Προχωρημένα Θέματα Βάσεων Δεδομένων

Εξαμηνιαία Εργασία

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, ΕΜΠ

Ομάδα 7

Οικονόμου Νικόλαος (03120014)

Ραφτόπουλος Μιχαήλ (03120114)

Αποθετήριο GitHub

Ιανουάριος 2025

Query 1

Εκτελέστηκε το query 1, τόσο με το DataFrame API, όσο και με το RDD API του Spark. Συγκρίνοντας τον χρόνο των δύο υλοποιήσεων, η υλοποίηση με DataFrame API αποδείχθηκε ταχύτερη[1]. Το αποτέλεσμα αυτό είναι αναμενόμενο, καθώς το DataFrame API αποτελεί μια υψηλωτερη αφαιρετικά διεπαφή, με πολλές βελτιστοποιήσεις να πραγματοποιούνται στο υπόβαθρο (από τον Catalyst Optimizer). Με το RDD API μπορούμε θεωρητικά να πετύχουμε την ίδια αποδοτικότητα (αφού και το DataFrame API χρησιμοποιεί στο υπόβαθρο RDDs), αλλά απαιτείται πολλή εμειρία και προσεκτικός χειρισμός.

Age Group	Count
Adults	121093
Young Adults	33605
Children	15928
Elderly	5985

Πίνακας 1: Αποτελέσματα query 1.

Query 2

(α) Υπολογίστηκαν για κάθε έτος τα 3 Αστυνομικά Τμήματα με το υψηλότερο ποσοστό κλεισμένων υποθέσεων και ταξινομήθηκαν ανά έτος και ανά ποσοστό. Χρησιμοποιήθηκαν δύο διαφορετικά APIs του Spark: το DataFrame API και το SQL API. Μετά από αρκετές επαναλήψεις, η υλοποίσηση με SQL API αποδείχθηκε ταχύτερη από το αυτήν με DataFrame API. Θεωρητικά δε θα αναμέναμε ουσιαστική διαφορά μεταξύ των δύο, καθώς αποτελούν απλά διαφορετικές διεπαφές του ίδιου optimizer. Η απόκλιση λοιπόν των δύο υλοποιήσεων μάλλον οφείλεται στον τρόπο που αυτές είναι γραμμένες, με τον κώδικα σε DataFrames να οδηγεί σε περισσότερα operations.

(β) Σε αυτό το σημείο, έγινε η μετατροπή των δεδομένων εισόδου από `.csv` σε `.parquet`. Εκτελέστηκε η ίδια υλοποίηση SQL, χρησιμοπιώντας το δεύτερο format. Η υλοποίηση με τα δεδομένα σε `.parquet` ήταν ταχύτερη. Αυτό είναι αναμενόμενο, καθώς το `.parquet` αποτελεί έναν τύπο αρχείου σχεδιασμένο για κατανεμημένα filesystems.

year	AREA NAME	closed_rate	#
2010	Rampart	32.84713448949121	1
2010	Olympic	31.515289821999087	2
2010	Harbor	29.36028339237341	3
2011	Olympic	35.0400600901352	1
2011	Rampart	32.496447181430604	2
...			

Πίνακας 2: Αποτελέσματα query 2 (φαίνονται μόνο οι πρώτες γραμμές).

[1]Σε αυτό το σημείο να αναφερθεί ότι στις διάφορες δοκιμές του χρόνου εκτέλεσης των queries τα αποτελέσματα παρουσιάζαν πολύ μεγάλη διακύμανση, που πιθανώς οφείλεται στο cloud περιβάλλον εκτέλεσης. Οι σχετικοί χρόνοι όμως ανάμεσα στις υλοποιήσεις παρέμεναν σταθεροί. Για το λόγο αυτό, σημαντικότεροι είναι οι σχετικοί χρόνοι και όχι τα απόλυτα νούμερα.

Query 3

Ύστερα από χρήση της μεθόδου `.explain()` στα διαφορετικά joins που πραγματοποιούνται σε αυτό το query, πήραμε τα παρακάτω αποτελέσματα για τις επιλογές που κάνει ο Catalyst Optimizer του Spark:

- Για Join μεταξύ Census και Income Dataset (μικρός πίνακας) χρησιμοποιεί **BroadcastHash Join**. Η επιλογή βγάζει νόημα, αφού το broadcast join αξιοποείται σε περιπτώσεις όπως αυτήν, δηλαδή για συνένωση μεγάλου με μικρό πίνακα.
- Για Join μεταξύ Crimes και Census Dataset χρησιμοποιεί **Range Join**. Η επιλογή είναι λογική αφού αξιοποείται σε περιπτώσεις με αριθμητικές τιμής όπως στην συγκεκριμένη που προσπαθεί να ταιριάξει τις συντεταγμένες των εγκλημάτων με την γεωμετρία του κάθε block. Είναι, λοιπόν, η πλέον κατάλληλη μέθοδος συνένωσης μεταξύ δύο dataset που συνδέονται με σχέσεις του τύπου:

$$\leq, \geq, <, > \text{ ή BETWEEN}$$

- Για Join μεταξύ των αποτελεσμάτων από τα δύο παραπάνω datasets χρησιμοποιεί **SortMerge Join**. Η επιλογή είναι αποδεδειγμένα καλή για συνθήκες ισότητας όπως στην συγκεκριμένη που γίνεται με βάση το κλειδί "COMM".

