ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Σύστημα δεικτοδότησης με δυνατότητα ενημερώσεων για δεδομένα μεγάλης κλίμακας, χρησιμοποιώντας κατανεμημένες τεχνικές επεξεργασίας (MapReduce και NoSQL)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Παναγιώτης Χ. Αντωνόπουλος

Επιβλέπων : Νεκτάριος Κοζύρης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2011

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Σύστημα δεικτοδότησης με δυνατότητα ενημερώσεων για δεδομένα μεγάλης κλίμακας, χρησιμοποιώντας κατανεμημένες τεχνικές επεξεργασίας (MapReduce και NoSQL)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Παναγιώτης Χ. Αντωνόπουλος

Επιβλέπων : Νεκτάριος Κοζύρης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 18η Ιουλίου 2011.

.....	.....	.....
Νεκτάριος Κοζύρης Αναπληρωτής Καθηγητής Ε.Μ.Π.	Τιμολέων Σελλής Καθηγητής Ε.Μ.Π.	Νικόλαος Παπασπύρου Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2011

.....
Παναγιώτης Χ. Αντωνόπουλος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Με τη χρήση κατανεμημένων τεχνικών επεξεργασίας, έχουμε τη δυνατότητα να μειώσουμε σημαντικά το χρόνο δημιουργίας και ενημέρωσης των ευρετηρίων που αφορούν δεδομένα μεγάλης κλίμακας, όπως για παράδειγμα αυτά που είναι διαθέσιμα στο διαδίκτυο, εκμεταλλευόμενοι τις δυνατότητες που μας προσφέρουν οι σύγχρονες αρχιτεκτονικές υπολογιστών, όπως οι shared-nothing αρχιτεκτονικές και το Cloud. Ταυτόχρονα, με την κατανεμημένη αποθήκευση και διαχείριση των ευρετηρίων, καθίσταται εφικτή η αντιμετώπιση του αυξημένου φόρτου ερωτημάτων που έχει προκύψει ως αποτέλεσμα της εκρηκτικής αύξησης του αριθμού των χρηστών του διαδικτύου.

Στην διπλωματική αυτή εργασία παρουσιάζεται μια κατανεμημένη αρχιτεκτονική για τη δημιουργία και την ενημέρωση ανεστραμμένων ευρετηρίων (inverted index) για συλλογές κειμένων μεγάλης κλίμακας. Αναλυτικότερα, παρουσιάζεται μια μεθοδολογία για την κατανεμημένη δημιουργία και, στη συνέχεια, την κατανεμημένη ενημέρωση ανεστραμμένων ευρετηρίων, η οποία καθιστά δυνατή την ενημέρωση ενός υπάρχοντος ευρετηρίου σε χρόνο πρακτικά ανεξάρτητο από το μέγεθος του, αξιοποιώντας τα ιδιαίτερα χαρακτηριστικά των NoSQL βάσεων δεδομένων. Ακόμα, προτείνεται ένας αλγόριθμος σύγκρισης μεταξύ των παλαιών και νέων εκδόσεων των τροποποιημένων κειμένων, ο οποίος ελαχιστοποιεί τις τροποποιήσεις που πρέπει να πραγματοποιηθούν στο ευρετήριο, επιταχύνοντας έτσι σημαντικά τη διαδικασία ενημέρωσης. Με τη χρήση των μεθόδων αυτών, καθίσταται εφικτή η ταχύτερη και συχνότερη ενημέρωση ανεστραμμένων ευρετηρίων που έχουν δημιουργηθεί από μεγάλες συλλογές κειμένων, με στόχο την επιστροφή ενημερωμένων αποτελεσμάτων στους τελικούς χρήστες.

Για την αποδοτικότερη εκτέλεση των διαδικασιών δημιουργίας και ενημέρωσης του ευρετηρίου, προτείνεται η χρήση του Hadoop MapReduce, το οποίο αποτελεί μια υλοποίηση ανοικτού λογισμικού του MapReduce framework και είναι κατάλληλο για την κατανεμημένη επεξεργασία μεγάλου όγκου δεδομένων. Επιπρόσθετα, για την ταχύτερη επεξεργασία του μεγάλου φόρτου ερωτημάτων των χρηστών, προτείνεται η αποθήκευση του ευρετηρίου στην HBase, η οποία αποτελεί μια κατανεμημένη, NoSQL βάση δεδομένων που καθιστά δυνατή την αποθήκευση μεγάλου όγκου δεδομένων και την κατανομή του φορτίου ερωτημάτων στους κόμβους του συστήματος.

Λέξεις Κλειδιά: ανεστραμμένο ευρετήριο, κατανεμημένη, δημιουργία, ενημέρωση, αποθήκευση, νεφελώδης υπολογισμός, Hadoop, MapReduce, HBase, NoSQL

Abstract

By using distributed processing techniques, we can significantly reduce the time needed for the creation and update of indexes on large-scale datasets, such as those available on the Internet, taking advantage of the capabilities of modern architectures, such as shared-nothing architectures and the Cloud. At the same time, the distributed storage and management of such indexes allows us to process the increased query workload, which has arisen as a result of the explosive growth in the number of Internet users.

In this thesis, we present a distributed architecture for creating and updating inverted indexes for large-scale document collections. In more detail, we present a methodology for the distributed creation and update of inverted indexes, which allows us to update an existing inverted index in time practically independent of its size, utilizing the characteristics of NoSQL databases. Moreover, we propose an algorithm which compares the old and new versions of the modified documents in order to minimize the changes needed in the index, and therefore accelerate the update process. By using these methods, it becomes possible to quickly and frequently update existing inverted indexes, created from large document collections, in order to provide fresh results to the users.

For the efficient execution of these processes, we suggest the usage of Hadoop MapReduce, which is an open-source implementation of the MapReduce framework and is suitable for the distributed processing of large-scale datasets. Furthermore, in order to speed up the processing of users' queries, we recommend the storage of the index in HBase, a distributed, NoSQL database which allows the storage of large volumes of data and the distribution of the query workload to the nodes of the cluster.

Keywords: inverted index, distributed, creation, update, storage, cloud computing, Hadoop, MapReduce, HBase, NoSQL

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή της εργασίας μου και Αναπληρωτή Καθηγητή Ε.Μ.Π. κ. Νεκτάριο Κοζύρη, για την βοήθεια και την υποστήριξη του καθ' όλη τη διάρκεια της εργασίας. Επίσης, θα ήθελα να ευχαριστήσω τους ερευνητές του εργαστηρίου Υπολογιστικών Συστημάτων του Ε.Μ.Π. και ειδικότερα τους Ιωάννη Κωνσταντίνου και Δημήτριο Τσουμάκο, για την αμέριστη βοήθεια και τις χρήσιμες συμβουλές τους, οι οποίες διευκόλυναν σημαντικά την εκπόνηση της εργασίας αυτής. Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου για την υποστήριξή τους σε κάθε μου επιλογή όλα αυτά τα χρόνια.

Πίνακας Περιεχομένων

Πίνακας σχημάτων

Πίνακας αλγορίθμων

Κεφάλαιο 1
Εισαγωγή

1.1 Το ανεστραμμένο ευρετήριο (Inverted Index)

Με την εξαιρετική ανάπτυξη του διαδικτύου, ο όγκος των δεδομένων που είναι διαθέσιμα σε κάθε χρήστη έχει φτάσει σε πρωτόγνωρα μεγέθη. Προκειμένου, όμως, κάθε χρήστης να έχει πρόσβαση στα δεδομένα που τον ενδιαφέρουν, απορρίπτοντας τη μη σχετική πληροφορία, είναι απαραίτητο να υπάρχει ένα είδος αναζήτησης με βάση το περιεχόμενο των δεδομένων, ώστε κάθε χρήστης να έχει τη δυνατότητα να καθορήσει τις προτιμήσεις του και να αναζητήσει τα δεδομένα που ταιριάζουν σε αυτές.

Τα κείμενα αποτελούν ίσως τη σημαντικότερη πηγή πληροφορίας στο διαδίκτυο, γεγονός που καθιστά απαραίτητη την ύπαρξη ενός είδους αναζήτησης με βάση το περιεχόμενο τους. Δεν είναι εφικτό, ωστόσο, να γίνεται σάρωση του περιεχομένου ενός τεράστιου πλήθους κειμένων για κάθε αναζήτηση των χρηστών. Για το λόγο αυτό χρησιμοποιείται μια ειδική δομή δεδομένων, με το όνομα ‘ανεστραμμένο ευρετήριο’ (Inverted Index)[1]. Τα ευρετήρια αυτού του είδους αποθηκεύουν μια αντιστοίχηση μεταξύ κάθε λέξης που περιέχεται σε μία συλλογή κειμένων και των κειμένων που περιέχουν τη λέξη αυτή. Σε ορισμένες περιπτώσεις, αποθηκεύεται και η θέση της λέξης μέσα στο αντίστοιχο κείμενο, ώστε να είναι δυνατή η καλύτερη παρουσίαση των αποτελεσμάτων στο χρήστη. Με τον τρόπο αυτό υπάρχει πλέον η δυνατότητα να πραγματοποιούμε αναζητήσεις με βάση το περιεχόμενο των κειμένων με εξαιρετικά μικρότερο κόστος. Το ανεστραμμένο ευρετήριο αποτελεί την πιο συνηθισμένη δομή δεδομένων για την αναζήτηση κειμένων και χρησιμοποιείται σε μεγάλης κλίμακας συστήματα, όπως οι μηχανές αναζήτησης.

Στο παρακάτω σχήμα παρουσιάζεται ένα παράδειγμα ενός ανεστραμμένου ευρετηρίου:

Λέξη	**Λίστα κείμενων της συλλογής**
distributed	Doc2, Doc3, Doc7, Doc10
indexing	Doc2, Doc7, Doc12
using	Doc3, Doc4, Doc5, Doc6, Doc7, Doc8
Hadoop	Doc2, Doc7, Doc10
HBase	Doc7
MapReduce	Doc2, Doc7, Doc10

Σχήμα 1: Παράδειγμα ανεστραμμένου ευρετηρίου

Όπως φαίνεται το ανεστραμμένο ευρετήριο είναι ουσιαστικά μια αντιστοίχηση μεταξύ των λέξεων μιας συλλογής κειμένων (πρώτη στήλη πίνακα) και μιας λίστας από αναφορές στα κείμενα της συλλογής που περιλαμβάνουν τη συγκεκριμένη λέξη (δεύτερη στήλη πίνακα).

Όταν, λοιπόν, ένας χρήστης αναζητήσει τα κείμενα μιας συλλογής που περιέχουν μια συγκεκριμένη λέξη, το σύστημα χρησιμοποιεί το ευρετήριο προκειμένου να εντοπίσει τη λίστα με τις αναφορές στα κείμενα που περιέχουν τη λέξη αυτή και στη συνέχεια ανακτά τα αντίστοιχα κείμενα και τα επιστρέφει στο χρήστη. Αν ο χρήστης επιθυμεί την ύπαρξη περισσοτέρων λέξεων σε ένα κείμενο, τότε του επιστρέφονται τα κείμενα που ανήκουν στην τομή των λιστών των λέξεις που αυτός όρισε, ενώ αν επιθυμεί την ύπαρξη τουλάχιστον μίας λέξης από ένα μεγαλύτερο σύνολο, τότε του επιστρέφεται η ένωση των αντίστοιχων λιστών.

Παραδείγματα:

Λέξεις προς αναζήτηση	**Λίστα αποτελεσμάτων**
distributed	Doc2, Doc3, Doc7, Doc10
distributed AND indexing	Doc2, Doc7
distributed OR indexing	Doc2, Doc3, Doc7, Doc10, Doc12

Σχήμα 2: Παράδειγμα αποτελεσμάτων αναζήτησης σε ένα ανεστραμμένο ευρετήριο

1.2 Διαδικασία κατασκευής ενός ανεστραμμένου ευρετηρίου

Η κατασκευή ενός ανεστραμμένου ευρετηρίου, ειδικά όταν αυτό δεν χρησιμοποιεί κάποιον αλγόριθμο αξιολόγησης των κειμένων με σκοπό την κατάταξη των αποτελεσμάτων, αποτελεί μια σχετικά απλή διαδικασία, όσον αφορά το αλγοριθμικό της μέρος, ενώ η υπολογιστική της πολυπλοκότητα είναι χαμηλή.

Συγκεκριμένα, τα κείμενα της συλλογής σαρώνονται το ένα μετά το άλλο ώστε να αναγνωριστούν οι λέξεις που προβλέπεται να προστεθούν στο ευρετήριο, ενώ για κάθε νέα λέξη που αναγνωρίζεται σε ένα κείμενο προσθέτουμε μια αναφορά του κειμένου στην λίστα της αντίστοιχης λέξης.

Παρακάτω παρουσιάζεται ο αλγόριθμος κατασκευής ενός ανεστραμμένου ευρετηρίου με σειριακό τρόπο:

Αλγόριθμος 1: Αλγόριθμος σειριακής δημιουργίας ενός ανεστραμμένου ευρετηρίου

```
Για κάθε κείμενο που ανήκει στη συλλογή:
    Για κάθε λέξη του κειμένου:
        Αν η λέξη δεν έχει ξαναβρεθεί στο συγκεκριμένο κείμενο:
            Αν υπάρχει λίστα για τη συγκεκριμένη λέξη:
                Πρόσθεσε μια αναφορά του κειμένου στη λίστα;
            Αλλιώς:
                Δημιούργησε μια κενή λίστα για τη λέξη αυτή;
                Πρόσθεσε μια αναφορά του κειμένου στη λίστα;
```

Όπως φαίνεται η δημιουργία του ευρετηρίου είναι αλγοριθμικά απλή, ωστόσο ο τεράστιος αριθμός των κειμένων στο διαδίκτυο την καθιστά ιδιαίτερα χρονοβόρα. Παρότι ο όγκος του κάθε αρχείου κειμένου είναι μικρός σε σχέση με τα αρχεία πολυμέσων, για παράδειγμα, το εξαιρετικά μεγάλο και ταχύτατα αυξανόμενο πλήθος τους καθιστά την επεξεργασία τους μια απαιτητική διαδικασία.

1.3 Ανάγκη για κατανεμημένη δημιουργία των ανεστραμμένων ευρετηρίων και κατανεμημένη επεξεργασία των ερωτημάτων – Αξιοποίηση του Cloud

Όπως αναφέρθηκε και προηγουμένως, το πρόβλημα που καλούμαστε να αντιμετωπίσουμε κατά τη δημιουργία ενός ανεστραμμένου ευρετηρίου δεν είναι η πολυπλοκότητα της επεξεργασίας, αλλά ο τεράστιος όγκος των δεδομένων. Ταυτόχρονα τα δεδομένα-κείμενα είναι ανεξάρτητα μεταξύ τους, με αποτέλεσμα η σάρωση του κάθε κειμένου να μπορεί να γίνει ξεχωριστά και να είναι αναγκαία μόνο οι τελική συνένωση των επιμέρους λιστών των λέξεων. Αυτό μας δίνει τη δυνατότητα να επεξεργαστούμε τα δεδομένα με κατανεμημένο τρόπο, χωρίς να απαιτείται επικοινωνία μεταξύ των κόμβων που εκτελούν την επεξεργασία παρά μόνο για την τελική ένωση των λιστών. Έτσι, μπορούμε να εκμεταλλευτούμε στο έπακρο τις δυνατότητες των shared-nothing αρχιτεκτονικών, αλλά και του Cloud όπου μπορούμε να έχουμε ελαστική ανάθεση των πόρων ανάλογα με τις ανάγκες που επιβάλλει η συλλογή κειμένων που καλούμαστε να επεξεργαστούμε.

Αντίστοιχα, ο τεράστιος όγκος των δεδομένων και ο ιδιαίτερα αυξημένος αριθμός χρηστών και ερωτημάτων καθιστά την ανάκτηση των αποτελεσμάτων μιας αναζήτησης μια απαιτητική διαδικασία. Η κατανεμημένη αποθήκευση του ευρετηρίου και η χρήση αντιγράφων (replication) σε διαφορετικούς κόμβους επιτρέπει τη κατανομή του φόρτου των ερωτημάτων σε περισσότερα μηχανήματα-κόμβους, με αποτέλεσμα να αυξάνεται σημαντικά ο συνολικός ρυθμός ανάκτησης των αποτελεσμάτων. Ακόμα, η κατανεμημένη αποθήκευση αυξάνει την ευρωστία του συστήματος, το οποίο, σε αντίθεση με ένα κεντρικό σύστημα ενός κόμβου, μπορεί να αντιμετωπίσει τυχόν προβλήματα σε κάποιον κόμβο χωρίς να μειωθεί η διαθεσιμότητα του συστήματος. Η ελαστικότητα στην ανάθεση των πόρων στο Cloud μπορεί να φανεί εξαιρετικά χρήσιμη και όσον αφορά την επεξεργασία των ερωτημάτων των χρηστών. Σε χρονικές στιγμές όπου ο αριθμός των ερωτημάτων είναι αυξημένος, το σύστημα θα μπορούσε να ζητήσει περισσότερους πόρους και με τον τρόπο αυτό να αντιμετωπίσει το αυξημένο φορτίο με επιτυχία.

