ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Προχωρημένα Θέματα Βάσεων Δεδομένων

Εξαμηνιαία Εργασία

Ειρήνη Κληρονόμου

9ο εξάμηνο

03116094

Μέρος 1ο

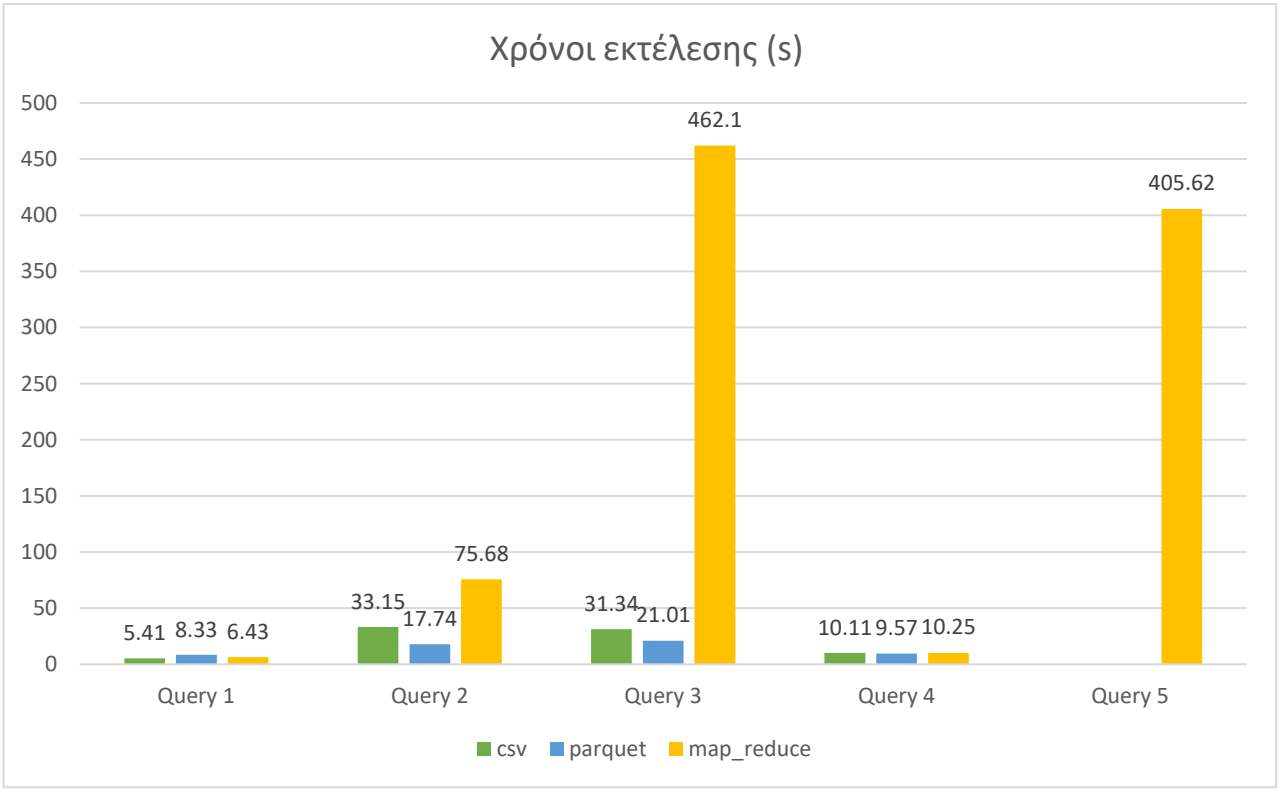
Αρχικά υλοποιήθηκαν τα ζητούμενα 1 και 2, στα οποία μεταφέραμε τα δοθέντα CSV αρχεία στο hdfs και στη συνέχεια τα μετατρέψαμε στη μορφή parquet (είναι αποθηκευμένα στο path "hdfs://master:9000/user/user").

Ζητούμενο 3ο

Στο ζητούμενο 3, μας ζητήθηκε να υλοποιήσουμε τα queries τόσο με RDD API όσο και με Spark SQL, με την δεύτερη υλοποίηση να μπορεί να διαβάζει είτε αρχεία CSV χρησιμοποιώντας το option inferSchema είτε αρχεία Parquet. Ως γλώσσα προγραμματισμού χρησιμοποιήθηκε η Python και για τις δύο υλοποιήσεις και η διεκπεραίωσή τους έγινε με βάση τους κώδικες που δόθηκαν στην εκφώνηση και το βοηθητικό υλικό.

Ζητούμενο 4ο

Παρακάτω παρατίθεται ένα ραβδόγραμμα το οποίο παρουσιάζει τους χρόνους εκτέλεσης σε seconds, ομαδοποιημένους ανά query.



