Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Προχωρημένα Θέματα Βάσεων Δεδομένων
9ο εξάμηνο, ακαδημαϊκό έτος 2025-2026

Εξαμηνιαία Εργασία

Ομάδα: group16

Τάτση Αικατερίνη Δανάη, ΑΜ: 03121146

Σπυρόπουλος Νικολας, ΑΜ: 03121202

Περιεχόμενα

1 Query 1

Το πρώτο ερώτημα μας ζητείται να υλοποιηθεί με 3 διαφορετικούς τρόπους - χρησιμοποιώντας το DataFrame API μία φορά χωρίς και μία φορά με User Defined Function (UDF) και χρησιμοποιώντας το RDD API. Από την θεωρία ξέρουμε ότι όταν χρησιμοποιούμε το RDD API, το φυσικό πλάνο των πράξεων το ορίζουμε εμείς επακριβώς. Δεν μπορεί, δηλαδή, να παρέμβει ο Catalyst Optimizer. Από την άλλη, όταν χρησιμοποιούμε το DataFrame API, ο Catalyst παρεμβαίνει και βελτιστοποιεί το query μας. Όμως, στην περίπτωση μιας συνάρτησης ορισμένης από τον χρήστη (UDF), ο Catalyst την "βλέπει" ως ένα μαύρο κουτί και το συγκεκριμένο κομμάτι δεν μπορεί να το αλλάξει. Επομένως, περιμένουμε η γρηγορότερη υλοποίηση να είναι το DataFrame χωρίς UDF, στην συνέχεια το DataFrame με UDF και τέλος το RDD.

Πράγματι, μετρώντας τους χρόνους εκτέλεσης των τριών παραπάνω υλοποιήσεων επαληθεύουμε αυτό που αναμέναμε. Οι χρόνοι φαίνονται στον πίνακα 1. Παρατηρούμε ότι η μη χρήση UDF είναι περίπου $1.3s$ γρηγορότερη από την χρήση μιας UDF, ενώ η υλοποίηση με RDD είναι περίπου 2 φορές πιο αργή από τις υλοποιήσεις με το DataFrame API.

Πίνακας 1: Χρόνοι εκτέλης των τριών υλοποιήσεων του Query 1

Method	Time (s)
DF-no-UDF	7.23
DF-UDF	8.53
RDD	16.9

2 Query 3

Σε αυτό το ερώτημα πέρα από τα δύο μεγάλα datasets για τα εγκλήματα, χρησιμοποιούμε και το dataset "MO Codes" για την λεκτική περιγραφή κάθε κωδικού εγκλήματος. Επομένως, είναι αναγκαίο να ενώσουμε (join) τα δύο datasets μας για το σωστό σχηματισμό του query. Μπορούμε να δούμε ότι το dataset των εγκλημάτων είναι πολύ μεγαλύτερο από το dataset των κωδικών. Από την θεωρία γνωρίζουμε ότι για τέτοιες περιπτώσεις, ο αλγόριθμος Broadcast Hash Join είναι ο κατάλληλος. Πράγματι, χρησιμοποιώντας την συνάρτηση *"explain"* στο τελικό dataframe object βλέπουμε ότι το Spark χρησιμοποιεί αυτόν τον αλγόριθμο χωρίς να του δώσουμε καμία εντολή ή παράμετρο επιπλεόν.

Αρχικά, μας ζητείται η σύγκριση δύο υλοποιήσεων - μία με DataFrame API και μία με RDD API. Όπως έχουμε ήδη αναφέρει, μόνο στην πρώτη θα αξιοποιηθεί ο Catalyst optimizer και αυτό φαίνεται και από τους χρόνους που λαμβάνουμε στον πίνακα 2. Η υλοποίηση με DataFrame είναι περίπου $4s$ πιο γρήγορη.

Πίνακας 2: Χρόνοι εκτέλεσης των δύο πρώτων υλοποιήσεων του Query 3

Method	Time (s)
DF	14.2
RDD	18.1

Στην συνέχεια μας ζητείται να χρησιμοποιήσουμε την συνάρτηση *"hint"* ώστε να ωθήσουμε το Spark να χρησιμοποιήσει διαφορετικούς αλγορίθμους για το join των δύο πινάκων. Έτσι, χρησιμοποιούμε την συνάρτηση καλώντας την από τον μικρό πίνακα και παίρνουμε τις μετρήσεις του πίνακα 3. Παρατηρούμε ότι οι χρόνοι εκτέλεσης μεταξύ των διαφορετικών στρατηγικών Join (SortMerge, ShuffleHash, Broadcast) παρουσιάζουν πολύ μικρές αποκλίσεις (εύρος περίπου $0.7s$). Αυτό το φαινομενικά παράδοξο αποτέλεσμα εξηγείται από τη φύση των δεδομένων μας.

Το dataset "MO Codes" είναι αρκετά μικρό σε μέγεθος. Ως αποτέλεσμα, το κόστος μεταφοράς του (network shuffle) ή ταξινόμησής του (sort) είναι αμελητέο συγκριτικά με τον χρόνο που απαιτείται για την ανάγνωση (I/O) και επεξεργασία του κύριου dataset των εγκλημάτων. Μπορούμε, λοιπόν, να θεωρήσουμε ότι το μεγαλύτερο χρονικό μέρος καταναλώνεται από το διάβασμα του μεγάλου πίνακα (CSV read operations) και όχι από τη διαδικασία του Join.

Στον πίνακα 3 αναγράφεται "CartesianProduct" η ονομασία του join όταν δώσαμε στην συνάρτηση *"hint"* την παράμετρο *"SHUFFLE_REPLICATE_NL"*.

Πίνακας 3: Χρόνοι εκτέλεσης των διαφορετικών join

Join	Time (s)
SortMergeJoin	14.9
ShuffleHashJoin	14.5
CartesianProduct	14.3