Χρόνοι εκτέλεσης

Οι χρόνοι εκτέλεσης χωρίς την εφαρμογή κάποιου **.hint()** είναι:

- Φόρτωση δεδομένων: **18.56** seconds
- Μέσο εισόδημα ανά άτομο σε κάθε περιοχή: **16.77** seconds
- Εγκλήματα ανά άτομο σε κάθε περιοχή: **45.58** seconds
- Συνένωση τελικών αποτελεσμάτων: **22.53** seconds

Παρακάτω βλέπουμε τον πίνακα με την εφαρμογή των διαφορετικών hints για τα επιλεγμένα Join operations (σημειώνεται πως ο χρόνος φόρτωσης των δεδομένων ήταν πάνω κάτω ο ίδιος σε όλες τις εκτελέσης 18 sec). Υπογραμμισμένα είναι τα καλύτερα αποτελέσματα. Οι

CODE PART	BROADCAST	MERGE	SHUFFLE_HASH	SHUFFLE_REPLICATE_NL
MEDIAN INCOME PER PERSON	15 (BroadcastHash)	17.4 (SortMerge)	16.2 (ShuffledHash)	16.3 (BroadcastHash)
CRIMES PER PERSON	(Spark crushes)	40.5 (Range)	37.6 (Range)	38.5 (Range)
FINAL MERGE	22.2 (BroadcastHash)	21.5 (SortMerge)	23.1 (ShuffledHash)	26.2 (BroadcastHash)

Πίνακας 3: Comparison of Join Strategies

διαφορές δεν είναι μεγάλες αλλά υπάρχουν και ταυτίζονται με τις προτάσεις του catalyst optimizer. Αυτό βασίζεται στην αιτιολόγηση που δίνουμε και παραπάνω για τις επιλογές που κάνει και επιβεβαιώνεται από τα αποτελέσματα μας. Οι χρόνοι προήλθαν ως μέσος όρος διαφορετικων εκτελέσεων.
Σημειώνουμε, επίσης, τις παρακάτω παρατηρήσεις:

- Το Broadcast Join αποτυγχάνει στην συνένωση δύο μεγάλων Dataset, δηλαδή του Crimes με το Census, μιας και δεν μπορεί να το υποστηρίξει η μνήμη.
- Το Spark επιλέγει αυτόματα το Range Join για την συνένωση των δύο παραπάνω, λόγω της χρήσης της μεθόδου **ST_Within**.
- Στην προσπάθεια να μετρήσουμε τους χρόνους εκτέλεσης έγινε εμφανής η στρατηγική του lazy evaluation, μιας και τα timer ακριβώς πριν και μετά την εντολή του join ήταν πολύ μικρά, αφού δεν συνέβαινε τότε κατά την διάρκεια της εκτέλεσης. Γι' αυτόν τον λόγο όπως φαίνεται και στον κώδικα υπολογίσαμε τον συνολικό χρόνο που έκανε το πρόγραμμα για το πρώτο μισό (Median per person ανά περιοχή), το δεύτερο μισό (Crimes per person ανά περιοχή) και τέλος την συνένωση των δύο (Final combined result). Αυτοί οι τελικοί χρόνοι που φαίνονται και στον πίνακα επηρεάζονταν αναλόγως και την στρατηγική Join που χρησιμοποιούσαμε.

COMM	...	Median Income Per Person	...	Crimes Per Person Ration
Adams-Normandie		8791.4583		0.73693
Alsace		11239.50119		0.55133
Angeles National Forest		28117.65		11.35
...				

Πίνακας 4: Αποτελέσματα query 3 (φαίνονται μόνο οι πρώτες γραμμές και επιλεγμένες στήλες).