1.4 Ανάγκη για αποδοτική ενημέρωση των ευρετηρίων

Η εκτεταμένη χρήση του διαδικτύου και, ειδικότερα, η συμμετοχή κάθε απλού χρήστη στην διαμόρφωση του περιεχομένου των σελίδων, όπως για παράδειγμα σε sites κοινωνικής δικτύωσης, όπως το Facebook ή το Twitter, αλλά και σε εγκυκλοπαίδειες που επιτρέπουν τη συμμετοχή των χρηστών για την διαμόρφωση του περιεχομένου τους, όπως η Wikipedia, έχουν οδηγήσει σε έναν εξαιρετικά υψηλό ρυθμό ανανέωσης του περιεχομένου του διαδικτύου. Καθημερινά ένας τεράστιος αριθμός σελίδων προστίθεται ή τροποποιείται, με αποτέλεσμα ακόμα και τα ευρετήρια που δημιουργήθηκαν πριν από πολύ μικρό χρονικό διάστημα να είναι εντελώς ανενημέρωτα και άρα σχεδόν άχρηστα για τους χρήστες. Προκύπτει, λοιπόν, η ανάγκη για την συνεχή ενημέρωση ή αντικατάσταση των ευρετηρίων, που χρησιμοποιούνται για την αναζήτηση ιστοσελίδων και κειμένων, ώστε να περιέχουν ενημερωμένα δεδομένα, τα οποία είναι χρήσιμα για τους χρήστες.

Όσο ο αριθμός, αλλά και ο ρυθμός ανανέωσης των ιστοσελίδων και των κειμένων ήταν μικρός, η ανανέωση του περιεχομένου των ευρετηρίων γινόταν με την δημιουργία ενός νέου ευρετηρίου, εξ ολοκλήρου από την αρχή, και την αντικατάσταση του παλαιότερου ευρετηρίου ανά τακτά χρονικά διαστήματα. Ωστόσο, με την εκρηκτική αύξηση του αριθμού, καθώς και του ρυθμού ανανέωσης των ιστοσελίδων, που παρατηρείται τα τελευταία χρόνια, κάτι τέτοιο δεν είναι πλέον αποδοτικό. Είναι, λοιπόν απαραίτητο να

βρεθούν τεχνικές για την αποδοτική ενημέρωση των υπαρχόντων ευρετηρίων, ώστε αυτά να συμπεριλαμβάνουν τις ενημερωμένες και νέες ιστοσελίδες, χωρίς να απαιτείται η εξαιρετικά χρονοβόρα δημιουργία νέων. Όλες οι μεγάλες μηχανές αναζήτησης έχουν στρέψει πλέον την προσοχή τους στην ανεύρεση τέτοιων τεχνικών, ώστε να μπορούν να προσφέρουν στους χρήστες τους όσο το δυνατόν πιο ενημερωμένα δεδομένα. Με την αξιοποίηση των τεχνικών κατανεμημένης επεξεργασίας, που έχουν αναπτυχθεί τα τελευταία χρόνια, καθώς και των σύγχρονων αρχιτεκτονικών υπολογιστών, υπάρχει η δυνατότητα να επιταχυνθεί σημαντικά η διαδικασία ενημέρωσης των ευρετηρίων, ώστε αυτά να παραμένουν όσο το δυνατόν περισσότερο ενημερωμένα.

1.5 Αντικείμενο της διπλωματικής εργασίας

Σκοπός της παρούσας διπλωματικής εργασίας είναι η σχεδίαση μιας κατανεμημένης αρχιτεκτονικής για την δημιουργία, την αποθήκευση και την ενημέρωση ανεστραμμένων ευρετηρίων που έχουν δημιουργηθεί από συλλογές κειμένων μεγάλης κλίμακας, όπως για παράδειγμα αυτές που είναι διαθέσιμες στο διαδίκτυο. Αναγνωρίζοντας τις δυσκολίες που προκύπτουν από την εκρηκτική αύξηση του αριθμού χρηστών, του όγκου των δεδομένων, καθώς και του ρυθμού ανανέωσης τους, επιχειρούμε να σχεδιάσουμε ένα σύστημα το οποίο να ανταποκρίνεται στις σύγχρονες απαιτήσεις.

Με τη χρήση κατανεμημένων τεχνικών επεξεργασίας, έχουμε τη δυνατότητα να μειώσουμε σημαντικά το χρόνο δημιουργίας και ενημέρωσης των ανεστραμμένων ευρετηρίων, κατανέμοντας το φορτίο επεξεργασίας στους κόμβους του συστήματος. Για το σκοπό αυτό, προτείνεται η χρήση του Hadoop MapReduce, το οποίο αποτελεί μια υλοποίηση ανοιχτού λογισμικού του MapReduce framework και έχει σχεδιαστεί με στόχο την κατανεμημένη επεξεργασία μεγάλου όγκου δεδομένων. Ταυτόχρονα, με την κατανεμημένη αποθήκευση και διαχείριση των ευρετηρίων, καθίσταται εφικτή η ταχύτερη επεξεργασία του αυξημένου φορτίου ερωτημάτων των χρηστών, το οποίο είναι δύσκολο να αντιμετωπιστεί από ένα κεντρικό σύστημα. Για το σκοπό αυτό, προτείνεται η χρήση της HBase, μιας κατανεμημένης, NoSQL βάσης δεδομένων στην οποία αποθηκεύονται, τόσο τα ίδια τα κείμενα της συλλογής, όσο και το ευρετήριο που δημιουργείται.

Για την αποδοτικότερη ενημέρωση των υπαρχόντων ευρετηρίων, η οποία είναι εξαιρετικά σημαντική, δεδομένου του ρυθμού ανανέωσης των δεδομένων στο διαδίκτυο,

παρουσιάζεται μια κατανεμημένη διαδικασία ενημέρωσης, της οποίας η υπολογιστική πολυπλοκότητα, αλλά και ο χρόνος εκτέλεσης, είναι πρακτικά ανεξάρτητοι από το μέγεθος του υπάρχοντος ευρετηρίου. Προκειμένου να επιτευχθεί αυτό, προτείνονται ορισμένες τροποποιήσεις στη διαδικασία αρχικής δημιουργίας του ευρετηρίου, αλλά και ένα σχήμα αποθήκευσης του ευρετηρίου, το οποίο βασίζεται στα ιδιαίτερα χαρακτηριστικά των NoSQL βάσεων δεδομένων. Παράλληλα, για την επιτάχυνση της διαδικασίας ενημέρωσης παρουσιάζεται ένας αλγόριθμος σύγκρισης μεταξύ των παλαιών και νέων εκδόσεων των τροποποιημένων κειμένων, ο οποίος ελαχιστοποιεί τον αριθμό των αλλαγών που πρέπει να πραγματοποιηθούν στο ευρετήριο. Με τον τρόπο αυτό, καθίσταται εφικτή η συχνότερη και ταχύτερη ενημέρωση ανεστραμμένων ευρετηρίων μεγάλων συλλογών κειμένων, ώστε οι χρήστες να έχουν πρόσβαση στα ενημερωμένα δεδομένα.

Πέρα από τη θεωρητική τεκμηρίωση της λειτουργίας και της πολυπλοκότητας των διαδικασιών, παρουσιάζεται η υλοποίηση του συστήματος δημιουργίας και ενημέρωσης ανεστραμμένων ευρετηρίων, με τη χρήση των κατανεμημένων τεχνολογιών που αναφέρθηκαν προηγουμένως. Τα πειραματικά αποτελέσματα, που προκύπτουν από τη χρήση του συστήματος, επιβεβαιώνουν την θεωρητική πολυπλοκότητα της διαδικασίας ενημέρωσης και αναδεικνύουν τη δυνατότητα της να επιταχύνει σημαντικά την ενημέρωση του ευρετηρίου.

1.6 Οργάνωση κειμένου

Στο πρώτο κεφάλαιο έγινε μια γενική εισαγωγή στο πρόβλημα που καλούμαστε να αντιμετωπίσουμε και παρουσιάστηκε το αντικείμενο της παρούσας διπλωματικής εργασίας. Στο δεύτερο κεφάλαιο παρουσιάζονται εργασίες σχετικές με τη κατανεμημένη δημιουργία, αποθήκευση και ενημέρωση ανεστραμμένων ευρετηρίων. Το τρίτο κεφάλαιο αναφέρεται στην αρχιτεκτονική, τα χαρακτηριστικά και την καταλληλότητα των υποσυστημάτων που χρησιμοποιήθηκαν για την ανάπτυξη του συστήματος που εξετάζουμε. Στο τέταρτο κεφάλαιο γίνεται μια εκτενής περιγραφή των κατανεμημένων διαδικασιών αρχικής δημιουργίας και ενημέρωσης, που προτείνονται από την παρούσα διπλωματική εργασία, ενώ ταυτόχρονα παρουσιάζονται τα ιδιαίτερα χαρακτηριστικά τους, τα οποία καθιστούν την ενημέρωση των ευρετηρίων ιδιαίτερα αποδοτική. Στο πέμπτο κεφάλαιο παρουσιάζονται τα πειραματικά αποτελέσματα από τη χρήση του συστήματος και αποτιμάται το κατά πόσο αυτά συμφωνούν με τη θεωρητική ανάλυση που έχει προηγηθεί. Τέλος, στο έκτο κεφάλαιο,

παρουσιάζονται χρήσιμα συμπεράσματα που αφορούν τη λειτουργία και την απόδοση του συστήματος, καθώς και τα βασικά χαρακτηριστικά των διαδικασιών δημιουργίας και ενημέρωσης που παρουσιάστηκαν, ενώ προτείνονται και ορισμένες πιθανές μελλοντικές επεκτάσεις της συγκεκριμένης έρευνας.

Κεφάλαιο 2
Σχετικές εργασίες

Από τα τέλη της δεκαετίας του ‘90, όπου το διαδίκτυο είχε αρχίσει να παίρνει μεγάλη έκταση και ο όγκος των διαθέσιμων δεδομένων αυξανόταν με υψηλούς ρυθμούς, είχε εμφανιστεί η ανάγκη για την ταχύτερη δημιουργία ευρετηρίων με σκοπό την αποτελεσματικότερη αναζήτηση στο διαδίκτυο. Ταυτόχρονα, η αύξηση του αριθμού των χρηστών οδήγησε σε σημαντική αύξηση του φόρτου ερωτημάτων, με αποτέλεσμα να είναι απαραίτητη η δημιουργία συστημάτων που θα μπορούσαν να ανταπεξέλθουν στο αυξημένο αυτό φορτίο. Κατανοώντας τους περιορισμούς από τη χρήση κεντρικών συστημάτων, η έρευνα στράφηκε στην χρήση κατανεμημένων αρχιτεκτονικών για τη δημιουργία και τη διαχείριση ευρετηρίων για μεγάλες συλλογές κειμένων ή ιστοσελίδων[2]. Στην επόμενη δεκαετία, η έρευνα πάνω στην κατανεμημένη δημιουργία ευρετηρίων έγινε ακόμα εντατικότερη και το 2004 παρουσιάστηκε το προγραμματιστικό μοντέλο MapReduce [3], το οποίο είναι κατάλληλο για την επεξεργασία μεγάλων datasets, με τη χρήση ενός κατανεμημένου συστήματος πολλών κόμβων, και αναπτύχθηκε από τη Google με σκοπό την αποδοτικότερη δημιουργία ευρετηρίων για το διαδίκτυο.

Ωστόσο, η αυξανόμενη συμμετοχή των απλών χρηστών στη διαμόρφωση των δεδομένων του διαδικτύου οδήγησε στην αύξηση του ρυθμού δημιουργίας νέων δεδομένων, αλλά και στη συνεχή τροποποίηση των υπαρχόντων δεδομένων, τα οποία στο παρελθόν ήταν σε γενικές γραμμές στατικά. Έτσι, όλες οι μεγάλες μηχανές αναζήτησης στράφηκαν στην σχεδίαση συστημάτων με σκοπό την αποδοτική ενημέρωση των ευρετηρίων τους, αφού πλέον η περιοδική επαναδημιουργία του ευρετηρίου δεν ήταν επαρκής για να ικανοποιήσει την ανάγκη των χρηστών για ‘φρέσκα’ δεδομένα. Το 2010 η Google ανακοίνωσε την ολοκλήρωση ενός νέου συστήματος δεικτοδότησης με το όνομα Google Caffeine[4], το οποίο έχει τη δυνατότητα να παρέχει στους χρήστες ενημερωμένα δεδομένα, ενημερώνοντας σταδιακά το υπάρχον ευρετήριο της εταιρίας. Στην ίδια κατεύθυνση κινήθηκε και το Twitter, το οποίο δημιούργησε μία πλατφόρμα αναζήτησης, ονόματι ‘Twitter Search’[5], με βάση το Apache Lucene[6], με σκοπό να προσφέρει στους χρήστες τους τις τελευταίες ενημερώσεις σε όσο το δυνατόν μικρότερο χρονικό διάστημα.

Από την κοινότητα ελεύθερου λογισμικού έχουν αναπτυχθεί ορισμένες εφαρμογές για την κατανεμημένη δημιουργία ευρετηρίων, αλλά κυρίως για την κατανεμημένη αναζήτηση σε αυτά. Το Katta[7] είναι ένα project του sourceforge.net, το οποίο χρησιμοποιεί σαν βάση του το Hadoop MapReduce και το Apache Lucene, και επιτρέπει τόσο την κατανεμημένη δημιουργία ευρετηρίων, όσο και την κατανεμημένη αναζήτηση σε αυτά. Προκειμένου να επιτευχθεί αυτό, το Katta διαιρεί τα κείμενα της συλλογής που δέχεται σαν είσοδο σε τμήματα και αναθέτει τη δημιουργία των επιμέρους ευρετηρίων για κάθε ένα από τα τμήματα της συλλογής σε διαφορετικούς κόμβους. Μετά την ολοκλήρωση της διαδικασίας αυτής, ο κόμβος του συστήματος που λειτουργεί ως Master αναθέτει σε κάθε κόμβο ένα σύνολο από τα επιμέρους ευρετήρια που έχουν δημιουργηθεί, ώστε ο κόμβος αυτός να αναλάβει τη συνένωση τους, σχηματίζοντας ένα μεγαλύτερο ευρετήριο το οποίο δημοσιεύει στον Master. Κάθε κόμβος, λοιπόν, διατηρεί ένα τμήμα του συνολικού ευρετηρίου και είναι υπεύθυνος για την εκτέλεση των αναζητήσεων σε αυτό. Όταν γίνεται ένα ερώτημα, κάθε κόμβος αναζητά τα αποτελέσματα στο τμήμα που διαθέτει και, στη συνέχεια, τα αποτελέσματα από τους επιμέρους κόμβους συνενώνονται ώστε να επιστραφούν στο χρήστη. Με τον τρόπο αυτό η αναζήτηση εκτελείται με κατανεμημένο τρόπο. Το μειονέκτημα του συστήματος αυτού είναι ότι δεν υποστηρίζει τη δυνατότητα ενημέρωσης του ευρετηρίου, με αποτέλεσμα να είναι αναγκαία η επαναδημιουργία του ευρετηρίου στην περίπτωση που απαιτείται να συμπεριληφθούν τροποποιημένα δεδομένα. Το Apache Solr[8] αποτελεί μια επέκταση του Apache Lucene, η οποία επιτρέπει την ενημέρωση του ευρετηρίου, αλλά και την κατανεμημένη αναζήτηση σε αυτό, χωρίς όμως να προσφέρει τη δυνατότητα κατανεμημένης δημιουργίας ή ενημέρωσης του ευρετηρίου.

Από την ακαδημαϊκή κοινότητα έχουν γίνει επίσης προσπάθειες για την κατανεμημένη δημιουργία ευρετηρίων για δεδομένα μεγάλης κλίμακας, αλλά και για την κατανεμημένη αναζήτηση σε αυτά. Το σύστημα Ivory[9], αξιοποιώντας τις δυνατότητες του Hadoop MapReduce, επιτρέπει την κατανεμημένη δημιουργία του ευρετηρίου, χωρίς ωστόσο να υποστηρίζει την κατανεμημένη αναζήτηση σε αυτό. Αντίθετα, το σύστημα HIndex[10] δημιουργεί κεντρικά το ευρετήριο, ενώ στη συνέχεια χρησιμοποιεί την κατανεμημένη βάση δεδομένων HBase, προκειμένου να γίνει εφικτή η κατανεμημένη αναζήτηση στο ευρετήριο που έχει δημιουργηθεί. Τέλος, από την ομάδα Κατανεμημένων Συστημάτων του εργαστηρίου Υπολογιστικών Συστημάτων του Ε.Μ.Π. έχει δημιουργηθεί μια πλατφόρμα[11] κατανεμημένης δημιουργίας ανεστραμμένων ευρετηρίων, η οποία μάλιστα επιτρέπει την κατανεμημένη αναζήτηση του περιεχομένου των ευρετηρίων, με τη χρήση της

HBase. Το μειονέκτημα του συγκεκριμένου συστήματος είναι ότι δεν έχει τη δυνατότητα ενημέρωσης του ευρετηρίου, με αποτέλεσμα, όπως και στην περίπτωση του Katta, να είναι απαραίτητη η περιοδική επαναδημιουργία του ευρετηρίου προκειμένου να συμπεριληφθούν τα νέα ή τροποποιημένα δεδομένα. Η παρούσα εργασία βασίζεται στο συγκεκριμένο σύστημα και προτείνει ορισμένες αλλαγές που είναι απαραίτητο να γίνουν στη δομή και τη λειτουργία του, προκειμένου να γίνει εφικτή η αποδοτική ενημέρωση του ευρετηρίου.

