Πανεπιστήμιο Μακεδονίας

Τμήμα Εφαρμοσμένης Πληροφορικής

Μάθημα: BIG DATA - Ανάλυση Δεδομένων Μεγάλου Όγκου

Ομαδική Εργασία 2

Θέμα: Apache Spark

Παρασκευή Ξανθοπούλου (it1490)
Κωνσταντίνος Τσεμπέρης (ics20071)

Καθηγητής: Αλέξανδρος Καρακασίδης
Ακαδημαϊκό Έτος: 2023-2024

Θεσσαλονίκη, Ιανουάριος 2024

Περιεχόμενα

1. Περιγραφή Υλοποίησης

1.1 Αλγόριθμος

1. Διάβασμα των δεδομένων από το HDFS

- Χρήση του sparkContext.wholeTextFiles() για τη φόρτωση των αρχείων.
- Διαχωρισμός των αρχείων σε δύο κατηγορίες, ανάλογα με το αν πρόκειται για αρχεία "original" ή "paraphrase".

2. Μετατροπή RDD σε DataFrame

Χρήση της μεθόδου toDF() για τη δημιουργία των δύο DataFrames (original και paraphrase) με ονόματα στηλών "filename" και "content".

3. Προσθήκη στήλης "type"

Χρήση της μεθόδου withColumn() για την προσθήκη της στήλης "type" με τιμή "original" ή "paraphrase".

4. Συγχώνευση των δύο DataFrames

Χρήση της μεθόδου union() για τη δημιουργία ενός DataFrame που περιλαμβάνει και τα δύο σύνολα δεδομένων.

5. Διαχωρισμός κειμένων σε λέξεις (Tokenization)

Χρήση του Tokenizer της PySpark για τη δημιουργία νέας στήλης "words" που περιέχει τις λέξεις από το κείμενο.

6. Αφαίρεση των Stop Words από τις λίστες λέξεων

Χρήση του StopWordsRemover για τη δημιουργία νέας στήλης "filtered_words" χωρίς τα stop words.

7. Υπολογισμός της Συχνότητας Εμφάνισης των Λέξεων

Χρήση του CountVectorizer για τη δημιουργία ενός νέου DataFrame με τη στήλη "features" που αναπαριστά τη συχνότητα των λέξεων.

8. Αφαίρεση μηδενικών εγγραφών

Για την αφαίρεση των εγγραφών με κενές λίστες λέξεων πραγματοποιήθηκε “φιλτράρισμα” του DataFrame, έτσι ώστε να αποφευχθεί η εφαρμογή της συνάρτησης MinHashLSH σε εγγραφές με κενό σύνολο λέξεων:

- Χρήση της συνάρτησης remove_empty_filtered_words().

9. Εφαρμογή του MinHashLSH

Χρήση του MinHashLSH της PySpark για τον εντοπισμό ομοιοτήτων μεταξύ των κειμένων και τη δημιουργία στήλης "hashes".

10. Εύρεση Ομοίων Ζευγών

Χρήση του approxSimilarityJoin με κατώφλι 0.8 και δημιουργία DataFrame με τα όμοια ζεύγη.

11. Φιλτράρισμα Εγγραφών (διατήρηση μόνο των επιθυμητών ζευγών)

Χρήση της συνάρτησης filter_rows().

12. Επιλογή και Εμφάνιση Αποτελεσμάτων με τις μικρότερες αποστάσεις

Χρήση των μεθόδων sort() και limit().

13. Εγγραφή Αποτελεσμάτων σε CSV

Χρήση της μεθόδου write.format("csv").mode("append").option("header", "true").save("/home/user/sparkout/").

1.2 Διαδικασία Εκτέλεσης

Η διαδικασία εκτέλεσης του προγράμματος περιλαμβάνει τα παρακάτω βήματα:

1. Δημιουργία script αρχείου: Δημιουργείται ένα αρχείο με το όνομα "script.py" χρησιμοποιώντας τον επεξεργαστή κειμένου nano. Αυτό το αρχείο περιλαμβάνει τον πηγαίο κώδικα του προγράμματος.

2. Εκτέλεση: Χρησιμοποιώντας το εργαλείο spark-submit, το πρόγραμμα εκτελείται στο περιβάλλον του Apache Spark. Η έξοδος του προγράμματος αποθηκεύεται σε ένα αρχείο κειμένου με το όνομα "spark_output.txt". Επιπλέον, οποιαδήποτε έξοδος σφαλμάτων καταγράφεται στο ίδιο αρχείο.