Query 4

Τα αποτελέσματα σχετικά με την επίδοση του query για τα διαφορετικά configurations αναλύονται παρακάτω. Πριν από αυτό, χωρίς το μάθημα να αποτελεί μάθημα στατιστικής ανάλυσης θέλαμε να παρατηρήσουμε τις αξιοσημείωτες διαφορές μεταξύ των φτωχότερων και των πλουσιότερων περιοχών του Los Angeles. Το ένα ενδιαφέρον είναι συνολικός αριθμός των εγκλημάτων (άρα και των θυμάτων) που παρατηρούμε τουλάχιστον τάξη μεγέθους παραπάνω στις φτωχότερες περιοχές. Επίσης, ενδιαφέρον αποτέλεσμα αποτελούν οι φυλετικές διαφορές στα θύματα, πράγμα που απεικονίζει και τις φυλετικές διαφορές στους ίδιους τους κατοίκους της κάθε περιοχής με τις πλουσιότερες περιοχές να έχουν κύρια θύματα Λευκούς ("White") και τις φτωχότερες Ισπανόφωνες φυλές ("Hispanic/Latin/Mexican") και Αφροαμερικάνους ("Black").

Τα αποτελέσματα για τα διαφορετικά configuration μετά από μέσο όρο 10 εκτελέσεων για το καθένα φαίνονται παρακάτω:

- 1 core/2GB memory: **7.559s**
- 2 cores/4GB memory: **7.182s**
- 4 cores/8GB memory: **7.006s**

Με βάση τα αποτελέσματα που πήραμε δεν φαίνεται να κλιμακώνει ο κώδικας μας. Αυτό μπορεί να οφείλεται σε πολλούς παράγοντες και αναλύουμε κάποιους από αυτούς. Το μέγεθος των dataset δεν είναι πάρα πολύ μεγάλο με αποτέλεσμα το overhead του spark (data

shuffling, setting up tasks) να ξεπερνά τα οφέλη που μπορεί να έχουν κάποια παραπάνω cores και extra μνήμη. Επίσης, είναι πιθανό ένα bottleneck του χρόνου εκτέλεσης να είναι το φόρτωμα των δεδομένων από το S3 πράγμα που είναι κοινό σε όλα τα configurations. Συνοπτικά δηλαδή, υπάρχει περίπτωση και τα ”λιγότερα” resources να επαρκούν για τον υπολογισμό του ζητούμενο και γι’ αυτό να μην κάνει μεγάλη διαφορά η προσθήκη μνήμης και πυρήνων. Να σημειωθεί ότι αξιοποιούμε προηγουμένως υπολογισμένα αποτελέσματα του Median Income per Person (Query 3) για να ξεχωρίσουμε τις τρεις πλουσιότερες και τις τρεις φτωχότερες περιοχές του LA.

vict_descent	total_victims	...
White	8429	
Other	1125	
Hispanic/Latin/Mexican	868	
Unknown	651	
...		

Πίνακας 5: Αποτελέσματα query 4 για την περίπτωση των περιοχών με το υψηλότερο κατά κεφαλήν εισόδημα (φαίνονται μόνο οι πρώτες γραμμές και επιλεγμένες στήλες).

vict_descent	total_victims	...
Hispanic/Latin/Mexican	47026	
Black	17151	
White	7265	
Other	3256	
...		

Πίνακας 6: Αποτελέσματα query 4 για την περίπτωση των περιοχών με το υψηλότερο κατά κεφαλήν εισόδημα (φαίνονται μόνο οι πρώτες γραμμές και επιλεγμένες στήλες).

Query 5

Πραγματοποιήθηκε η υλοποίηση του Query 5 και εκτελέστηκε με τα 3 ζητούμενα configuration. Για τον υπολογισμό του χρόνου εκτέλεσης, το πρόγραμμα επαναλήφθηκε 10 φορές για κάθε configuration και υπολογίστηκε ο μέσος χρόνος για κάθε περίπτωση. Παρακάτω φαίνονται τα αποτελέσματα:

- 2 executors × 4 cores/8GB memory: **9.82s**
- 4 executors × 2 cores/4GB memory: **7.60s**
- 8 executors × 1 core/2GB memory: **7.25s**

Βλέπουμε ότι πολλοί «αδύναμοι» executors αποδίδουν καλύτερα απ' ο, τι λίγοι «ισχυροί». Αυτή η παρατήρηση δεν αποτελεί έκπληξη. Από τη «φύση» της, η κατανεμημένη βάση δεδομένων χρησιμοποιεί κατά κόρον την παραλληλοποίηση. Οι υψηλά παραλληλοποιήσιμες αυτές ενέργειες εποφελούνται από περισσότερες, έστω και πιο «αδύναμες», διεργσίες.

DIVISION	crime_count	mean_distance
HOLLYWOOD	213080	2.269
VAN NUYS	211457	3.181
WILSHIRE	198150	2.921
SOUTHWEST	186742	2.395
...		

Πίνακας 7: Αποτελέσματα query 5 (φαίνονται μόνο οι πρώτες γραμμές).