Κεφάλαιο 3

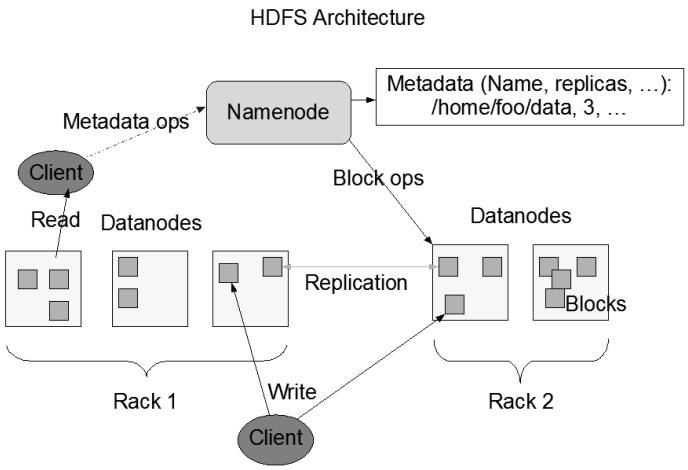
Υποσυστήματα που χρησιμοποιήθηκαν – Καταλληλότητα των υποσυστημάτων αυτών

3.1 Hadoop Distributed File System (HDFS)

Το Hadoop Distributed File System (HDFS)[12] είναι ένα project της Apache, το οποίο είναι διαθέσιμο στην ιστοσελίδα: http://hadoop.apache.org/hdfs/. Το HDFS βασίζεται στις αρχές του Google File System (GFS)[13] και αποτελεί μια υλοποίηση ανοιχτού λογισμικού του συστήματος αυτού. Πρόκειται για ένα κατανεμημένο σύστημα αρχείων σχεδιασμένο για εμπορικό hardware και όχι για ακριβά μηχανήματα με εξωτικά χαρακτηριστικά. Για το λόγο αυτό, είναι σχεδιασμένο για να κλιμακώνει σε μεγάλο αριθμό κόμβων, να επεκτείνεται εύκολα, καθώς και να είναι ανεκτικό σε σφάλματα, τα οποία είναι πολύ συχνότερα σε εμπορικό hardware. Έχει τη δυνατότητα να παρέχει υψηλού ρυθμού πρόσβαση στα δεδομένα και έτσι είναι κατάλληλο για εφαρμογές μαζικής επεξεργασίας (batch processing) σε μεγάλα datasets. Αυτό το χαρακτηριστικό το κάνει ιδιαίτερα κατάλληλο για τη δημιουργία ανεστραμμένων ευρετηρίων, αφού η εφαρμογή αυτή χαρακτηρίζεται από ιδιαίτερα αυξημένη πρόσβαση στα δεδομένα, με αποτέλεσμα να απαιτεί υψηλούς ρυθμούς πρόσβασης σε αυτά.

3.1.1 Η αρχιτεκτονική του HDFS

Στο παρακάτω διάγραμμα παρουσιάζεται η αρχιτεκτονική του HDFS:



Σχήμα 3: Η αρχιτεκτονική του HDFS

Το HDFS χρησιμοποιεί μια αρχιτεκτονική η οποία αποτελείται από έναν NameNode, ο οποίος ελέγχει το χώρο ονομάτων (namespace) του συστήματος αρχείων καθώς και την πρόσβαση των πελατών (clients) στα αρχεία, και από έναν αριθμό από DataNodes, συνήθως έναν ανά κόμβο, οι οποίοι ελέγχουν τα δεδομένα που βρίσκονται αποθηκευμένα στον κόμβο στον οποίο εκτελούνται. Το HDFS παρέχει ένα ενιαίο namespace και επιτρέπει την αποθήκευση των δεδομένων σε αρχεία, τα οποία στη συνέχεια διαιρούνται σε ένα ή περισσότερα blocks και αποθηκεύονται σε διαφορετικούς κόμβους. Ο NameNode είναι υπεύθυνος για τις απαραίτητες λειτουργίες στο namespace του συστήματος αρχείων, όπως το άνοιγμα και το κλείσιμο αρχείων, καθώς και για την κατανομή των blocks στους DataNodes, ενώ οι DataNodes είναι υπεύθυνοι για την ανάγνωση και εγγραφή των αρχείων, καθώς και την δημιουργία, τη διαγραφή και τη δημιουργία αντιγράφων (replication) των blocks σύμφωνα με τις εντολές του NameNode. Όπως φαίνεται ο NameNode αποθηκεύει όλα τα μεταδεδομένα για τα αρχεία του συστήματος, ενώ η λειτουργία του συστήματος δεν προβλέπει τη ροή δεδομένων μέσα από αυτόν, ώστε να επιταχύνεται η πρόσβαση στα δεδομένα και ταυτόχρονα να αποφεύγεται η συμφόρηση του συστήματος από την υπερβολική ροή δεδομένων μέσα από τον NameNode.

3.1.2 Βασικά χαρακτηριστικά και καταλληλότητα συστήματος

Στη συνέχεια παρουσιάζονται τα βασικά χαρακτηριστικά του HDFS, όπως αυτά παρουσιάζονται από την επίσημη σελίδα του project, τα οποία καθιστούν τη χρήση του κατάλληλη για τη δημιουργία και την ενημέρωση ανεστραμμένων ευρετηρίων:

Ανεκτικότητα σε σφάλματα του υλικού (Fault Tolerance)
Το HDFS, όπως αναφέραμε και προηγουμένως, είναι ιδιαίτερα ανεκτικό στα σφάλματα του υλικού, τα οποία είναι εξαιρετικά συνηθισμένα στην περίπτωση συστημάτων εκατοντάδων ή χιλιάδων κόμβων εμπορικού hardware. Το σύστημα αναγνωρίζει τα σφάλματα και ανακάμπτει αυτόματα. Ακόμα, με τη δημιουργία αντιγράφων στα δεδομένα (replication), η οποία περιγράφεται στη συνέχεια, και τη διανομή τους σε διαφορετικούς DataNodes αποφεύγεται η απώλεια πολύτιμων δεδομένων κατά την εμφάνιση ενός σφάλματος στο υλικό.

Αντίγραφα Δεδομένων (Data Replication)

Κάθε αρχείο στο HDFS διαιρείται σε έναν αριθμό από blocks στα οποία δημιουργούνται αντίγραφα προκειμένου να επιτευχθεί η ανοχή στα σφάλματα του υλικού. Με τη δημιουργία και τη σωστή τοποθέτηση των αντιγράφων στους διάφορους DataNodes, αυξάνεται η διαθεσιμότητα των δεδομένων, αφού σε ένα αίτημα ανάγνωσης, το σύστημα έχει τη δυνατότητα να επιλέγει το 'κοντινότερο' αντίγραφο του block που πρέπει να διαβαστεί, επιταχύνοντας σημαντικά τη διαδικασία ανάγνωσης. Η τελευταία αυτή ιδιότητα είναι ιδιαίτερα σημαντική για εφαρμογές με μεγάλο πλήθος αναγνώσεων, όπως η εφαρμογή που εξετάζουμε.

Πρόσβαση Ροής Δεδομένων (Data Stream Access)

Το HDFS δίνει πρόσβαση στα δεδομένα με υψηλό ρυθμό, αλλά και με σχετικά υψηλή καθυστέρηση. Αυτό το καθιστά κατάλληλο για μαζική επεξεργασία δεδομένων, όπως η εφαρμογή που εξετάζουμε όπου απαιτείται ταχεία ανάγνωση μεγάλου όγκου δεδομένων.

Δεδομένα Μεγάλου Όγκου

Το HDFS είναι σχεδιασμένο και ρυθμισμένο για την αποθήκευση και τη διαχείριση μεγάλου όγκου δεδομένων. Αυτό είναι εμφανές από το ιδιαίτερα μεγάλο μέγεθος των blocks το οποίο είναι αρχικά ορισμένο στα 64MB. Για το λόγο αυτό, το HDFS θεωρείται κατάλληλο για εφαρμογές οι οποίες επεξεργάζονται μεγάλο όγκο δεδομένων.

Μοντέλο Συνάφειας (Coherency Model)

Το HDFS παρέχει ένα απλό μοντέλο συνάφειας σύμφωνα με το οποίο κάθε αρχείο, στο οποίο έχει ολοκληρωθεί η εγγραφή, δεν πρέπει να τροποποιηθεί. Έτσι, υπάρχει η δυνατότητα για υψηλό ρυθμό ανάγνωσης των δεδομένων από διαφορετικούς κόμβους του συστήματος, με αποτέλεσμα εφαρμογές, όπως αυτή που εξετάζουμε, οι οποίες αρχικά περιλαμβάνουν την ανάγνωση μεγάλου όγκου δεδομένων και, στη συνέχεια, τη μαζική εγγραφή στο τέλος της εφαρμογής να επωφελούνται σημαντικά από το μοντέλο αυτό.

Εκτέλεση επεξεργασίας 'κοντά' στα δεδομένα

Τέλος, ένα από τα βασικότερα χαρακτηριστικά του HDFS, το οποίο επιταχύνει σημαντικά την εκτέλεση εφαρμογών με απαιτητική πρόσβαση στα δεδομένα, είναι η δυνατότητα του να μεταφέρει το επεξεργαστικό μέρος της εφαρμογής 'κοντά' στα δεδομένα που απαιτούνται. Με τον τρόπο αυτό αποφεύγεται η μεταφορά μεγάλου όγκου δεδομένων μέσα από το δίκτυο, η οποία είναι ιδιαίτερα χρονοβόρα. Το χαρακτηριστικό αυτό επιτυγχάνεται

σε συνδυασμό με το MapReduce framework, το οποίο παρουσιάζεται στη συνέχεια, και είναι ιδιαίτερα σημαντικό για εφαρμογές οι οποίες απαιτούν την ανάγνωση μεγάλου όγκου δεδομένων. Χωρίς τη συγκεκριμένη δυνατότητα, θα ήταν απαραίτητη η μεταφορά ενός μεγάλου όγκου δεδομένων μέσω του δικτύου, γεγονός που θα επιβράδυνε σημαντικά την εκτέλεση της εφαρμογής.

3.2 Hadoop MapReduce

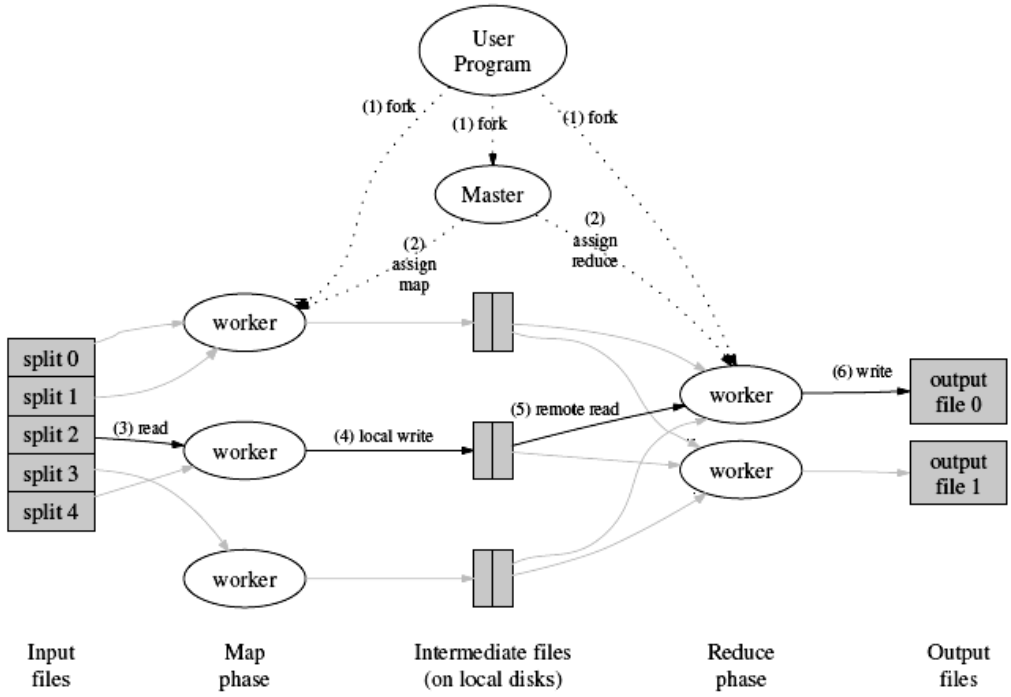
Το Hadoop MapReduce[14] είναι ένα project της Apache, το οποίο αποτελεί μια υλοποίηση, ανοιχτού κώδικα, της γενικότερης ιδέας του MapReduce και βασίζει τη λειτουργία του στο HDFS. Το project αυτό είναι διαθέσιμο στην ιστοσελίδα: http://hadoop.apache.org/mapreduce/. Πρόκειται για ένα framework το οποίο διευκολύνει την ανάπτυξη εφαρμογών, με σκοπό την κατανεμημένη επεξεργασία μεγάλου όγκου δεδομένων σε συστήματα shared-nothing αρχιτεκτονικής, πολλών κόμβων, παρέχοντας ταυτόχρονα αξιοπιστία και ανοχή σε σφάλματα.

3.2.1 Αρχιτεκτονική και λειτουργία του συστήματος

Για την καλύτερη κατανόηση των πλεονεκτημάτων του MapReduce framework, που το καθιστούν κατάλληλο για την εφαρμογή που εξετάζουμε, είναι απαραίτητο να γίνει μια αναλυτική περιγραφή της αρχιτεκτονικής, αλλά και της λειτουργίας του.

Η ιδέα του MapReduce είναι απλή, αλλά ταυτόχρονα ιδιαίτερα αποδοτική για ένα μεγάλο αριθμό εφαρμογών. Παρουσιάστηκε το 2004 από την Google[3] και έχει επικρατήσει, τόσο για την δυνατότητα κλιμάκωσης σε μεγάλες συστοιχίες υπολογιστών, όσο και για την απλότητα του μοντέλου, που το καθιστά ιδιαίτερα εύχρηστο. Η διαδικασία που προτείνεται από το μοντέλο αυτό μπορεί να διακριθεί σε δύο διαδοχικές φάσεις επεξεργασίας: τη φάση του Map και τη φάση του Reduce. Η ιδέα αυτή προέρχεται από τον συναρτησιακό προγραμματισμό, σύμφωνα με τον οποίο μια συνάρτηση επιστρέφει πάντα την ίδια έξοδο για μια συγκεκριμένη είσοδο. Έτσι, ένα πρόγραμμα που αναπτύσσεται με αυτή τη λογική είναι ιδιαίτερα εύκολο να παραλληλοποιηθεί, αφού η επεξεργασία ενός τμήματος των δεδομένων μπορεί να γίνεται ανεξάρτητα από τα υπόλοιπα δεδομένα, χωρίς αυτό να επηρεάζει το τελικό αποτέλεσμα.

Στο παρακάτω σχήμα παρουσιάζονται γραφικά η αρχιτεκτονική, αλλά και ο τρόπος λειτουργίας του MapReduce:



Σχήμα 4: Η αρχιτεκτονική και η λειτουργία του μοντέλου MapReduce

Η διαδικασία ελέγχεται πλήρως από το MapReduce framework, το οποίο εκτελεί όλες τις απαραίτητες ενέργειες για την ορθή εκτέλεση της εργασίας. Συγκεκριμένα, ο MasterNode, ή αλλιώς JobTracker, όπως ονομάζεται στο Hadoop MapReduce, λαμβάνει τις πληροφορίες σχετικά με την εργασία που επιθυμεί να εκτελέσει ο χρήστης και αναθέτει στους άλλους κόμβους του συστήματος, δηλαδή τους Workers, ή αλλιώς TaskTrackers όπως ονομάζονται στο Hadoop MapReduce, το είδος των map και reduce tasks που πρέπει να εκτελέσουν. Αφού ολοκληρωθεί η εκτέλεση των map tasks, ο MasterNode ενημερώνεται από τα map tasks για την τοποθεσία των ενδιάμεσων αποτελεσμάτων και στη συνέχεια ενημερώνει τα reduce tasks για τη θέση των δεδομένων που απαιτείται να διαβάσουν. Τέλος, μετά την ολοκλήρωση και των reduce tasks, ο MasterNode ενημερώνεται για την ολοκλήρωση της εργασίας και επιστρέφει τις απαραίτητες πληροφορίες στο χρήστη.

Κατά τη φάση του Map τα δεδομένα μοιράζονται στους διάφορους κόμβους του συστήματος ώστε να εκτελεστεί το πρώτο στάδιο επεξεργασίας, ενώ τα αποτελέσματα της φάσης αυτής ομαδοποιούνται και ταξινομούνται προκειμένου να χρησιμοποιηθούν ως είσοδος για τη δεύτερη φάση της διαδικασίας. Κατά τη φάση του Reduce τα ενδιάμεσα

δεδομένα περνούν από το δεύτερο στάδιο επεξεργασίας προκειμένου να υπολογιστεί η τελική έξοδος της εφαρμογής η οποία αποθηκεύεται στο κατανεμημένο σύστημα αρχείων.

