellow_tripdata_1m.csv και χρησιμοποιούμε μία φάση **MapReduceMap**, όπως αναφέρεται και στη σχετική υπόδειξη.

Στη φάση **Map**, για κάθε εγγραφή του αρχείου υπολογίζουμε τη χρονική διάρκεια κάθε διαδρομής σε λεπτά χρησιμοποιόντας την ημερομηνία και ώρα έναρξης και λήξης της και κάνουμε emit tuples της μορφής *(key = start_hour, value = duration)*.

Στη φάση **Reduce**, υπολογίζουμε το συνολικό άθροισμα των διαδρομών ανά ώρα έναρξης και και για κάθε ώρα έναρξης κάνουμε emit tuples της μορφής *(key = start_hour, value = (total_trips,duration))*.

Στη φάση **Map**, για κάθε tuple υπολογίζουμε τη μέση διάρκεια διαδρομής ανά ώρα έναρξης και κάνουμε emit tuples της μορφής *(key = start_hour, value = avg_duration)*. Τα δεδομένα μας είναι στη μορφή που επιθυμούμε.

Τέλος, ταξινομούμε τα αποτελέσματα με βάση την ώρα έναρξης της διαδρομής και τα γράφουμε σε αρχείο και τα εμφανίζουμε.

1.1.2 | Ερώτημα Β

Το πρώτο ερώτημα ζητά να βρεθεί το μέγιστο ποσό που πληρώθηκε σε μία διαδρομή σε κάθε εταιρία ταξί. Για την επίλυση του ερωτήματος, διαβάζουμε τα δεδομένα από τα αρχεία yellow_tripdata_1m.csv και yellow_tripvendors_1m.csv. Χρησιμοποιούμε μία φάση **MapJoin** και έπειτα άλλη μία **MapReduce**.

Στη φάση **Map**, για κάθε εγγραφή του yellow_tripdata_1m.csv κάνουμε emit tuples της μορφής *(key = trip_id, value = company_id)*, ενώ για κάθε εγγραφή του yellow_tripvendors_1m.csv κάνουμε emit tuples της μορφής *(key = trip_id, value = amount)*.

Στη συνέχεια, πραγματοποιούμε inner join με βάση το *trip_id* μεταξύ των δύο παραπάνω *RDDs*.

Στη φάση **Map**, για κάθε tuple της μορφής *(key = trip_id, value = (company_id,amount))* κάνουμε emit tuples της μορφής *(key = company_id, value = amount)*.

Στη φάση **Reduce**, βρίσκουμε το μέγιστο ποσό ανά εταιρία και κάνουμε emit tuples της μορφής *(key = company_id, value = max_amount)*.

Τέλος, ταξινομούμε τα αποτελέσματα με βάση το key = *company_id*, τα γράφουμε σε αρχείο και τα εμφανίζουμε.

1.2 | Ψευδοκώδικας

1.2.1 | Ερώτημα Α

Map Input

```
1: function MAP(key, value)          ▷ key is some id, value is the line of the csv file
2:     line ← value.split(",")
3:     start ← line[1]
4:     end ← line[2]
5:     start ← start.format("%Y − %m − %d %H : %M : %S")
6:     end ← end.format("%Y − %m − %d %H : %M : %S")
7:     emit (start.hour, diff_in_minutes(end − start))
8: end function
```

Reduce Mapped Input

```
1: function REDUCE(key, values)     ▷ key is some start hour, value is list of durations
2:     sum ← 0
3:     for dur in values do
4:         sum ← sum + dur
5:     end for
6:     average ← sum/len(values)
7:     emit (key, average)
8: end function
```

1.2.2 | Ερώτημα Β

Map Vendors

```
1: function MAP(key, value)          ▷ key is some id, value is csv line
2:     line ← value.split(",")
3:     id ← line[0]
4:     vendor ← line[1]
5:     emit (id, ("vendor", vendor))
6: end function
```

Map Trips

```
1: function MAP(key, value)                    ▷ key is some id, value is csv line
2:     line ← value.split(",")
3:     id ← line[0]
4:     payment ← line[7]
5:     emit (id, ("pay", payment))
6: end function
```

Join Mapped Inputs through Reduce

```
1: function REDUCE(key, values)    ▷ key is trip id, value is list of one vendor id and one payment
2:     vendor ← find_assoc(values, "vendor")
3:     payment ← find_assoc(values, "pay")
4:     emit (vendor, payment)
5: end function
```