- **Query 1:** Στο πρώτο query παρατηρούμε ότι η ταχύτερη υλοποίηση είναι εκείνη της Spark SQL που διαβάζει αρχείο CSV (χωρίς να έχει πολύ μεγάλη διαφορά με εκείνη του parquet αρχείου), ενώ η πιο αργή είναι εκείνη του Map Reduce (60% περισσότερος χρόνος περίπου).

- **Query 2:** Σε αντίθεση με πριν, εδώ πιο γρήγορη ήταν η εκτέλεση που διαβάζει parquet αρχείο της Spark SQL και μάλιστα με μεγάλη διαφορά σε σχέση με το CSV αρχείο (με το τελευταίο να εκτελείται για διπλάσιο χρόνο σε σχέση με το πρώτο). Και πάλι μεγαλύτερο χρόνο εκτέλεσης είχε το Map Reduce, με τον κώδικα να εκτελείται σχεδόν για 4πλάσιο χρόνο σε σχέση με την ταχύτερη υλοποίηση.

- **Query 3:** Όσον αφορά στην Spark SQL υλοποίηση, το query 3 εμφανίζει παρόμοια συμπεριφορά με εκείνη του query 2, δηλαδή η ταχύτερη υλοποίηση είναι εκείνη που διαβάζει parquet αρχείο και ακολουθεί αυτή με CSV με σχεδόν διπλάσιο χρόνο εκτέλεσης. Πιο χρονοβόρα ήταν και πάλι η Map Reduce υλοποίηση, σημειώνοντας πολύ σημαντική διαφορά σε σχέση με τις άλλες 2 υλοποιήσεις (σχεδόν 20 φορές περισσότερο χρόνο από εκείνη της parquet).

- **Query 4:** Στο 4^{ο} query παρατηρούμε ότι οι 3 υλοποιήσεις έχουν σχεδόν ίδιο χρόνο εκτέλεσης, με την υλοποίηση Spark SQL με parquet να είναι ελαφρώς πιο ταχεία και την Map Reduce ελαφρώς πιο αργή σε σχέση με τις άλλες 2.

- **Query 5:** Στο Query 5, δεν κατάφερα να πραγματοποιήσω την Spark υλοποίηση οπότε δεν μπορώ να εξάγω κάποια συμπεράσματα για αυτό.

Γενικά, σύμφωνα με τις μετρήσεις μας, παρατηρούμε ότι η Map Reduce υλοποίηση είναι στην πλειονότητα των περιπτώσεων εκείνη που απαιτεί και τον μεγαλύτερο χρόνο εκτέλεσης, άλλοτε με μεγαλύτερη και άλλοτε με μικρότερη διαφορά σε σχέση με τις άλλες δύο υλοποιήσεις. Το γεγονός αυτό είναι αναμενόμενο και έγκειται στη διαφορετική προσέγγιση επεξεργασίας των δεδομένων που έχουν η Map Reduce και η Spark. Στην πρώτη περίπτωση τα δεδομένα διαβάζονται και γράφονται σε δίσκο, ενώ στην δεύτερη τα δεδομένα φορτώνονται και επεξεργάζονται παράλληλα απευθείας στην μνήμη. Με αυτόν τον τρόπο στην Spark ορισμένες φορές, μία διεργασία μπορεί να χρησιμοποιηθεί ως είσοδος σε μία άλλη διεργασία, πράγμα που δεν συμβαίνει στην Map Reduce και γι' αυτόν τον λόγο μπορεί πολλές φορές να γίνει έως και 100 φορές πιο αργή η εκτέλεσή της.

Παράλληλα, βλέπουμε ότι με τη χρήση της parquet η εκτέλεση των προγραμμάτων μας γίνεται ταχύτερα. Η parquet είναι ευέλικτη ως προς την συμπίεση και πιο