Πιο συγκεκριμένα, η διαδικασία που ακολουθείται από μια MapReduce εργασία είναι η εξής:

- Τα δεδομένα εισόδου διαιρούνται σε έναν αριθμό ανεξάρτητων τμημάτων, τα οποία ανατίθενται σε έναν ίσο αριθμό από map tasks .
- Κάθε map task διαβάζει το τμήμα της εισόδου που του αντιστοιχεί, το οποίο βρίσκεται αποθηκευμένο σε ένα κατανεμημένο σύστημα αρχείων, όπως το HDFS, και χρησιμοποιώντας μια συνάρτηση η οποία έχει δημιουργηθεί από το χρήστη μετατρέπει το σύνολο των δεδομένων σε ένα πλήθος ζευγών Κλειδιού-Τιμής (key/value pairs).
- Κάθε map task επεξεργάζεται τα key/value pairs που του έχουν ανατεθεί σύμφωνα με το πρόγραμμα που έχει αναπτύξει ο χρήστης και παράγει έναν αριθμό από ενδιάμεσα αποτελέσματα, τα οποία έχουν και πάλι τη μορφή key/value pairs.
- Για κάθε key/value pair χρησιμοποιείται μια συνάρτηση διαχωρισμού (partitioning function), η οποία ορίζεται από το χρήστη και καθορίζει τον τρόπο με τον οποίο θα μοιραστούν τα ενδιάμεσα αποτελέσματα στα reduce tasks προκειμένου να περάσουν από το τελικό στάδιο επεξεργασίας.
- Προαιρετικά, τα key/value pairs που προκύπτουν σαν έξοδος των map tasks μπορούν να περάσουν από ένα ενδιάμεσο στάδιο επεξεργασίας που ονομάζεται combine. Το στάδιο αυτό έχει τη μορφή ενός τοπικού reduce για τα δεδομένα εξόδου ενός map task και χρησιμεύει κυρίως όταν ο σκοπός του reducer είναι η εκτέλεση ενός aggregation στα δεδομένα. Το σύστημα δεν εγγυάται την εκτέλεση της φάσης του combine και για το λόγο αυτό η χρήση του λειτουργεί μόνο επικουρικά για τον περιορισμό των δεδομένων που πρέπει να μεταφερθούν στους reducers.
- Τα ενδιάμεσα δεδομένα που έχουν παραχθεί από την επεξεργασία των map tasks αποθηκεύονται στο τοπικό σύστημα αρχείων των κόμβων όπου εκτελούνται τα map tasks.
- Το framework διαβάζει και ταξινομεί τα ενδιάμεσα δεδομένα με βάση το κλειδί τους, ομαδοποιώντας ταυτόχρονα τα ζεύγη που έχουν κοινό κλειδί.

- Τα ομαδοποιημένα και ταξινομημένα ενδιάμεσα δεδομένα αποτελούν την είσοδο των reduce tasks. Κάθε reduce task παίρνει τα key/value pairs που του αντιστοιχούν, ανάλογα με την partitioning function που όρισε ο χρήστης.
- Τα reduce tasks επεξεργάζονται τα δεδομένα και υπολογίζουν την τελική έξοδο της εργασίας, περιορίζοντας συνήθως τον αριθμό των key/value pairs.
- Η έξοδος των reduce tasks είναι και η τελική έξοδος της εφαρμογής και αποθηκεύεται σε ένα κατανεμημένο σύστημα αρχείων, όπως το HDFS.

Η ροή των δεδομένων κατά την εκτέλεση μιας MapReduce εργασίας μπορεί να αναπαρασταθεί ως εξής:

$$(input)\langle k1, v1\rangle \rightarrow map \rightarrow \langle k2, v2\rangle \rightarrow combine \rightarrow \langle k2, v2\rangle \rightarrow reduce \rightarrow \langle k3, v3\rangle output$$

Όπου τα k1…3 και v1…3 αναπαριστούν τον τύπο των keys και values και όχι την τιμή τους.

3.2.2 Βασικά χαρακτηριστικά και καταλληλότητα συστήματος

Στο σημείο αυτό είναι απαραίτητο να παρουσιάσουμε τα χαρακτηριστικά του MapReduce framework που το καθιστούν μια εξαιρετική επιλογή για την εφαρμογή που εξετάζουμε.

Το MapReduce αποτελεί ουσιαστικά ένα μοντέλο σχεδίασης εργασιών, το οποίο αποτελείται από δύο διαδοχικές και διακριτές φάσεις, όπως περιγράφτηκε προηγουμένως. Το γεγονός αυτό καθιστά το MapReduce ακατάλληλο για ένα μεγάλο αριθμό εργασιών οι οποίες δεν είναι δυνατό να υλοποιηθούν αναλύοντας την εργασία σε έναν αριθμό MapReduce εργασιών ή η ανάλυση τους σε εργασίες MapReduce είναι τόσο πολύπλοκη που το επιπλέον κόστος από τα ενδιάμεσα βήματα που θα χρειαστεί να πραγματοποιηθούν υπερκαλύπτει τα οφέλη από την χρήση του framework.

Η εφαρμογή που εξετάζουμε, ωστόσο, είναι εύκολα αντιληπτό ότι μπορεί με απλό τρόπο να υλοποιηθεί με τη χρήση του MapReduce framework, αφού εξ ορισμού το πρόβλημα που καλείται να αντιμετωπίσει μπορεί να επιλυθεί μέσω διαδοχικών βημάτων map και reduce. Αυτό, βέβαια δεν είναι τυχαίο, αφού η ιδέα του MapReduce παρουσιάστηκε για πρώτη φορά από την Google, μία από τις μεγαλύτερες μηχανές αναζήτησης παγκοσμίως, με κύριο στόχο την ταχύτερη δημιουργία ευρετηρίων για τις ιστοσελίδες του διαδικτύου.

Συγκεκριμένα, η δημιουργία ανεστραμμένων ευρετηρίων απαιτεί, αρχικά, την σάρωση των κειμένων, ώστε να αναγνωριστούν οι λέξεις που θα συμπεριληφθούν στο ευρετήριο και, στη συνέχεια, την συγκέντρωση, για κάθε λέξη ξεχωριστά, των κειμένων στα οποία εμφανίστηκε η λέξη αυτή, ώστε να δημιουργηθεί η τελική λίστα κειμένων για κάθε λέξη. Είναι εμφανές ότι η διαδικασία αυτή ταιριάζει απόλυτα σε μία διαδικασία MapReduce και ότι η χρήση του MapReduce framework είναι μια πολύ καλή επιλογή για την υλοποίηση της εφαρμογής που εξετάζουμε.

Πέρα από την ευκολία της υλοποίησης της εφαρμογής με τη χρήση του MapReduce framework, υπάρχουν και άλλα χαρακτηριστικά που υποδεικνύουν πως η χρήση του θα οδηγήσει στα επιθυμητά αποτελέσματα. Το MapReduce framework, σε συνδυασμό με το HDFS, είναι σχεδιασμένο για να παρέχει υψηλού ρυθμού πρόσβαση στα δεδομένα, με αποτέλεσμα να είναι ιδιαίτερα αποδοτικό για εργασίες batch processing, όπου το κόστος πρόσβασης στα δεδομένα είναι αυξημένο. Ακόμα, όπως αναφέρθηκε σε προηγούμενη ενότητα, το MapReduce framework, σε συνδυασμό με το HDFS, επιτρέπει την εκτέλεση του υπολογιστικού μέρους της εφαρμογής κοντά στα δεδομένα, αποφεύγοντας την άσκοπη μεταφορά δεδομένων, η οποία επιβραδύνει σημαντικά την εκτέλεση της εφαρμογής και επιβαρύνει το δίκτυο. Με τον τρόπο αυτό, στην εφαρμογή που εξετάζουμε, αποφεύγεται, κατά το δυνατόν, η μεταφορά των κειμένων μέσω του δικτύου για την επεξεργασία τους σε κάποιον απομακρυσμένο κόμβο, με αποτέλεσμα να επιταχύνεται η εκτέλεση της επεξεργασίας.

Επιπρόσθετα, η χρήση εμπορικού υλικού, χωρίς ιδιαίτερα χαρακτηριστικά, αυξάνει σε μεγάλο βαθμό την πιθανότητα σφαλμάτων. Τα σφάλματα υλικού είναι πιθανό να επιβραδύνουν ή και να διακόψουν εντελώς την εκτέλεση μιας εφαρμογής, καθώς και να οδηγήσουν στην απώλεια πολύτιμων δεδομένων. Το MapReduce framework αναλαμβάνει την αντιμετώπιση τέτοιων περιπτώσεων χωρίς να απαιτείται κάποια επιπλέον εργασία από τον προγραμματιστή. Όσον αφορά την ορθή εκτέλεση της εργασίας, το framework ελέγχει περιοδικά την πρόοδο των διαφόρων tasks και επαναλαμβάνει την εκτέλεση τους σε περίπτωση που κάποιο αποτύχει ή καθυστερήσει σημαντικά. Ακόμα και αν ένας ή περισσότεροι κόμβοι σταματήσουν να αποκρίνονται το framework μεταφέρει τα tasks, που εκτελούνταν στους κόμβους αυτούς, σε άλλους ενεργούς κόμβους και η εκτέλεση της εργασίας συνεχίζεται κανονικά.

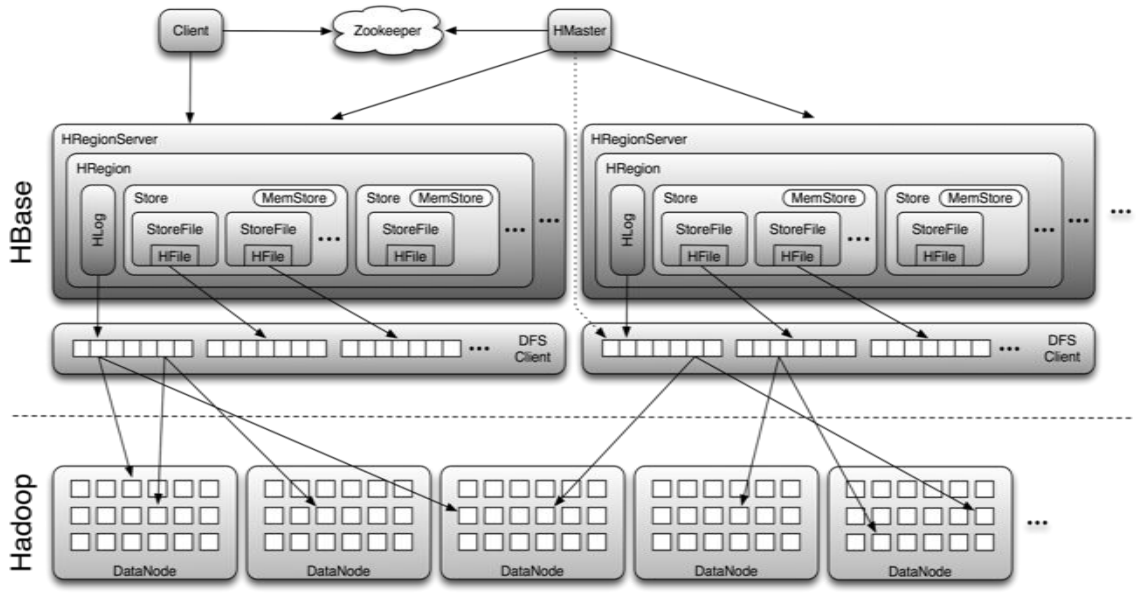
3.3 Apache HBase

Η HBase[15] είναι ένα project της Apache, το οποίο είναι διαθέσιμο στην ιστοσελίδα: http://hbase.apache.org/. Πρόκειται για ένα κατανεμημένο σύστημα αποθήκευσης δομημένων δεδομένων, το οποίο αποθηκεύει τα δεδομένα με αραιό τρόπο, χρησιμοποιώντας διαφορετικές εκδόσεις και με τρόπο προσανατολισμένο στις στήλες και όχι στις γραμμές των δομημένων δεδομένων, σε αντίθεση με τα περισσότερα εμπορικά συστήματα βάσεων δεδομένων. Το σύστημα αυτό βασίζεται στο HDFS για την αποθήκευση των δεδομένων και δημιουργήθηκε για να παρέχει πραγματικού χρόνου πρόσβαση σε μεγάλου όγκου δεδομένα, κάτι το οποίο δεν προσφέρει το HDFS. Η HBase έχει τη δυνατότητα να διαχειρίζεται πίνακες δισεκατομμυρίων γραμμών και εκατομμυρίων στηλών, αποθηκεύοντας τους σε συστοιχίες υπολογιστών εμπορικού υλικού.

Η HBase σχεδιάστηκε με πρότυπο το Bigtable[16], που παρουσιάστηκε από την Google το 2006, και αποτελεί ένα κατανεμημένο σύστημα αποθήκευσης δεδομένων με στόχο να διαχειρίζεται αποδοτικά δομημένα δεδομένα και να κλιμακώνει για εξαιρετικά μεγάλο όγκο δεδομένων και μεγάλο αριθμό κόμβων. Η κυριότερη διαφορά των δύο αυτών συστημάτων σε σχέση με τις παραδοσιακές βάσεις δεδομένων είναι ότι δεν παρέχουν πλήρες σχεσιακό μοντέλο. Ο χρήστης έχει τη δυνατότητα να τροποποιεί δυναμικά το μοντέλο και τη μορφή των δεδομένων, αφού ο αριθμός και το όνομα των στηλών, καθώς και ο τύπος των δεδομένων κάθε στήλης δεν προκαθορίζονται κατά την κατασκευή του πίνακα. Ακόμα, είναι δυνατή η αποθήκευση κάθε μορφής συμβολοσειράς με αποτέλεσμα να δίνεται η δυνατότητα στο χρήστη να αποθηκεύσει κάθε μορφής δομή δεδομένων, εφόσον αυτή έχει σειριοποιηθεί.

3.3.1 Αρχιτεκτονική του συστήματος

Στο παρακάτω σχήμα παρουσιάζεται γραφικά η αρχιτεκτονική της HBase και ο τρόπος με τον οποίο αυτή συνδέεται με το HDFS:



Σχήμα 5: Η αρχιτεκτονική της HBase

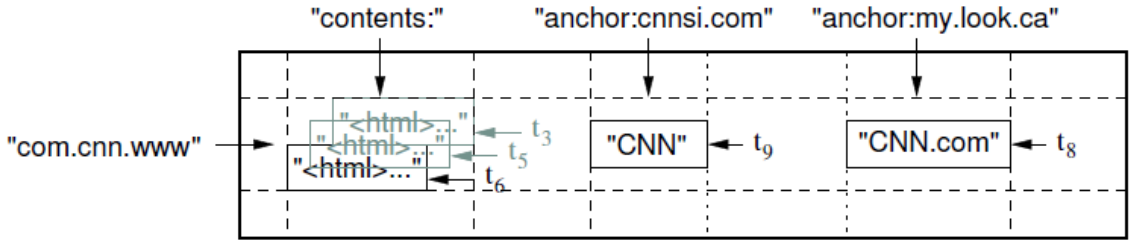
Η αρχιτεκτονική της HBase είναι ιδιαίτερα απλή και μοιάζει σε μεγάλο βαθμό με αυτή του HDFS. Το σύστημα αποτελείται από έναν Master Server (HMaster) και έναν αριθμό από Region Servers (HRegionServer), οι οποίοι εκτελούνται στους διαφορετικούς κόμβους της συστοιχίας υπολογιστών. Ο HMaster είναι υπεύθυνος για τον έλεγχο των HRegionServers, καθώς και για την αποθήκευση των μεταδεδομένων. Ακόμα, ο HMaster εκτελεί τις απαραίτητες διαδικασίες προκειμένου να επιτευχθεί ομοιόμορφη κατανομή του φορτίου στους διάφορους κόμβους του συστήματος. Οι HRegionServers, αντίθετα, είναι υπεύθυνοι για την αποθήκευση, διαχείριση και ανάκτηση των δεδομένων μετά από κάποιο αίτημα του χρήστη ή κατά το load balancing. Για το συγχρονισμό των διαφόρων ενεργειών πάνω στα δεδομένα, η HBase απαιτεί την ύπαρξη του ZooKeeper[17]. Πρόκειται για ένα κεντρικό σύστημα για το συγχρονισμό κατανεμημένων εφαρμογών, το οποίο είναι απαραίτητο για την εξασφάλιση της συνάφειας των δεδομένων της HBase. Όπως φαίνεται και στο παραπάνω σχήμα, οι HRegionServers χρησιμοποιούν το HDFS για την αποθήκευση των δεδομένων τους.

3.3.2 Μοντέλο δεδομένων

Το μοντέλο που χρησιμοποιείται από την HBase διαφέρει, όπως αναφέραμε, από το παραδοσιακό σχεσιακό μοντέλο. Το μοντέλο αυτό προέκυψε αναλύοντας τις ανάγκες αποθήκευσης δεδομένων της Google, και πιο συγκεκριμένα με σκοπό την αποθήκευση ποικίλλων πληροφοριών για ένα μεγάλο αριθμό ιστοσελίδων του διαδικτύου. Σύμφωνα με

το μοντέλο δεδομένων που περιγράφεται από τους δημιουργούς του Bigtable, το οποίο ακολουθήθηκε πιστά κατά την ανάπτυξη της HBase, τα συστήματα αυτά μπορούν να θεωρηθούν σαν ένας αραιός, κατανεμημένος και ταξινομημένος χάρτης δεδομένων. Για την ταχύτερη πρόσβαση στα δεδομένα τα συστήματα αυτά διαθέτουν ευρετήριο όχι μόνο για τις γραμμές, αλλά και για τις στήλες, καθώς και τις διαφορετικές εκδόσεις των δεδομένων του χάρτη.