Map Joined Input

```
1: function MAP(key, value)
2:     emit (key, value)
3: end function
```

Reduce to Find Maximum

```
1: function REDUCE(key, values) ▷ key is vendor id, values is list of payment amounts
2:     payment ← 0
3:     for pay in values do
4:         if pay > payment then
5:             payment ← pay
6:         end if
7:     end for
8:     emit (key, payment)
9: end function
```

2 | Machine Learning - Εκτέλεση k-Means με fixed k

2.1 | Μεθοδολογία

Η άσκηση ζητάει χρησιμοποιώντας τα δεδομένα του πρώτου ερωτηματος να βρούμε τις κεντρικές συντεταγμένες των top 5 περιοχών επιβίβασης πελατών με την χρήση του αλγορίθμου $k-means$. Για την επίλυση, διαβάζουμε τα δεδομένα από το αρχείο yellow_tripdata_1m.csv.

Αρχικα, για κάθε εγγραφή του αρχείου yellow_tripdata_1m.csv κρατάμε τις συντεταγμένες, αρχικοποιούμε μεταβλητές και ορίζουμε τα κέντρα ως τις πρώτες 5 τιμές του αρχείου.

Σε κάθε επανάληψη, εφαρμόζουμε μία φάση **Map** για να δημιουργήσουμε ένα RDD με τις συντεταγμένες κάθε σημείου και ένα αντίγραφο των κέντρων. Έπειτα, εφαρμόζουμε μία φάση **Map** και κάνουμε emit tuples της μορφής *(key = closest_cluster_id, value = coordinates)*. Στη συνέχεια, εφαρμόζουμε μία φάση **MapReduceMap**, για την εύρεση των νέων κέντρων με βάση τους μέσους όρους ανά συντεταγμένη (η διαδικασία είναι παρόμοια με την εύρεση του μέσου όρου στην άσκηση 1 με τη διαφορά ότι εδώ βρίσκουμε μέσο όρο σε πολλά στοιχεία ανά στοιχείο).

Μετά το πέρας των 5 επαναλήψεων τα έχουμε βρει τα τελικά κέντρα του αλγορίθμου $k-means$, τα γράφουμε σε αρχείο και τα εμφανίζουμε.

2.2 | Ψευδοκώδικας

Προετοιμασία αρχικού dataset:

Map Input

```
1: function MAP(key, value)                    ▷ key is some id, value is csv line
2:     line ← value.split(",")
3:     lng ← line[3]
4:     lat ← line[4]
5:     emit (key, (lng, lat))
6: end function
```

Reduce Mapped Input

```
1: function REDUCE(key, values)
2:     emit (key, values)
3: end function
```

Τώρα, θεωρώντας πως έχουμε υπολογίσει τα αρχικά κέντρα, θα υλοποιήσουμε τον κύριο αλγόριθμο:

Map Population

Require: *centroids* ▷ centroids are tuples of id, tuple of lng, lat
Require: *euclidean* ▷ euclidean is a function
1: **function** MAP($key, value$) ▷ key is some id, value is tuple of lng, lat
2: $id \leftarrow -1$
3: $distance \leftarrow INF$ ▷ infinite starting distance
4: **for** *centroid* **in** *centroids* **do**
5: $dist \leftarrow euclidean(value, centroid[1])$
6: **if** $dist < distance$ **then**
7: $id \leftarrow centroid[0]$
8: $distance \leftarrow dist$
9: **end if**
10: **end for**
11: **emit** $(id, value)$
12: **end function**

Reduce Centroids to Find New Centre

1: **function** REDUCE($key, values$) ▷ key is centroid id, values are coordinates in this centroid
2: $lngs \leftarrow 0$
3: $lats \leftarrow 0$
4: **for** (lng, lat) **in** *values* **do**
5: $lngs \leftarrow lngs + lng$
6: $lats \leftarrow lats + lat$
7: **end for**
8: $N \leftarrow len(values)$
9: $lng \leftarrow lngs/N$
10: $lat \leftarrow lats/N$
11: **emit** $(key, (lng, lat))$
12: **end function**

Από το τελευταίο `reduce` λαμβάνουμε τα νέα `centroids`, τα οποία θα ξαναχρησιμοποιήσουμε στην επόμενη επανάληψη, ενώ η διαδικασία της παραπάνω `MapReduce` επαναλαμβάνεται 3 φορές.