- Εντολή: ~/spark/bin/spark-submit --master spark://83.212.81.105:7077 script.py > spark_output.txt 2>&1

3. Παράμετροι Περιβάλλοντος Spark: Κατά την εκτέλεση του Spark, ορίζονται παράμετροι περιβάλλοντος με χρήση του αντικειμένου SparkConf. Αυτές οι παράμετροι περιλαμβάνουν τη μνήμη driver, τη μνήμη executor, τον αριθμό executors και τους πυρήνες executor.

- Εντολή: conf = SparkConf().setAppName("DocumentSimilarity") \
  .set("spark.driver.memory", "2000M").set("spark.executor.memory", "6000M") \
  .set("spark.executor.instances", "4").set("spark.executor.cores", "4")

4. Εκτέλεση σε Έναν Κόμβο: Για να εκτελεστεί το πρόγραμμα σε έναν μόνο κόμβο, χρησιμοποιήθηκε το εργαλείο jps για τον εντοπισμό του αριθμού του Spark Worker και στη συνέχεια η εντολή kill για να σταματήσει.

- Εντολή: jps
- kill <αριθμός_worker>

2. Προβλήματα και Τρόποι Επίλυσης

Αρχικά το PySpark δεν αναγνώριζε την εντολή "**from pyspark.ml.feature import MinHashLSH**" η οποία είναι υπεύθυνη για την συμπερίληψη της βιβλιοθήκης MinHashLSH στο πρόγραμμα. Για την αναγνώριση της βιβλιοθήκης ήταν απαραίτητη η εγκατάσταση του numpy και του pip της python, με τις εντολές:

1. **sudo apt-get install python3-pip**
2. **pip3 install numpy**

Κατά την εκτέλεση της εντολής για τη μετατροπή ενός Resilient Distributed Dataset (RDD) σε ένα DataFrame, αντιμετωπίσαμε το παρακάτω σφάλμα:

WARN BlockReaderFactory: I/O error constructing remote block reader.

java.io.InterruptedIOException: Interrupted while waiting for IO on channel java.nio.channels.SocketChannel[connection-pending

Αυτό το μήνυμα προέκυψε κατά τη διάρκεια της προσπάθειας σύνδεσης με το HDFS για την ανάγνωση των δεδομένων και ενδέχεται να οφείλεται σε προβλήματα συνδεσιμότητας με το HDFS, ενδεχομένως λόγω υπερφόρτωσης του συστήματος ή δικτυακών προβλημάτων. Για την αντιμετώπιση του, πραγματοποιήθηκε επανάληψη της διαδικασίας μετατροπής.

3. Χρονική Αξιολόγηση

3.1 Αποτελέσματα Μετρήσεων

Παρακάτω παρουσιάζονται οι μέσοι χρόνοι εκτέλεσης για διάφορες παραμετροποιήσεις του προγράμματος:

	1w_2ht	1w_5ht	1w_10ht	2w_2ht	2w_5ht	2w_10ht
It 1	2,2	2,6	4,1	2,8	3,2	3,7
It 2	2,0	2,7	3,5	2,5	3,1	4
It 3	2,0	2,8	3,4	2,6	3,3	4,5
Μέσος Χρόνος	**2,07**	**2,7**	**3,67**	**2,63**	**3,20**	**4,07**

3.2 Σχολιασμός

Κατά την αξιολόγηση των αποτελεσμάτων, παρατηρούμε τα εξής:

- Το πλήθος των HashTables φαίνεται να επηρεάζει σημαντικά τη διάρκεια εκτέλεσης. Παρατηρούμε ότι οι μέσοι χρόνοι διαφέρουν ανάλογα με τις τιμές του αριθμού των HashTables, για μεγαλύτερο πλήθος HashTables έχουμε μεγαλύτερες διάρκειες.

- Όσον αφορά στον αριθμό των workers, παρατηρείται ότι οι διάρκειες εκτέλεσης με έναν worker είναι μικρότερες σε σύγκριση με τις περιπτώσεις με δύο workers. Αυτό ενδέχεται να οφείλεται στο overhead που προκύπτει από τον συντονισμό των εργασιών σε περιβάλλον πολλαπλών workers.

Συνολικά, οι αναλύσεις των μετρήσεων και οι παρατηρήσεις παρέχουν μια σαφή εικόνα της απόδοσης του προγράμματος για διάφορες παραμετροποιήσεις. Αυτό επιτρέπει την εύρεση βέλτιστων παραμετρικών επιλογών για την επίτευξη βέλτιστων επιδόσεων κατά την εκτέλεση του προγράμματος.