Στο παρακάτω σχήμα παρουσιάζεται γραφικά η μορφή μιας γραμμής στην HBase ώστε να γίνει περισσότερο κατανοητή η ανάλυση που ακολουθεί:



Σχήμα 6: Παράδειγμα γραμμής ενός πίνακα στην HBase

Γραμμή (Row)

Κάθε γραμμή δεδομένων περιέχει ένα κλειδί, το οποίο μπορεί να είναι μια οποιαδήποτε συμβολοσειρά. Στο παράδειγμα, το κλειδί της υποθετικής αυτής γραμμής είναι το "com.cnn.www". Η HBase διατηρεί τις γραμμές αποθηκευμένες σε λεξικογραφική σειρά με βάση το κλειδί τους. Το διάστημα τιμών του κλειδιού διαιρείται σε ανεξάρτητα τμήματα τα οποία μοιράζονται στους RegionServers, προκειμένου να κατανεμηθεί το φορτίο στους διαφορετικούς κόμβους του συστήματος.

Οικογένεια Στηλών (Column Family)

Οι στήλες μιας γραμμής ομαδοποιούνται σε σύνολα τα οποία ονομάζονται "column families". Αυτή είναι η βασική μονάδα οργάνωσης στην HBase και έτσι σε γενικές γραμμές τα δεδομένα που αποθηκεύονται σε ένα column family πρέπει να είναι του ίδιου τύπου. Οι column families ενός πίνακα ορίζονται κατά τη δημιουργία του, ωστόσο κάθε column family μπορεί να περιέχει έναν απεριόριστο αριθμό από στήλες, ο οποίος, μάλιστα, μπορεί να μεταβάλλεται και να διαφέρει από γραμμή σε γραμμή, γεγονός που αποδεικνύει ότι πρόκειται για έναν αραιό πίνακα-χάρτη δεδομένων. Οι στήλες μιας column family μπορούν να έχουν οποιοδήποτε όνομα, το οποίο ονομάζεται qualifier, ενώ το συνολικό τους όνομα διαμορφώνεται από το συντακτικό: family_name:qualifier. Στο παραπάνω παράδειγμα,

έχουμε δύο column families, τις “content” και “anchor”, ενώ η anchor περιέχει, για την συγκεκριμένη γραμμή δύο στήλες με qualifiers “cnnsi.com” και “my.look.ca” αντίστοιχα.

Κελιά (Cells)

Κάθε κελί της HBase προσδιορίζεται μονοσήμαντα από την πλειάδα (row:string, column:string, timestamp:int64) και περιέχει έναν πίνακα από bytes.

Εκδόσεις (Versions)

Η HBase παρέχει τη δυνατότητα αποθήκευσης ενός απεριόριστου αριθμού από δεδομένα για το ίδιο row και column, χρησιμοποιώντας όμως διαφορετικό timestamp, δηλαδή έκδοση για τα δεδομένα. Η αποθήκευση στη διάσταση της έκδοσης γίνεται με φθίνουσα ταξινόμηση ως προς το timestamp, ώστε τα πιο πρόσφατα δεδομένα να ανακτώνται πρώτα. Στο παραπάνω παράδειγμα, παρατηρούμε ότι η στήλη “content:” περιέχει τρεις διαφορετικές εκδόσεις των δεδομένων, τις t_6, t_5 και t_3.

3.3.3 Βασικά χαρακτηριστικά

Κατανεμημένη αποθήκευση και πρόσβαση στα δεδομένα

Η HBase προσφέρει κατανεμημένη αποθήκευση και πρόσβαση στα δεδομένα. Διαιρώντας το σύνολο των κλειδιών ενός πίνακα σε ανεξάρτητα τμήματα και αναθέτοντας τη διαχείριση κάθε τμήματος σε ξεχωριστό RegionServer, η HBase εξασφαλίζει τη κατανομή του φορτίου στους κόμβους του συστήματος. Ταυτόχρονα, χρησιμοποιώντας σαν βάση το HDFS, η HBase έχει τη δυνατότητα να μοιράζει τα δεδομένα σε ένα σύνολο κόμβων και κατά συνέπεια να χρησιμοποιεί τους κόμβους αυτούς για την ανάκτηση των δεδομένων. Με τη δημιουργία αντιγράφων από το HDFS εξασφαλίζεται η ανοχή του συστήματος σε σφάλματα και υπάρχει η δυνατότητα διαμοιρασμού του φορτίου αναγνώσεων μεταξύ των κόμβων.

Μοντέλο αραιής αποθήκευσης των δεδομένων

Η HBase παρέχει ένα μοντέλο αραιής αποθήκευσης των δεδομένων. Κάθε γραμμή μπορεί να αποτελείται από έναν οποιοδήποτε αριθμό στηλών. Για το λόγο αυτό, είναι απαραίτητη η δημιουργία ευρετηρίων και ως προς τις στήλες, για την γρήγορη αναζήτηση ενός στοιχείου του πίνακα. Αυτό είναι μια σημαντικά διαφορά της HBase σε σχέση με τα παραδοσιακά συστήματα βάσεων δεδομένων, η οποία την καθιστά κατάλληλη για την αποθήκευση δεδομένων των οποίων η δομή δεν είναι απόλυτα προκαθορισμένη.

Ανοχή στα σφάλματα του υλικού

Χρησιμοποιώντας το HDFS για την αποθήκευση των δεδομένων που διαχειρίζεται, η HBase κληρονομεί τη δυνατότητα να αντιμετωπίζει σφάλματα στο υλικό, τα οποία είναι ιδιαίτερα συνηθισμένα όταν χρησιμοποιείται εμπορικό υλικό. Με τη δημιουργία αντιγράφων, η οποία εξασφαλίζεται από το HDFS, η HBase έχει τη δυνατότητα να ανακτά τα δεδομένα ακόμα και όταν κάποιοι κόμβοι του συστήματος είναι εκτός λειτουργίας.

3.3.4 Καταλληλότητα της HBase

Έχοντας αναλύσει τα βασικά χαρακτηριστικά της HBase, μπορούμε να αναδείξουμε την καταλληλότητα της για την εφαρμογή που εξετάζουμε. Οι βασικότεροι λόγοι που οδήγησαν στην επιλογή της HBase για την υλοποίηση του συστήματος είναι οι εξής:

- Ο μεγάλος φόρτος ερωτημάτων απαιτεί την ύπαρξη μιας βάσης δεδομένων, και όχι ενός απλού συστήματος αρχείων, όπως το HDFS. Με τη χρήση ευρετηρίων, τόσο στις γραμμές, όσο και τις στήλες των πινάκων, η HBase επιταχύνει σημαντικά την πρόσβαση στα δεδομένα, η οποία αποτελούσε αδυναμία του HDFS, το οποίο παρέχει πρόσβαση υψηλού ρυθμού, αλλά και με υψηλή καθυστέρηση. Με τη χρήση της HBase έχουμε πρόσβαση πραγματικού χρόνου σε οποιοδήποτε τμήμα των δεδομένων, γεγονός που μας επιτρέπει να επιταχύνουμε την επιστροφή των αποτελεσμάτων στους χρήστες. Ταυτόχρονα, η ταχύτερη ανάκτηση των δεδομένων βοηθάει και στην επιτάχυνση της διαδικασίας ενημέρωσης του ευρετηρίου. Αυτό γίνεται καλύτερα κατανοητό σε επόμενο κεφάλαιο, όπου αναλύεται η διαδικασία ενημέρωσης.
- Στο πρώτο κεφάλαιο παρουσιάστηκε η αναγκαιότητα κατανεμημένης αναζήτησης των δεδομένων του ανεστραμμένου ευρετηρίου, εξαιτίας του ιδιαίτερα μεγάλου μεγέθους του, αλλά και του αυξημένου φορτίου ερωτημάτων από τους χρήστες του διαδικτύου. Η HBase είναι μια κατανεμημένη βάση δεδομένων, η οποία επιτρέπει την κατανομή του φορτίου των ερωτημάτων σε διαφορετικούς κόμβους του συστήματος. Με τον τρόπο αυτό το σύστημα έχει τη δυνατότητα να διαχειριστεί μεγαλύτερο φορτίο ερωτημάτων σε σχέση με μια κεντρική βάση δεδομένων, η οποία θα δεχόταν όλα τα ερωτήματα των χρηστών.
- Η HBase είναι βασισμένη στο HDFS και έτσι είναι σχεδιασμένη για να λειτουργεί αποδοτικά σε συνδυασμό με αυτό. Ακόμα, είναι σχεδιασμένη για να λειτουργεί αποδοτικά σε συνδυασμό με το Hadoop MapReduce, παρέχοντας διαπροσωπίες για

την ανάγνωση δεδομένων από την HBase, με σκοπό την επεξεργασία τους από το MapReduce, την εγγραφή των δεδομένων εξόδου μιας MapReduce εργασίας στην HBase, αλλά και την μαζική εισαγωγή των δεδομένων (bulk loading) που προέκυψαν ως έξοδος του MapReduce, με τη χρήση ειδικών αρχείων που ονομάζονται HFiles.

- Το μοντέλο δεδομένων που χρησιμοποιεί η HBase, το οποίο διαφέρει από το παραδοσιακό σχεσιακό μοντέλο, και κυρίως το γεγονός ότι κάθε γραμμή μπορεί να διαθέτει διαφορετικό αριθμό στηλών, με οποιοδήποτε όνομα, ταιριάζει με τον καλύτερο δυνατό τρόπο στις ανάγκες της εφαρμογής που εξετάζουμε. Αυτό θα γίνει περισσότερο κατανοητό σε επόμενο κεφάλαιο όπου περιγράφεται αναλυτικά το σχήμα που επιλεγούμε για την αποθήκευση των εγγραφών του ευρετηρίου. Η καταλληλότητα του μοντέλου που χρησιμοποιεί η HBase, βέβαια, δεν είναι τυχαία, αφού το μοντέλο αυτό αρχικά παρουσιάστηκε από την Google με σκοπό την εξυπηρέτηση παρόμοιων εφαρμογών.

Κεφάλαιο 4
Δημιουργία και ενημέρωση ανεστραμμένου ευρετηρίου

4.1 Εισαγωγή

Στο κεφάλαιο αυτό παρουσιάζεται αναλυτικά το πρόβλημα που καλούμαστε να αντιμετωπίσουμε, καθώς και οι διαδικασίες που σχεδιάστηκαν για την αποδοτική δημιουργία και ενημέρωση ανεστραμμένων ευρετηρίων. Στόχος της εργασίας αυτής, όπως περιγράφτηκε και στο πρώτο κεφάλαιο, είναι η αποδοτική ενημέρωση του ευρετηρίου μετά από την τροποποίηση ενός μέρους των κειμένων και την προσθήκη νέων κειμένων, χωρίς ωστόσο να επηρεάζεται ο χρόνος αρχικής δημιουργίας του ευρετηρίου και κυρίως ο χρόνος ανάκτησης των δεδομένων για την επεξεργασία των ερωτημάτων των χρηστών.

4.2 Βασικά χαρακτηριστικά του προβλήματος

Πέρα από τα γενικά χαρακτηριστικά του προβλήματος, τα οποία περιγράφθηκαν στα προηγούμενα κεφάλαια, είναι αναγκαίο να αναλύσουμε ορισμένες σημαντικές λεπτομέρειες του προβλήματος οι οποίες επηρεάζουν τον τρόπο αντιμετώπισης του.

4.2.1 Αρχική δημιουργία του ευρετηρίου

Η είσοδος του συστήματος δημιουργίας ενός ανεστραμμένου ευρετηρίου αποτελείται από μία συλλογή κειμένων μεγάλου όγκου. Τα κείμενα αυτά πρέπει να σαρωθούν, το καθένα ξεχωριστά, και να καταγραφούν οι λέξεις που αυτά περιέχουν και είναι επιθυμητό να συμπεριληφθούν στο ευρετήριο. Έτσι, ουσιαστικά, για κάθε κείμενο χρειαζόμαστε ένα ID, το οποίο λειτουργεί σαν αναφορά στο κείμενο αυτό, καθώς και το σύνολο των λέξεων που αυτό περιέχει. Τέλος, για κάθε λέξη πρέπει να δημιουργηθεί μια λίστα με τα IDs όλων των κειμένων στα οποία περιέχεται. Για την ταχύτερη ανάκτηση των δεδομένων κατά την επεξεργασία των ερωτημάτων των χρηστών, τα ζεύγη (λέξη, λίστα ID κειμένων) που έχουν δημιουργηθεί πρέπει να αποθηκευτούν σε μία βάση δεδομένων.

Το γεγονός ότι τα κείμενα της συλλογής μπορούν να σαρωθούν ανεξάρτητα μεταξύ τους, αλλά και το γεγονός ότι η λίστα κειμένων για κάθε λέξη μπορεί επίσης να δημιουργηθεί ανεξάρτητα από τις λίστες των υπολοίπων λέξεων, καθιστά εφικτή τη χρήση κατανεμημένων τεχνικών επεξεργασίας, οι οποίες έχουν τη δυνατότητα να επιταχύνουν σημαντικά τη διαδικασία δημιουργίας, αξιοποιώντας τους πόρους ενός υπολογιστικού συστήματος πολλών κόμβων. Για το λόγο αυτό χρησιμοποιούμε το MapReduce framework, το οποίο όπως περιγράψαμε νωρίτερα είναι κατάλληλο για τέτοιου είδους εφαρμογές. Ταυτόχρονα, για την αποθήκευση των τελικών αποτελεσμάτων χρησιμοποιούμε την HBase, η οποία προσφέρει τις δυνατότητες μιας βάσης δεδομένων, αλλά ταυτόχρονα είναι σχεδιασμένη για να λειτουργεί αποδοτικά με το Hadoop MapReduce, χωρίς ιδιαίτερη δυσκολία για τον προγραμματιστή. Ακόμα, η HBase, ως κατανεμημένη βάση δεδομένων επιτρέπει και την κατανομή του φορτίου ερωτημάτων στους διαφορετικούς κόμβους του συστήματος, η οποία είναι σημαντική, δεδομένου του αυξημένου αριθμού χρηστών και ερωτημάτων στο διαδίκτυο.

4.2.2 Ενημέρωση του ευρετηρίου

Η είσοδος του συστήματος ενημέρωσης αποτελείται από μία σχετικά μεγάλη συλλογή κειμένων, τα οποία έχουν προστεθεί στη συλλογή ή έχουν τροποποιηθεί σε σχέση με την έκδοση που έχουμε συμπεριλάβει στο υπάρχον ευρετήριο. Είναι σημαντικό να τονίσουμε ότι με την ενημέρωση του ευρετηρίου επιδιώκουμε να επεξεργαστούμε μόνο τα τροποποιημένα ή νέα κείμενα, αποφεύγοντας την εξ αρχής επεξεργασία όλης της συλλογής κειμένων, της οποίας ο όγκος είναι εξαιρετικά μεγάλος. Επιπλέον, προκειμένου να εξασφαλίσουμε την αποδοτικότητα της ενημέρωσης του ευρετηρίου, είναι απαραίτητο να σχεδιάσουμε μια διαδικασία η οποία να είναι ανεξάρτητη από το μέγεθος του υπάρχοντος ευρετηρίου. Η απαίτηση αυτή είναι ιδιαίτερα σημαντική, αφού σε αντίθετη περίπτωση ακόμα και για μικρό όγκο νέων ή τροποποιημένων κειμένων, η διαδικασία ενημέρωσης θα έχει υψηλό χρόνο εκτέλεσης, με αποτέλεσμα να καθιστά ανέφικτη την συχνή ενημέρωση του ευρετηρίου. Ακόμα, πρέπει να σημειωθεί ότι η διαδικασία ενημέρωσης είναι επιθυμητό να αφήνει το ευρετήριο σε συνεπή μορφή, χωρίς να επηρεάζει τη δομή του ή να κρατά δεσμευμένο αποθηκευτικό χώρο ο οποίος πλέον δεν περιέχει χρήσιμα δεδομένα. Κάτι τέτοιο είναι σημαντικό αφού εξασφαλίζει ότι σε κάθε χρονική στιγμή, ανεξάρτητα από τον αριθμό των ενημερώσεων που έχουν πραγματοποιηθεί, το ευρετήριο βρίσκεται ακριβώς στην κατάσταση που θα ήταν ακόμα και αν δεν είχε πραγματοποιηθεί καμία ενημέρωση και

είχε μόλις κατασκευαστεί εξ αρχής. Με τον τρόπο αυτό εξασφαλίζεται τόσο η λειτουργικότητα του συστήματος, όσο και η σταθερότητα της απόδοσης του, η οποία δεν επηρεάζεται από τον αριθμό των ενημερώσεων.