3 | Γράφοι - PageRank Computation

3.1 | Μεθοδολογία

Η άσκηση ζητάει να βρούμε το *PageRank* κάθε κόμβου που βρίσκεται στα δεδομένα της Google με βάση τον επαναληπτικό τύπο που δίνεται στην εκφώνηση. Για την επίλυση του ερωτήματος, διαβάζουμε τα δεδομένα από το αρχείο web-Google.txt.

Αρχικά, εφαρμόζουμε μία φάση **Map** και για να δημιουργήσουμε ένα RDD που περιέχει σε tuples τα from και to ids των nodes. Έπειτα, πραγματοποιούμε **groupByKey** προκειμένου να ομαδοποιήσουμε τα δεδομένα σε tuples της μορφής *(key = from_id, value = list_of_to_ids)* και πραγματοποιούμε τις αρχικοποιήσεις παράγοντας tuples της μορφής *(key = id, value = score)*.

Σε κάθε επανάληψη, εφαρμόζουμε μία φάση **Join** για να δημιουργήσουμε tuples της μορφής *(key = from_id, value = (list_of_to_ids,score))*. Στη συνέχεια, εφαρμόζουμε μία φάση **FlatMapReduceMap** έτσι ώστε να αποκτήσουμε - από τη φάση **FlatMap** - tuples της μορφής *(key = from_id, value = a_fraction_of_outbounds_links)* , δηλαδή tuples της μορφής $(id, \frac{PR(pj)}{L(pj)})$, στη συνέχεια - φάση **Reduce** - να υπολογίσουμε tuples της μορφής $(key = from\_id, value = \sum_{pj \in M(pi)} \frac{PR(pj)}{L(pj)})$ και τέλος στη φάση **Map** εφαρμόζουμε τον τύπο υπολογισμού *PageRank* και λαμβάνουμε tuples της μορφής *(key = id, value = score)*.

Μετά το πέρας των 5 επαναλήψεων τα έχουμε βρει τα *PageRank* scores για κάθε *id*, τα ταξινομούμε ως προς *id*, τα γράφουμε σε αρχείο και τα εμφανίζουμε.

3.2 | Ψευδοκώδικας

Προετοιμασία αρχικού dataset:

Map Input

```
1: function MAP(key, value)                    ▷ key is some id, value is csv line
2:     line ← value.split(",")
3:     in ← line[0]
4:     out ← line[1]
5:     emit (in, out)
6: end function
```

Reduce Mapped Input

```
Require: scores  ▷ scores for each of the nodes, as dictionary, initialised at 0.5 for each node
1: function REDUCE(key, values)                ▷ key is from id, values is list of to ids
2:     emit (key, (values, scores[key]))
3: end function
```

Τώρα θα εφαρμόσουμε το κύριο κομμάτι του κώδικα:

Map Scores

```
1: function MAP(key, value)          ▷ key is some id, value is tuple of list, score
2:     ids ← value[0]
3:     score ← value[1]
4:     N ← len(ids)
5:     for id in ids do
6:         s ← score/N
7:         emit (id, s)
8:     end for
9: end function
```

Reduce Scores

```
Require: N                                              ▷ Number of Nodes
Require: d                                              ▷ float in [0, 1]
1: function REDUCE(key, values)          ▷ key is id, values is list of scores
2:     sum ← 0
3:     for score in values do
4:         sum ← sum + score
5:     end for
6:     score ← ((1 − d)/N) + (d ∗ score)
7:     emit (key, score)
8: end function
```

Η παραπάνω διαδιακασία πρέπει να επαναληφθεί 5 φορές, στις οποίες προϋποθέτουμε ότι το τελικό αποτέλεσμα της `reduce scores` είναι αυτό που χρησιμοποιείται στην αρχική `reduce`.