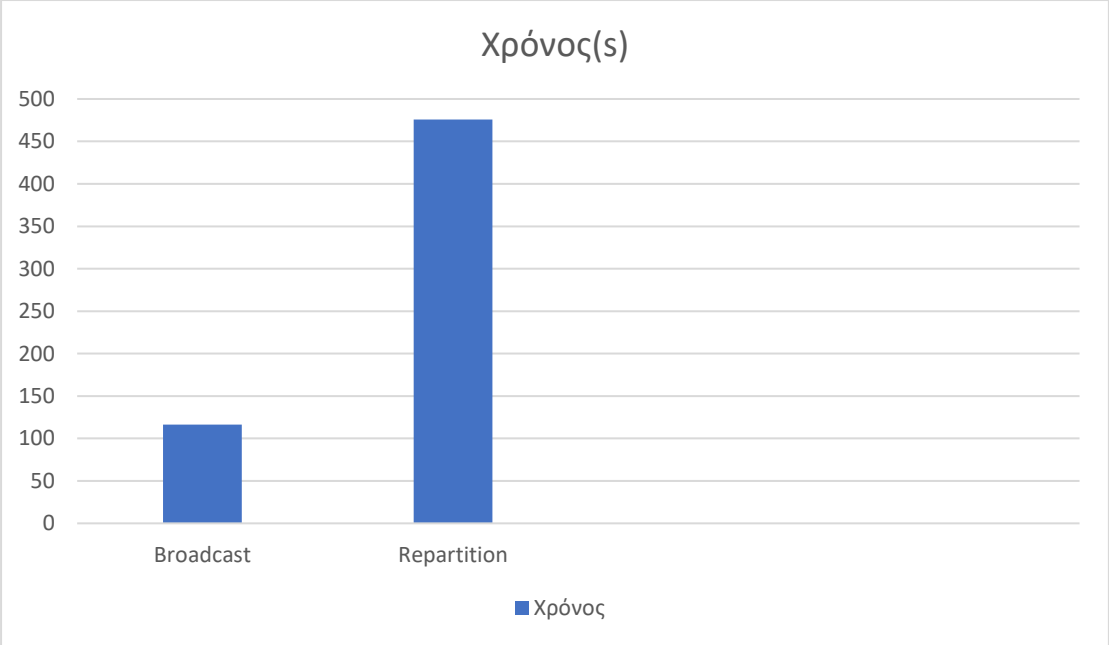
αποτελεσματική στα σχήματα κωδικοποίησης. Επομένως έχουμε παρόμοια τύπο δεδομένων σε κάθε στήλη, κάνοντας την συμπίεση τους πιο απλή και την εκτέλεση του query πιο γρήγορη.

Επιπροσθέτως, τα αρχεία parquet αποθηκεύουν το file schema στα metadata του αρχείου και για το λόγο αυτό δεν είναι απαραίτητη η χρήση του infer schema. Το infer schema υποθέτει αυτόματα τους τύπου δεδομένων για κάθε πεδίο και γι' αυτό πρέπει να χρησιμοποιείται στα CSV αρχεία.

Μέρος 2ο

Ζητούμενο 3ο

Στο παρακάτω ραβδόγραμμα παρουσιάζονται συγκριτικά οι χρόνοι εκτέλεσης των broadcast join και repartition join.



Όπως βλέπουμε παραπάνω το repartition join απαιτεί σχεδόν 4 φορές περισσότερο χρόνο εκτέλεσης σε σχέση με το broadcast join. Αυτό συμβαίνει για τους εξής λόγους:

Αρχικά, στο repartition γίνεται δυναμικός χωρισμός του πίνακα αναφοράς (L) και του πίνακα καταγραφής (R) στο κλειδί μέσω του οποίου γίνεται το join και ενώνονται τα αντίστοιχα ζεύγη των πινάκων. Κατά το στάδιο του mapping, κάθε διεργασία γίνεται είτε στο διαχωρισμό R είτε στο διαχωρισμό L. Προκειμένου να προσδιοριστεί ο πίνακας από τον οποίο προέρχεται η εκάστοτε εγγραφή εισόδου, κάθε διεργασία έχει ένα ζεύγος (key, value) όπου key το κλειδί σύνδεσης και value η εγγραφή. Στη συνέχεια τα αποτελέσματα θα συγχωνευτούν στο framework αφού έχουν πρώτα διαχωριστεί και ταξινομηθεί. Οι εγγραφές για κάθε join key θα

ομαδοποιηθούν και θα τροφοδοτούν έναν reducer, ο οποίος θα διαχωρίζει και θα αποθηκεύει κατάλληλα τις εγγραφές σύμφωνα με τον πίνακα αναφοράς τους.

Στην πλειονότητα των περιπτώσεων ο πίνακας R θα είναι πολύ μικρότερος από τον L.

Στο broadcast join, σε αντίθεση με το repartition, δεν μετακινούμε τους R,L σε όλο το δίκτυο αλλά μόνο τον R που είναι μικρότερος, με αποτέλεσμα να αποφεύγεται το κόστος της ταξινόμησης και της μετακίνησης του πίνακα L στο δίκτυο.

(Πηγή: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.644.9902&rep=rep1&type=pdf)

Ζητούμενο 4ο

Αφού ακολουθήσαμε τα βήματα της εκφώνησης και εκτελέσαμε τους κώδικες προέκυψε το παρακάτω ραβδόγραμμα στο οποίο παρουσιάζονται οι χρόνοι εκτέλεσης με και χωρίς βελτιστοποιητή.