Και στην περίπτωση αυτή τα νέα ή τροποποιημένα κείμενα πρέπει να σαρωθούν ώστε να αναγνωριστούν οι λέξεις που είναι επιθυμητό να συμπεριληφθούν στο ευρετήριο. Ωστόσο, όσον αφορά τα τροποποιημένα κείμενα της συλλογής, για τα οποία υπάρχουν ήδη εγγραφές στο ευρετήριο, είναι απαραίτητο να διαγραφούν οι εγγραφές που αφορούν την παλαιότερη έκδοση τους και να προστεθούν οι εγγραφές των νέων εκδόσεων, ανάλογα με τις λέξεις που περιέχονται σε αυτές. Αφού διαγραφούν οι εγγραφές των παλαιών εκδόσεων, αρκεί να δημιουργήσουμε και πάλι ζεύγη (λέξη, λίστα από IDs κειμένων) για τα νέα κείμενα και να συνενώσουμε τις λίστες αυτές με τις αντίστοιχες λίστες του υπάρχοντος ευρετηρίου. Αφού γίνει η συνένωση των λιστών, και πάλι τα δεδομένα πρέπει να αποθηκευτούν στη βάση δεδομένων, η οποία πλέον θα περιέχει το ανανεωμένο ευρετήριο.

Κατά την ενημέρωση του ευρετηρίου, ο όγκος των δεδομένων είναι ιδιαίτερα υψηλός με αποτέλεσμα να είναι απαραίτητο να εκτελέσουμε τόσο τη σάρωση των κειμένων, καθώς και η δημιουργία των νέων λιστών, με κατανεμημένο τρόπο. Για το λόγο αυτό, όπως και προηγουμένως, χρησιμοποιούμε το MapReduce framework, ενώ για την αποθήκευση των τελικών αποτελεσμάτων χρησιμοποιούμε την HBase.

Είναι σημαντικό να τονίσουμε ότι η πολυπλοκότητά της διαδικασίας ενημέρωσης εξαρτάται από τη πολυπλοκότητα της διαδικασίας διαγραφής των εγγραφών που αφορούν τις παλιές εκδόσεις των κειμένων. Η διαδικασία αυτή επηρεάζεται σε μεγάλο βαθμό από τον τρόπο αποθήκευσης του ευρετηρίου. Η ακριβής διαδικασία που αναπτύχθηκε, καθώς και ο τρόπος αποθήκευσης του ευρετηρίου αναλύονται διεξοδικά σε επόμενο κεφάλαιο.

4.2.3 Προβλήματα από την Zipfian κατανομή της συχνότητας εμφάνισης των λέξεων – Χρήση δειγματοληψίας για την επίλυση του προβλήματος

Προκειμένου να επιτύχουμε βέλτιστα αποτελέσματα και το μέγιστο δυνατό παραλληλισμό της διαδικασίας, είναι απαραίτητο να διαιρέσουμε ομοιόμορφα τα δεδομένα εισόδου κάθε φάσης επεξεργασίας, ώστε η επεξεργασία να διαρκέσει περίπου το ίδιο για όλα τα τμήματα των δεδομένων.

Σχετική έρευνα[18] έχει καταγράψει το πρόβλημα που δημιουργείται από την άνιση κατανομή των δεδομένων σε μια MapReduce εργασία, τόσο κατά τη φάση του reduce, όσο και κατά τη φάση του map. Συγκεκριμένα έχει παρατηρηθεί ότι σε αρκετές περιπτώσεις ένας μικρός αριθμός από mappers ή/και reducers ολοκληρώνουν την επεξεργασία των δεδομένων σε ένα σημαντικά μεγαλύτερο χρονικό διάστημα σε σχέση με τα υπόλοιπα tasks, επιβραδύνοντας έτσι την ολοκλήρωση της εφαρμογής συνολικά. Σε ορισμένες περιπτώσεις, το πρόβλημα αυτό οφείλεται στα διαφορετικά χαρακτηριστικά του υλικού ορισμένων κόμβων του συστήματος. Ωστόσο, τις περισσότερες φορές, το πρόβλημα οφείλεται στην ανομοιόμορφη κατανομή των δεδομένων εισόδου ή των ενδιάμεσων δεδομένων στα map και reduce tasks αντίστοιχα, η οποία προκαλείται από τη Zipfian[19] κατανομή των κλειδιών των key/value pairs. Αυτό πρακτικά σημαίνει ότι ένας μικρός αριθμός κλειδιών εμφανίζεται με πολύ μεγαλύτερη συχνότητα σε σχέση με το μέσο όρο, με αποτέλεσμα, αν δεν ληφθεί μέριμνα για την ομοιόμορφη κατανομή των key/value pairs, τα tasks που καλούνται να επεξεργαστούν τα ζεύγη με τα συγκεκριμένα κλειδιά να επιβαρύνονται σημαντικά.

Στην αγγλική γλώσσα, στην οποία είναι γραμμένα τα κείμενα της συλλογής που εξετάζουμε, όπως και στις περισσότερες ανθρώπινες γλώσσες, η συχνότητα εμφάνισης των λέξεων ακολουθεί Zipfian κατανομή. Κατά τη φάση του reduce, η πολυπλοκότητα της επεξεργασίας της κάθε λέξης(key) είναι γραμμική ως προς τη συχνότητα εμφάνισης της λέξης και, επομένως, αν διαιρέσουμε το σύνολο τιμών των λέξεων σε διαστήματα ίσου εύρους, οι reducers οι οποίοι αναλαμβάνουν την επεξεργασία των συχνά εμφανιζόμενων λέξεων θα αντιμετωπίσουν το πρόβλημα που περιγράψαμε. Είναι, λοιπόν, απαραίτητο να δώσουμε ιδιαίτερη προσοχή στο διαμοιρασμό των δεδομένων, προκειμένου να επιτύχουμε κατά το δυνατόν παράλληλη εκτέλεση της επεξεργασίας.

Στη φάση του map κάτι τέτοιο δεν είναι ιδιαίτερα δύσκολο, αφού η επεξεργασία είναι γραμμική ως προς το μέγεθος της εισόδου και η συλλογή των κειμένων διαιρείται σε blocks ίσου μεγέθους. Ταυτόχρονα ο αριθμός των map tasks είναι αρκετά μεγάλος σε σχέση με τους επεξεργαστικούς πυρήνες του συστήματος με αποτέλεσμα να επιτυγχάνεται στατιστικά το load balancing χωρίς την παρέμβαση του προγραμματιστή. Αντίθετα, κατά τη φάση του reduce, ο αριθμός των reducers δεν είναι ιδιαίτερα μεγάλος, προκειμένου να εξασφαλιστεί η αποδοτική λειτουργία του framework, με αποτέλεσμα να είναι αναγκαία η

σχεδίαση μιας ειδικής συνάρτησης προκειμένου να επιτευχθεί ο ομοιόμορφος διαχωρισμός των key/value pairs στους reducers.

Προκειμένου να αντιμετωπιστεί το παραπάνω πρόβλημα είναι απαραίτητο να χρησιμοποιήσουμε μια συνάρτηση διαχωρισμού (partitioning function) των ενδιάμεσων key/value pairs, η οποία να λαμβάνει υπ' όψιν της τη μεγάλη διακύμανση της συχνότητας των λέξεων-κλειδιών και να μην μοιράζει απλά τον ίδιο αριθμό λέξεων-κλειδιών σε κάθε reducer, όπως θα έκανε, για παράδειγμα, μια συνάρτηση που θα επέλεγε τον reducer για κάθε κλειδί ανάλογα με το hash του κλειδιού αυτού. Κάτι τέτοιο θα ήταν εφικτό αν είχαμε κάποια αξιόπιστα στατιστικά δεδομένα σχετικά με την συχνότητα εμφάνισης των λέξεων. Δεδομένου όμως ότι οι συχνότητες αυτές μεταβάλλονται ανάλογα με το είδος των κειμένων και τις χρονικές περιόδους συγγραφής των κειμένων (για παράδειγμα ένας πολύ ισχυρός σεισμός, που συνέβη μία συγκεκριμένη ημέρα, θα μπορούσε να οδηγήσει σε τεράστια αύξηση της συχνότητας εμφάνισης της λέξεις 'σεισμός' για τις επόμενες ημέρες), δεν είναι δυνατόν να βασιστούμε σε μία στατική κατανομή της συχνότητας εμφάνισης των λέξεων.

Για το λόγο αυτό επιλέγουμε, πριν την εκτέλεση της εφαρμογής μας, να εκτελέσουμε μία MapReduce εργασία, μικρότερης διάρκειας, με σκοπό την επεξεργασία ενός μικρού, αλλά αντιπροσωπευτικού, δείγματος των κειμένων και, στη συνέχεια, τον προσεγγιστικό υπολογισμό της συχνότητας εμφάνισης της κάθε λέξης. Με τα στοιχεία αυτά, έχουμε τη δυνατότητα να διαιρέσουμε το διάστημα τιμών των λέξεων σε ανεξάρτητα διαστήματα, διαφορετικού εύρους, τέτοια ώστε το σύνολο των λέξεων που ανήκουν σε αυτά να εμφανίζουν αθροιστικά περίπου την ίδια πιθανότητα εμφάνισης. Αναθέτοντας την επεξεργασία των κλειδιών κάθε διαστήματος σε διαφορετικό reducer, οι reducers αναλαμβάνουν την επεξεργασία περίπου του ίδιου αριθμού key/value pairs, με αποτέλεσμα να επιτυγχάνεται παράλληλη επεξεργασία και ο χρόνος εκτέλεσης τους να είναι περίπου ο ίδιος.

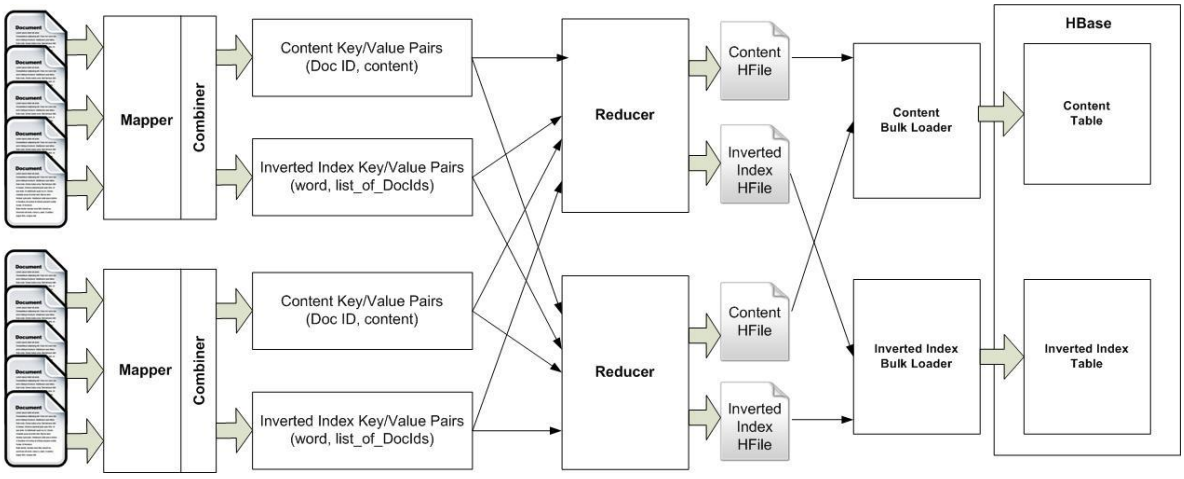
Η ιδέα της δειγματοληψίας έχει χρησιμοποιηθεί και σε άλλες ερευνητικές εργασίες[20],[21] με σκοπό την ομοιόμορφη κατανομή δεδομένων προς ταξινόμηση, καθώς και χωρικών δεδομένων. Η ακριβής διαδικασία της δειγματοληψίας παρουσιάζεται διεξοδικά στα επόμενα κεφάλαια.

4.3 Διαδικασία δημιουργίας του ανεστραμμένου ευρετηρίου

4.3.1 Αρχική δημιουργία του ευρετηρίου αγνοώντας την ανάγκη για αποδοτική ενημέρωση

Προκειμένου να γίνουν καλύτερα κατανοητές οι επιλογές που κάναμε κατά τη σχεδίαση του τελικού συστήματος, αρχικά, παρουσιάζουμε τον τρόπο δημιουργίας ενός ανεστραμμένου ευρετηρίου χρησιμοποιώντας το MapReduce framework και την HBase, χωρίς να λαμβάνουμε υπ' όψιν την ανάγκη για αποδοτική ενημέρωση του ευρετηρίου. Η διαδικασία που ακολουθεί βασίζεται σε σχετική έρευνα[11] από την ομάδα των Κατανεμημένων Συστημάτων του εργαστηρίου Υπολογιστικών Συστημάτων του Ε.Μ.Π. Έχουν γίνει ωστόσο κάποιες τροποποιήσεις για τον συνδυασμό της φόρτωσης του περιεχομένου των κειμένων, αλλά και της δημιουργίας του ανεστραμμένου ευρετηρίου σε μία MapReduce εργασία.

Η αρχική δημιουργία του ευρετηρίου απλουστεύεται σημαντικά, αν αμελήσουμε τη δυνατότητα αποδοτικής ενημέρωσης του ευρετηρίου. Η διαδικασία αυτή παρουσιάζεται γραφικά στο παρακάτω σχήμα. Ο αριθμός των mappers και των reducers έχει επιλεγεί για την καλύτερη απεικόνιση της διαδικασίας και δεν αντιστοιχεί στην πραγματική εκτέλεση της εργασίας.



Σχήμα 7: Διαδικασία δημιουργίας ενός ανεστραμμένου ευρετηρίου, αγνοώντας την ανάγκη για την αποδοτική ενημέρωση του ευρετηρίου

Αναλυτικότερα, η διαδικασία κατασκευής του ευρετηρίου, αγνοώντας την ανάγκη για μελλοντικές ενημερώσεις, είναι η εξής:

Πριν την εκκίνηση της επεξεργασίας, το MapReduce framework αναλαμβάνει να δημιουργήσει ένα map task για κάθε block δεδομένων της συλλογής κειμένων, το οποίο αναλαμβάνει την επεξεργασία των κειμένων που ανήκουν στο αντίστοιχο block. Τα map tasks μοιράζονται στους διαφορετικούς κόμβους της συστοιχίας.

Κάθε map task εκτελεί τον εξής αλγόριθμο:

Αλγόριθμος 2: Αλγόριθμος των mappers για τη δημιουργία ανεστραμμένου ευρετηρίου χωρίς τη δυνατότητα αποδοτικής ενημέρωσης

Σάρωση του block για την αναγνώριση των ανεξάρτητων κειμένων που ανήκουν σε αυτό και δημιουργία key/value pairs (ID κειμένου, περιεχόμενο);
Για κάθε ζεύγος:
    Εξαγωγή ενός key/value pair(ID κειμένου, περιεχόμενο);
    Σάρωση του περιεχομένου για την αναγνώριση των λέξεων που περιέχονται στο κείμενο και επιθυμούμε να συμπεριλάβουμε στο ευρετήριο;
    Για κάθε λέξη που αναγνωρίζεται:
        Εξαγωγή ενός key/value pair(λέξη, ID κειμένου);

Για κάθε key/value pair που εξάγεται, χρησιμοποιείται η partitioning function προκειμένου να προσδιοριστεί ο reducer ο οποίος είναι αρμόδιος για την επεξεργασία του συγκεκριμένου key. Μετά την ολοκλήρωση του κάθε map task εκτελείται (χωρίς ωστόσο να είναι εξασφαλισμένη από το framework η εκτέλεση του) ένας combiner για κάθε map task με σκοπό τη μείωση των δεδομένων που πρέπει να μεταφερθούν από τους mappers στους reducers. Ο combiner ομαδοποιεί τα key/value pair με βάση το κλειδί εκτελώντας τον εξής αλγόριθμο:

Αλγόριθμος 3: Αλγόριθμος των combiners για τη δημιουργία ανεστραμμένου ευρετηρίου χωρίς τη δυνατότητα αποδοτικής ενημέρωσης

Για κάθε ξεχωριστό key που εξήχθη από το map task:
    λίσταID={};
    Για κάθε key/value pair (λέξη, ID κειμένου), με το συγκεκριμένο key:
        λίσταID=λίσταID∪{ID κειμένου};
    Εξαγωγή ενός key/value pair (λέξη, λίσταID);

Μετά την ολοκλήρωση των map tasks τα key/value pairs ομαδοποιούνται και ταξινομούνται ως προς το κλειδί τους και στη συνέχεια διαβάζονται από τους αρμόδιους reducers που έχει ορίσει η partitioning function για κάθε κλειδί. Οι reducers εκτελούν τον εξής αλγόριθμο:

Αλγόριθμος 4: Αλγόριθμος των reducers για τη δημιουργία ανεστραμμένου ευρετηρίου χωρίς τη δυνατότητα αποδοτικής ενημέρωσης

```
Για κάθε ξεχωριστό κλειδί που διαβάζεται:
    τελική_λίσταID={};
    Αν το κλειδί αφορά το inverted index:
        Για κάθε key/value pair (λέξη, λίσταID):
            τελική_λίσταID =τελική_λίσταID ∪ λίσταID;
        Εξαγωγή ενός key/value pair (λέξη, τελική_λίσταID) στην έξοδο που άφορα
        το inverted index;
        //με συγκεκριμένη δομή ώστε η λίσταID να φορτωθεί σαν τιμή κελιού μιας
        //μοναδικής στήλης (με οποιοδήποτε όνομα) για την αντίστοιχη λέξη-γραμμή
    Αλλιώς
        Εξαγωγή ενός key/value pair (ID κειμένου, περιεχόμενο) στην έξοδο που
        άφορα το περιεχόμενο των κειμένων;
        //με συγκεκριμένη δομή ώστε το περιεχόμενο να φορτωθεί σαν τιμή κελιού
        //μιας μοναδικής στήλης (με οποιοδήποτε όνομα) για την γραμμή που
        //αντιστοιχεί στο συγκεκριμένο ID κειμένου
```

Τέλος, χρησιμοποιείται η κατάλληλη συνάρτηση εξόδου, η οποία είναι διαθέσιμη από την HBase και είναι συμβατή με το Hadoop MapReduce, ώστε να αποθηκευτούν τα δεδομένα σε ειδικά αρχεία, τα οποία ονομάζονται HFiles και είναι κατάλληλα για την μετέπειτα μαζική φόρτωσή (bulk loading) των δεδομένων στην HBase. Προτιμάται η τεχνική του bulk loading διότι η σειριακή προσθήκη του κάθε key/value pair στη βάση χρειάζεται σημαντικά περισσότερο χρόνο εξαιτίας της ενημέρωσης του index και άλλων στοιχείων της βάσης για την κάθε εισαγωγή.