4 | Γραμμική Άλγεβρα - Πολλαπλασιασμός Πίνακα

4.1 | Μεθοδολογία

Η άσκηση ζητάει να υπολογιστεί το γινόμενο δύο πινάκων. Για την επίλυση, διαβάζουμε τα δεδομένα από τα αρχεία A.csv και B.csv και χρησιμοποιούμε μία φάση **MapReduceMap**, έπειτα μία φάση **CartesianProduct** και τέλος μία φάση **MapMap**

Στη φάση **Map**, για κάθε εγγραφή του αρχείου A.csv κάνουμε emit tuples της μορφής *(key = row_id_of_A, value =(col_id_of_A,value)*. Επιπλέον, για κάθε εγγραφή του αρχείου B.csv κάνουμε emit tuples της μορφής *(key = col_id_of_B, value =(row_id_of_B,value)*.

Στη φάση **Reduce**, για κάθε tuple της πρώτης φάσης αθροίζουμε τα values (ως λίστες) και κάνουμε emit tuples της μορφής *(key = row_id, value =[(col_id,value)]* για την πρώτη περίπτωση και για τη δεύτερη περίπτωση *(key = col_id, value =[(row_id,value)]*.

Στη φάση **Map**, για κάθε tuple της πιο πάνω φάσης κάνουμε ταξινόμηση τα values με βάση το *col_id* και κάνουμε emit tuples της μορφής *(key = row_id, value=[value])* για την πρώτη περίπτωση, ενώ για τη δεύτερη περίπτωση κάνουμε ταξινόμηση τα values με βάση το *row_id* και κάνουμε emit tuples της μορφής *(key = col_id, value=[value])*.

Στη φάση **CartesianProduct**, δημιουργούμε tuples της μορφής *(key = (row_id_of_A, col_id_of_B), value = ([value_A], [value_B])*.

Τέλος, στη φάση **MapMap**, υπολογίζουμε την τιμή του στοιχείου της θέσης *(row_id_of_A, col_id_of_B)* του τελικού πίνακα, δηλαδή κάνουμε emit tuples της μορφής *(key = (row_id_of_A, col_id_of_B), value = final_value)*.

Μετά το πέρας της μεθόδου τα έχουμε βρει τις τιμές του τελικού πίνακα, τα γράφουμε σε αρχείο και τα ταξινομούμε ως προς $(row\_id\_of\_A, col\_id\_of\_B)$.

4.2 | Ψευδοκώδικας

Στον ψευδοκώδικα, θα υλοποιήσουμε κάπως διαφορετικά το καρτεσιανό γινόμενο (μέθοδος `cartesian` του `spark`).

Map Array A

Require: M ▷ number of columns

```
1: function MAP(key, value)                ▷ key is some id, value is csv line
2:     line ← value.split(",")
3:     row ← line[0]
4:     col ← line[1]
5:     value ← line[2]
6:     for i ← 0, M do
7:         emit ((row, i), ("A", col, value))
8:     end for
9: end function
```

Map Array B

Require: N ▷ number of rows

```
1: function MAP(key, value)                ▷ key is some id, value is csv line
2:     line ← value.split(",")
3:     row ← line[0]
4:     col ← line[1]
5:     value ← line[2]
6:     for i ← 0, N do
7:         emit ((i, col), ("B", row, value))
8:     end for
9: end function
```

Reduce Arrays and Multiply

```
1: function REDUCE(key, values)  ▷ key is (row, col), values is list of elements from A and B
2:     A = [ ]
3:     B = [ ]
4:     for element in values do
5:         if element[0] is "A" then
6:             A.append((element[1], element[2]))
7:         else
8:             B.append((element[1], element[2]))
9:         end if
10:    end for
11:    A.sort(key = lambda x : x[0])
12:    B.sort(key = lambda x : x[0])
13:    sum ← 0
14:    for i ← 0, len(A) do
15:        sum ← sum + A[i] * B[i]
16:    end for
17:    emit (key, sum)
18: end function
```

5 | Παρατηρήσεις - Επισημάνσεις

5.1 | Jupyter Notebook

Για την ευκολότερη και καλύτερη οργάνωση του κώδικά μας εγκαταστήσαμε Jupyter Notebook.

5.2 | Github

Παρακάτω δίνεται σχετικό link στο Github για τον κώδικα που χρησιμοποιήθηκε:

https://github.com/xspirus/AdvancedDatabases

Το link θα είναι δημόσιο μετά το πέρας της προθεσμίας.

5.3 | HDFS

Το link για το `HDFS` στο οποίο βρίσκονται τα δεδομένα μας είναι:

http://83.212.73.250:50070