Πρέπει να τονίσουμε ότι έχουμε δύο διαφορετικές ροές εξόδου. Μία για τα key/value pairs που αφορούν το ανεστραμμένο ευρετήριο και μία δεύτερη για τα key/value pairs που αφορούν το περιεχόμενο των κειμένων.

Πολυπλοκότητα

Αναλύοντας την πολυπλοκότητα κάθε επιμέρους τμήματος της διαδικασίας, εξαιρώντας την πολυπλοκότητα επεξεργασίας από το ίδιο το framework, καταλήγουμε στα εξής:

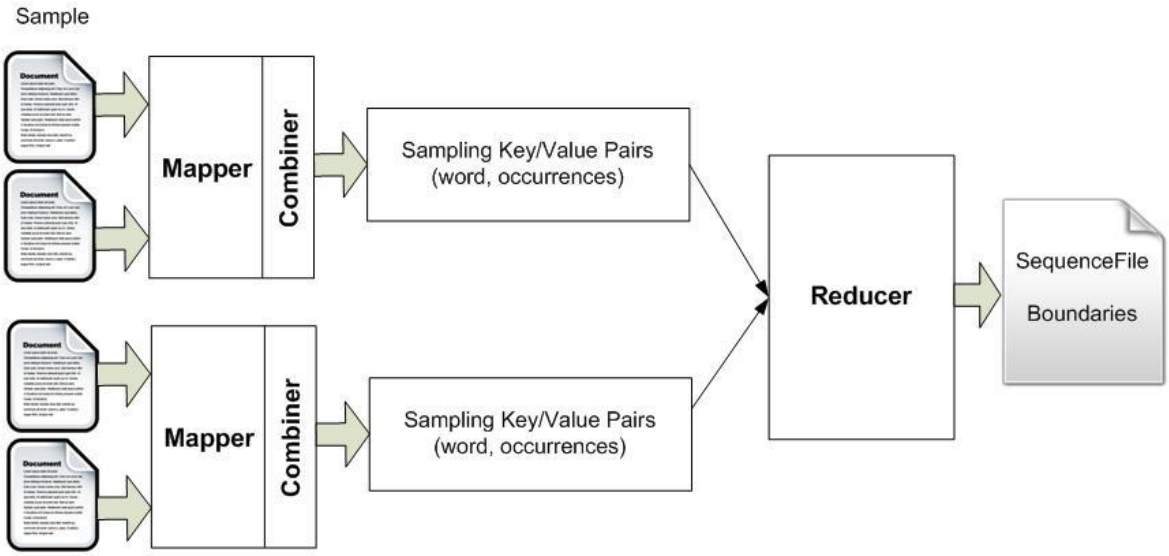
- Η πολυπλοκότητα κατά την ανάγνωση των κειμένων είναι γραμμική ως προς το μέγεθος του αρχείου-block των κειμένων που αναλαμβάνει το κάθε mapper task.
- Η πολυπλοκότητα κατά τη σάρωση του κάθε κειμένου, για την αναγνώριση των λέξεων που αυτό περιέχει, είναι γραμμική ως προς το μέγεθος του κειμένου και επομένως η σάρωση των κειμένων που αντιστοιχούν σε ένα mapper task είναι γραμμική ως προς το μέγεθος του αρχείου κειμένων που του έχει ανατεθεί.
- Η πολυπλοκότητα της κατασκευής των επιμέρους λιστών από τους combiners είναι γραμμική ως προς τον αριθμό των λέξεων των κειμένων που αντιστοιχούν στο συγκεκριμένο mapper task και επομένως γραμμική ως προς το μέγεθος του αντίστοιχου αρχείου κειμένων. Για τα key/value pairs που αφορούν το περιεχόμενο των κειμένων, το κάθε key έχει μοναδικό value και επομένως η επεξεργασία τους είναι γραμμική ως προς τον αριθμό των κειμένων που αντιστοιχούν στο συγκεκριμένο mapper task.
- Η πολυπλοκότητα της κατασκευής των τελικών λιστών από κάθε reducer είναι γραμμική ως προς τον αριθμό των key/value pairs που του έχουν ανατεθεί, τα οποία εξαρτώνται γραμμικά από το συνολικό μέγεθος της συλλογής κειμένων. Για τα key/value pairs που αφορούν το περιεχόμενο των κειμένων, το κάθε key έχει μοναδικό value και επομένως η επεξεργασία τους είναι γραμμική ως προς τον αριθμό των κειμένων της συλλογής που αντιστοιχούν στον κάθε reducer.

Όπως φαίνεται, η υπολογιστική πολυπλοκότητα της εφαρμογής είναι γραμμική ως προς το μέγεθος της εισόδου. Εξαιτίας της χαμηλής πολυπλοκότητας, το κόστος ανάκτησης και μεταφοράς των δεδομένων, το οποίο είναι ιδιαίτερα αυξημένο σε εφαρμογές μαζικής επεξεργασίας (batch processing), όπως αυτή που εξετάζουμε, παίζει σημαντικό ρόλο όσον αφορά το χρόνο εκτέλεσης της διαδικασίας. Η επεξεργασία και η μεταφορά των δεδομένων που εκτελείται από το MapReduce framework, με εξαίρεση την ταξινόμηση των ενδιάμεσων key/value pairs, η οποία, ωστόσο, δεν απαιτεί σημαντικό χρόνο διότι επεξεργάζεται μόνο τα keys των ενδιάμεσων key/value pairs, είναι επίσης γραμμική ως προς το μέγεθος της εισόδου, με αποτέλεσμα να περιμένουμε συνολικά γραμμική εξάρτηση του χρόνου εκτέλεσης σε σχέση με το μέγεθος της εισόδου.

4.3.2 Η σημασία της συνάρτησης partitioning – Περιγραφή της διαδικασίας δειγματοληψίας για την ομοιόμορφη κατανομή των key/value pairs στους reducers

Στην ανάλυση που προηγήθηκε θεωρήσαμε δεδομένη την ύπαρξη της συνάρτησης partitioning που καθορίζει την κατανομή των key/value pairs στους reducers. Ωστόσο, το είδος της συνάρτησης που πρέπει να χρησιμοποιηθεί δεν είναι πάντα προφανές. Όπως περιγράψαμε σε προηγούμενη ενότητα, η συχνότητα εμφάνισης των λέξεων ακολουθεί την κατανομή Zipfian και είναι απαραίτητο να εκτελέσουμε δειγματοληψία προκειμένου να χωρίσουμε το εύρος τιμών των λέξεων σε ανεξάρτητα διαστήματα, διαφορετικού εύρους, τέτοια ώστε να εξασφαλιστεί η ομοιόμορφη κατανομή του φορτίου στους reducers.

Η διαδικασία της δειγματοληψίας αποτελείται από μία MapReduce εργασία η οποία έχει σαν είσοδο ένα δείγμα από τη συλλογή κειμένων και σαν έξοδο τα όρια των διαστημάτων στα οποία πρέπει να διαχωριστεί το εύρος τιμών των λέξεων. Στο παρακάτω σχήμα παρουσιάζεται γραφικά η διαδικασία δειγματοληψίας. Ο αριθμός των mappers του σχήματος είναι ενδεικτικός για την καλύτερη απεικόνιση της διαδικασίας. Αντίθετα, ο αριθμός των reducers αντιστοιχεί στον πραγματικό αριθμό των reducers της διαδικασίας.



Σχήμα 8: Διαδικασία δειγματοληψίας κατά τη δημιουργία του ευρετηρίου

Πιο συγκεκριμένα η διαδικασία είναι η εξής:

- Κάθε mapper αναλαμβάνει ένα τμήμα από το σύνολο Α των κειμένων και σαρώνει ένα υποσύνολο Β από τα κείμενα που έχει αναλάβει, προκειμένου να αναγνωριστούν οι λέξεις που θα συμπεριληφθούν στο ευρετήριο. Επειδή κάθε

mapper αναλαμβάνει ένα ανεξάρτητο υποσύνολο, συγκεκριμένου μεγέθους, του αρχικού συνόλου κειμένων Α, τα δείγματα που επεξεργάζονται συνολικά οι mappers είναι ομοιόμορφα κατανεμημένα μέσα στο σύνολο Α και έτσι το δείγμα μπορεί να θεωρηθεί αντιπροσωπευτικό.

- Για κάθε λέξη που αναγνωρίζεται, ο mapper εξάγει ένα key/value pair της μορφής (λέξη, 1) το οποίο σημαίνει ότι είχαμε μία εμφάνιση της συγκεκριμένη λέξης.
- Ταυτόχρονα με την αναγνώριση των λέξεων του κειμένου, κάθε mapper καταγράφει και τον συνολικό αριθμό των λέξεων του συγκεκριμένου κειμένου, ενώ όταν ολοκληρωθεί η σάρωση του κειμένου εξάγει ένα key/value pair της μορφής (πολύ_μικρό_κλειδί, αριθμός λέξεων). Το πολύ_μικρό_κλειδί αναπαριστά μια συμβολοσειρά αλφαβητικά μικρότερη από κάθε πιθανή λέξη που μπορεί να αναγνωριστεί στα κείμενα της συλλογής και έχει σκοπό να ενημερώσει τον reducer για το συνολικό αριθμό των λέξεων των κειμένων. Με τον τρόπο αυτό δεν είναι αναγκαία η διπλή σάρωση των key/value pairs που φτάνουν στον reducer, με αποτέλεσμα να επιταχύνεται η διαδικασία.
- Ο combiner αθροίζει τις τιμές των values για κάθε κλειδί, και εξάγει ένα key/value pair της μορφής (λέξη ή πολύ_μικρο_κλειδί, άθροισμα values) για κάθε κλειδί. Με τον τρόπο αυτό μειώνεται ο όγκος των δεδομένων που πρέπει να μεταφερθούν στον reducer καθώς και ο αριθμός των πράξεων που θα πρέπει αυτός να εκτελέσει.
- Ο reducer, ο οποίος στη συγκεκριμένη διαδικασία είναι μόνο ένας, αρχικά λαμβάνει τα key/value pairs (πολύ_μικρό_κλειδί, αριθμός λέξεων) ώστε να υπολογίσει τον συνολικό αριθμό των λέξεων των κειμένων του δείγματος. Διαιρώντας τον αριθμό αυτό με τον αριθμό των reducers που θα χρησιμοποιηθούν στην εφαρμογή δημιουργίας του ευρετηρίου, υπολογίζεται ο αριθμός των λέξεων που πρέπει να αναλάβει κάθε reducer. Καθώς διαβάζονται τα key/value pairs για τις λέξεις των κειμένων, ο reducer αθροίζει τις εμφανίσεις των λέξεων μέχρι αυτές να ξεπεράσουν το όριο λέξεων ανά reducer. Όταν συμβεί αυτό ο reducer επιλέγει αν θα συμπεριλάβει την τελευταία λέξη στο διάστημα που δημιουργεί, εξάγει ένα key/value pair με την τιμή του ορίου του συγκεκριμένου διαστήματος και συνεχίζει με την επεξεργασία των επόμενων λέξεων, ώστε να ορίσει τα όρια των επόμενων διαστημάτων.

Συγκεκριμένα ο αλγόριθμος που εκτελείται στον reducer για τα keys εκτός του ‘πολύ_μικρό_κλειδί’, το οποίο χρησιμοποιείται για τον ταχύτερο υπολογισμό του συνολικού αριθμού των λέξεων, είναι ο εξής:

Αλγόριθμος 5: Αλγόριθμος των reducers της εργασίας δειγματοληψίας

```
αριθμός_λεξεων_ανα_reducer=συνολικός_αριθμός_λέξεων/αριθμός_reducers;
τρέχον_όριο=0;
τρέχον_αριθμός_λέξεων=0;
Για κάθε key:
    Αν (τρέχον_όριο == αριθμός_reducers-1)
        Αγνόησε όλα τα keys από εδώ και έπειτα;

    αριθμός_λέξεων=άθροισμα(values) //δηλαδή άθροισμα των εμφανίσεων της
                                     //συγκεκριμένης λέξης
    Αν (τρέχον_αριθμός_λέξεων+ αριθμός _λέξεων< αριθμός_λεξεων_ανα_reducer)
    //δηλαδή αν δεν ξεπεράσαμε το όριο λέξεων για τον συγκεκριμένο reducer
        τρέχον_αριθμός_λέξεων= τρέχον_αριθμός_λέξεων+ αριθμός _λέξεων;

    Αλλιώς
        Αν (τρέχον_αριθμός_λέξεων+ αριθμός _λέξεων-
            αριθμός_λεξεων_ανα_reducer > αριθμός_λεξεων_ανα_reducer-
            τρέχον_αριθμός_λέξεων)
            //αν δηλαδή είμαστε πιο κοντά στο όριο λέξεων ανά reducer όταν
            //δεν συμπεριλαμβάνουμε τη συγκεκριμένη λέξη
                όριο[τρέχον_όριο]=key;
                τρέχον_όριο ++;
                συνολικός_αριθμός_λέξεων= συνολικός_αριθμός_λέξεων-
                                           τρέχον_αριθμός_λέξεων;
                αριθμός_λεξεων_ανα_reducer= συνολικός_αριθμός_λέξεων/
                                            ( αριθμός_reducers- τρέχον_όριο);
                τρέχον_αριθμός_λέξεων= αριθμός _λέξεων;
        Αλλιώς
                όριο[τρέχον_όριο]=επόμενο key;
                τρέχον_όριο ++;
```

```
τρέχον_αριθμός_λέξεων= τρέχον_αριθμός_λέξεων+ αριθμός _λέξεων;
συνολικός_αριθμός_λέξεων= συνολικός_αριθμός_λέξεων-
                                τρέχον_αριθμός_λέξεων;
αριθμός_λεξεων_ανα_reducer= συνολικός_αριθμός_λέξεων/
                                ( αριθμός_reducers- τρέχον_όριο);
τρέχον_αριθμός_λέξεων=0;
```

Με την εκτέλεση της παραπάνω MapReduce εργασίας δειγματοληψίας, έχουμε καθορίσει τα όρια των διαστημάτων στα οποία πρέπει να διαιρεθεί το εύρος τιμών των λέξεων προκειμένου να έχουμε ομοιόμορφη κατανομή των key/value pairs στους reducers. Η συνάρτηση partitioning αρχικά διαβάζει τα όρια αυτά και, στη συνέχεια, για κάθε κλειδί επιλέγει τον κατάλληλο reducer, με βάση το διάστημα στο οποίο ανήκει το συγκεκριμένο κλειδί. Συγκεκριμένα για R reducers διαχωρίζουμε το εύρος τιμών σε R διαστήματα, τα οποία έχουνε μεταξύ τους R-1 όρια τα οποία συμβολίζουμε με Bound[i]. Η συνάρτηση partitioning, με βάση τον παραπάνω συμβολισμό είναι η εξής:

$$f(key) = \begin{cases} i-1, & \alpha v\ key < Bound[i-1], i \in [1, R-2] \\ R-1, & \alpha v\ key > Bound[R-1] \end{cases}$$

Όπου key η τιμή του κλειδιού για το οποίο αναζητείται ο αρμόδιος reducer.

Η διαδικασία δειγματοληψίας που περιγράψαμε είναι εξαιρετικά αποτελεσματική και επιτυγχάνει τον διαχωρισμό του συνόλου τιμών των λέξεων σε διαστήματα τέτοια ώστε το φορτίο των αντίστοιχων reducers να είναι σχεδόν ίδιο. Η αποτελεσματικότητα της αποδεικνύεται και πειραματικά, αφού παρά την Zipfian κατανομή των λέξεων, ο χρόνος ολοκλήρωσης όλων των reduce tasks, κατά την δημιουργία του ευρετηρίου, είναι περίπου ίδιος.

Ωστόσο, η παραπάνω διαδικασία δεν αντιμετωπίζει το πρόβλημα της ομοιόμορφης κατανομής των key/value pairs που αφορούν τη φόρτωση του περιεχομένου των κειμένων στην HBase. Για το λόγο αυτό είναι απαραίτητο να τροποποιήσουμε τη συνάρτηση partitioning ώστε να έχει την επιθυμητή συμπεριφορά και για τα key/value pairs που έχουν σαν key το ID ενός κειμένου.

Οι δύο κατηγορίες από key/value pairs είναι ανεξάρτητες μεταξύ τους με αποτέλεσμα να μπορούμε να θεωρήσουμε την ομοιόμορφη κατανομή της κάθε μίας σαν ένα ξεχωριστό

πρόβλημα. Κάθε key/value pair που αφορά τη φόρτωση του περιεχομένου των κειμένων έχει σαν key το ID του αντίστοιχου κειμένου. Επομένως, υπάρχει ένα μόνο pair για κάθε κείμενο, σε αντίθεση με τα key/value pairs που αφορούν τη δημιουργία του ανεστραμμένου ευρετηρίου, όπου για κάθε key έχουμε έναν μεταβλητό αριθμό από pairs, οποίος μάλιστα μπορεί να είναι εξαιρετικά μεγάλος. Για το λόγο αυτό, δεν είναι απαραίτητο να εκτελέσουμε δειγματοληψία για την ομοιόμορφη κατανομή των συγκεκριμένων δεδομένων στους reducers. Αντίθετα, αν θεωρήσουμε ότι τα IDs είναι ομοιόμορφα κατανεμημένα στο εύρος τιμών τους, μπορούμε να διαιρέσουμε το εύρος τιμών σε διαστήματα ίσου πλάτους και, με τον τρόπο αυτό, να επιτύχουμε τα επιθυμητά αποτελέσματα. Ωστόσο, επειδή δεν έχουμε τη δυνατότητα να είμαστε βέβαιοι ότι αυτή η υπόθεση είναι αληθής, επιλέγουμε, αντί για το πραγματικό ID των κειμένων, να χρησιμοποιήσουμε ένα ‘εικονικό’ ID το οποίο δημιουργείται με τη χρήση μιας συγκεκριμένης συνάρτησης, η οποία την ομοιόμορφη κατανομή των εικονικών IDs.

Στο σημείο αυτό πρέπει να επισημάνουμε ότι κατά την εισαγωγή των δεδομένων στην HBase, το κλειδί κάθε γραμμής αντιμετωπίζεται σαν συμβολοσειρά και, επομένως, η ταξινόμηση γίνεται αλφαβητικά. Έτσι, στη MapReduce εργασία που εκτελούμε, είναι απαραίτητο να διαχειριστούμε και να ταξινομήσουμε τα IDs των κειμένων σαν συμβολοσειρές και όχι σαν αριθμούς, ώστε τελικά να εισαχθούν ταξινομημένα στην HBase. Η διαδικασία που ακολουθεί βασίζεται στην απαίτηση αυτή και αντιμετωπίζει τα IDs των κειμένων σαν συμβολοσειρές.

Θεωρώντας ότι τα IDs είναι, αρχικά, ακέραιοι αριθμοί, σχεδιάσαμε μια συνάρτηση η οποία ανακατατάσσει τα ψηφία των IDs, προκειμένου να επιτύχει την ομοιόμορφη κατανομή τους. Πιο συγκεκριμένα, θεωρήσαμε ότι το τελευταίο ψηφίο των IDs θα είναι με μεγάλη πιθανότητα ομοιόμορφα κατανεμημένο στο διάστημα 0-9 και επομένως αν θέσουμε το ψηφίο αυτό σαν πρώτο ψηφίο του εικονικού ID και ολισθήσουμε τα υπόλοιπα ψηφία μία θέση δεξιά, θα μπορούμε να διαιρέσουμε το εύρος των εικονικών IDs σε 10 ανεξάρτητα διαστήματα τα οποία να περιέχουν περίπου τον ίδιο αριθμό από εικονικά IDs στο εσωτερικό τους. Ωστόσο, επειδή ο αριθμός των reducers δεν είναι ίσος με 10, αλλά μπορεί να είναι αρκετά μεγαλύτερος, επεκτείναμε τη διαδικασία αυτή για όλα τα ψηφία των IDs των κειμένων, θεωρώντας ότι τα ψηφία μικρότερη τάξης έχουν μεγαλύτερη πιθανότητα να είναι ομοιόμορφα κατανεμημένα. Έτσι, καταλήξαμε στην αντιστροφή των ψηφίων των IDs.

Ο αλγόριθμος της συνάρτησης που υλοποιεί την παραπάνω διαδικασία είναι ο εξής:

Αλγόριθμος 6: Αλγόριθμος για τη δημιουργία εικονικών ID με ομοιόμορφη κατανομή

εικονικό_ID="";
Όσο υπάρχουν ψηφία στο ID:
    τελευταίο_ψηφίο=αφαίρεσε_το_τελευταίο_ψηφίο(ID);
    εικονικό_ID=εικονικό ID+τελευταίο_ψηφίο;

Στο παρακάτω σχήμα παρουσιάζονται ορισμένα παραδείγματα της σχέσης μεταξύ ενός ID και του εικονικού ID που δημιουργήθηκε από αυτό:

Πραγματικό ID	Εικονικό ID
103926	629301
9018	8109

Σχήμα 9: Παραδείγματα της σχέσης μεταξύ πραγματικών και εικονικών ID

Με τη χρήση των εικονικών IDs εξασφαλίζουμε την ομοιόμορφη κατανομή των IDs, και επομένως διαιρώντας το εύρος τιμών τους (το οποίο κυμαίνεται μεταξύ "0000…" και "9999…") σε R ανεξάρτητα υποδιαστήματα ίσου πλάτους, όπου R ο αριθμός των reducers της εργασίας δημιουργίας του ευρετηρίου, εξασφαλίζουμε την ομοιόμορφη κατανομή των key/value pairs στους reducers.

Η partitioning function διαβάζει, τόσο τα όρια που έχουν δημιουργηθεί μέσα από τη διαδικασία της δειγματοληψίας και αφορούν τα key/value pairs του ανεστραμμένου ευρετηρίου, όσο και τα όρια των διαστημάτων για τα key/value pairs που αφορούν τη φόρτωση του περιεχομένου των κειμένων. Όταν, λοιπόν, χρειαστεί να επιλέξει τον αρμόδιο reducer για ένα key/value pair, αναγνωρίζει αν αυτό αφορά το περιεχόμενο των κειμένων ή το ανεστραμμένο ευρετήριο και χρησιμοποιώντας τα αντίστοιχα όρια επιλέγει τον αρμόδιο reducer. Με τον τρόπο αυτό επιτυγχάνουμε την ομοιόμορφη κατανομή του φορτίου στους reducers.

4.3.3 Μειονεκτήματα της μεθόδου - Απαιτήσεις για αποδοτικότερη ενημέρωση του ευρετηρίου

Η διαδικασία που περιγράφτηκε στην προηγούμενη ενότητα, παρότι είναι ιδιαίτερα αποδοτική, αφού εξασφαλίζει την ελάχιστη δυνατή πολυπλοκότητα και ταυτόχρονα την

ελαχιστοποίηση των μεταφερόμενων δεδομένων, παρουσιάζει ένα σημαντικό μειονεκτήματα, το οποίο καθιστά προβληματική, αφενός, την ανάκτηση των δεδομένων για την επεξεργασία των ερωτημάτων των χρηστών και, αφετέρου, τη δυνατότητα αποδοτικής ενημέρωσης του ευρετηρίου.

Το μειονέκτημα της παραπάνω μεθόδου αφορά τον τρόπο αποθήκευσης των τελικών αποτελεσμάτων. Όπως περιγράψαμε προηγουμένως, για κάθε λέξη-κλειδί εξάγουμε ένα key/value pair της μορφής (λέξη, λίσταID). Για κάθε λέξη, δηλαδή αποθηκεύουμε μία ενιαία λίστα με τα ID των κειμένων που την περιέχουν. Τα προβλήματα που δημιουργούνται από την επιλογή αυτή είναι τα εξής:

- Κατά την απάντηση των ερωτημάτων των χρηστών, για κάθε λέξη που αναζητά ο χρήστης είναι απαραίτητο να ανακτήσουμε ολόκληρη τη λίστα των IDs των κειμένων, με μία μοναδική πρόσβαση στη βάση δεδομένων. Για τις συχνά εμφανιζόμενες λέξεις, οι οποίες εμφανίζονται σχεδόν σε κάθε κείμενο της συλλογής, η λίστα αυτή θα είναι εξαιρετικά μεγάλη, γεγονός που καθιστά χρονοβόρα την ανάκτησή της, καταναλώνοντας, ταυτόχρονα, ένα υψηλό ποσοστό των πόρων του συστήματος. Εξάλλου, όταν ένας χρήστης εκτελεί μια αναζήτηση, δεν έχει την απαίτηση να του επιστραφούν άμεσα όλα τα κείμενα που περιέχουν τις λέξεις που αναζητά. Τα αποτελέσματα μπορούν να του επιστρέφονται σταδιακά ανά ομάδες, ώστε να έχει και τη δυνατότητα να τα επεξεργαστεί.
- Η διαδικασία ενημέρωσης του ευρετηρίου απαιτεί την διαγραφή των εγγραφών που αφορούν την παλιά έκδοση των κειμένων. Αυτό σημαίνει ότι για κάθε λέξη που περιείχαν τα παλιά κείμενα είναι απαραίτητο να ανακτήσουμε ολόκληρη τη λίστα των IDs, να τη διασχίσουμε, ώστε να διαγράψουμε τις κατάλληλες αναφορές, και να την αποθηκεύσουμε ξανά. Αυτό επιβραδύνει σημαντικά τη διαδικασία ενημέρωσης η οποία επηρεάζεται σημαντικά από το μέγεθος των λιστών και άρα από το μέγεθος του υπάρχοντος ευρετηρίου. Ειδικά στην περίπτωση που η νέα συλλογή κειμένων είναι μικρή, γεγονός το οποίο είναι συχνό όταν θέλουμε να έχουμε συχνές ενημερώσεις με σκοπό την παροχή ‘φρέσκων’ δεδομένων στους χρήστες, ο φόρτος από τη διάσχιση των λιστών του υπάρχοντος ευρετηρίου είναι εξαιρετικά μεγάλος σε σχέση με τον αριθμό των εγγραφών που επιθυμούμε να διαγράψουμε.

Για του λόγους αυτούς ο συγκεκριμένος τρόπος αποθήκευσης του ευρετηρίου κρίνεται ακατάλληλος και είναι αναγκαίο να αναζητήσουμε μια εναλλακτική μέθοδο η οποία να

επιλύει τα παραπάνω προβλήματα, χωρίς ωστόσο να επηρεάζει σημαντικά την αποδοτικότητα της εφαρμογής.

Ο νέος τρόπος αποθήκευσης πρέπει να εξασφαλίζει:

- Τη δυνατότητα ανάκτησης ενός μικρού τμήματος της λίστας για κάθε λέξη του ευρετηρίου.
- Ενδεχομένως τη δυνατότητα να γνωρίζουμε, με τη βοήθεια κάποιας συνάρτησης ή κάποιας βοηθητικής δομής, το τμήμα της λίστας στο οποίο έχει αποθηκευτεί το ID ενός συγκεκριμένου κειμένου για την κάθε λέξη, ώστε να μην απαιτείται η διάσχιση όλης της λίστας στην περίπτωση που επιθυμούμε να διαγράψουμε έναν μικρό αριθμό εγγραφών.

Εξασφαλίζοντας τις παραπάνω προδιαγραφές, θα μπορούσαμε να επιταχύνουμε σημαντικά την ανάκτηση των απαραίτητων δεδομένων για την απάντηση των ερωτημάτων των χρηστών, αλλά και τη διαδικασία ενημέρωσης του ευρετηρίου όταν ένα μέρος των κειμένων της αρχικής συλλογής έχει τροποποιηθεί.

4.3.4 Νέος τρόπος αποθήκευσης του ευρετηρίου με στόχο την αποδοτική ενημέρωση του

Με βάση την ανάλυση που προηγήθηκε, είναι αναγκαίο να τροποποιήσουμε τον τρόπο αποθήκευσης του ευρετηρίου κατά τη δημιουργία του, ώστε να είναι εφικτή η ανάκτηση μικρών τμημάτων της λίστας για κάθε λέξη, καθώς και να μπορούμε να προσδιορίζουμε το τμήμα της κάθε λίστας όπου είναι αποθηκευμένο το ID του κάθε κειμένου. Για το σκοπό αυτό είναι απαραίτητο να διαιρέσουμε την κάθε λίστα σε μικρότερα τμήματα, κάθε ένα από τα οποία θα περιέχει έναν συγκεκριμένο αριθμό από IDs κειμένων. Η λύση αυτή, ωστόσο, δεν εξασφαλίζει την δυνατότητα να γνωρίζουμε σε ποιο τμήμα της κάθε λίστας θα ανήκει το ID του κάθε κειμένου και θα ήταν απαραίτητη η χρήση μιας συνάρτησης ή μιας επιπλέον δομής δεδομένων, για τον προσδιορισμό του τμήματος αυτού.

Στο σημείο αυτό, είναι σημαντικό να υπενθυμίσουμε και να τονίσουμε τη δυνατότητα της HBase να αποθηκεύει έναν ιδιαίτερα μεγάλο αριθμό στηλών για την κάθε γραμμή, ο οποίος μάλιστα μπορεί να διαφέρει για την κάθε γραμμή χωρίς αυτό να αυξάνει το κόστος αποθήκευσης ή ανάκτησης των δεδομένων. Η δυνατότητα αυτή μας επιτρέπει να φτάσουμε την ιδέα της διαίρεσης της κάθε λίστας σε τμήματα στα όρια της, χρησιμοποιώντας μία

στήλη για κάθε στοιχείο της λίστας, δηλαδή για κάθε ID κειμένου που περιέχεται στη λίστα. Ακόμα, το όνομα της στήλης, σύμφωνα με τα χαρακτηριστικά της HBase, μπορεί να είναι οποιαδήποτε συμβολοσειρά, γεγονός που μας επιτρέπει να χρησιμοποιήσουμε το ίδιο το ID του κειμένου σαν όνομα της αντίστοιχης στήλης. Έτσι, για κάθε λέξη μπορούμε να γνωρίζουμε τα κείμενα στα οποία αυτή περιέχεται με βάση τα ονόματα των στηλών που διαθέτει η αντίστοιχη γραμμή του πίνακα της HBase.

Με τον τρόπο αυτό έχουμε τη δυνατότητα:

- Να ανακτήσουμε έναν οποιοδήποτε αριθμό από ID κειμένων από τη λίστα οποιασδήποτε λέξης, εφαρμόζοντας τα κατάλληλα φίλτρα που προσφέρει η HBase και επιτρέπουν η ενέργεια αυτή να πραγματοποιηθεί με αποδοτικό τρόπο.
- Να έχουμε άμεση πρόσβαση στην εγγραφή της HBase που υποδηλώνει την παρουσία ενός συγκεκριμένου ID κειμένου στη λίστα μιας λέξης, με σκοπό την διαγραφή ή την τροποποίηση της. Η εγγραφή αυτή βρίσκεται στη γραμμή με όνομα την ίδια τη λέξη και στη στήλη με όνομα το ID του κειμένου.
- Να γνωρίζουμε με μία μόνο πρόσβαση στη βάση και χωρίς επιπλέον επεξεργασία αν ένα κείμενο περιέχει μια συγκεκριμένη λέξη.

Με τον παραπάνω τρόπο αποθήκευσης εξασφαλίζονται, λοιπόν, οι απαιτήσεις, που προσδιορίστηκαν στην προηγούμενη ενότητα, για την αποδοτική ενημέρωση του ευρετηρίου. Ταυτόχρονα, ο χρόνος δημιουργίας παραμένει περίπου σταθερός, αφού ο όγκος των δεδομένων που επεξεργάζεται η MapReduce εργασία, καθώς και ο όγκος των ενδιάμεσων δεδομένων δεν μεταβάλλεται. Παρόλα αυτά, ο νέος αυτός τρόπος αποθήκευσης οδηγεί στην αύξηση του αποθηκευτικού χώρου που απαιτείται για την αποθήκευση του ευρετηρίου, αφού για κάθε στήλη είναι απαραίτητη η αποθήκευση επιπλέον δεδομένων στην HBase, όπως για παράδειγμα το timestamp του κελιού.

4.3.5 Αποθήκευση του forward index της συλλογής με σκοπό την επιτάχυνση της διαδικασίας ενημέρωσης

Προτού περιγράψουμε αναλυτικά την τροποποιημένη διαδικασία κατασκευής του ευρετηρίου, είναι απαραίτητο να αναφέρουμε ότι για την ταχύτερη ενημέρωση του ευρετηρίου επιλέγουμε να αποθηκεύσουμε επιπλέον δεδομένα για κάθε κείμενο της συλλογής. Τα επιπλέον δεδομένα που αποθηκεύουμε είναι, ουσιαστικά, το ευθύ ευρετήριο (forward index) της συλλογής των κειμένων, το οποίο περιέχει για κάθε κείμενο μια λίστα

από τις λέξεις που περιέχονται στο κείμενο αυτό. Το παρακάτω σχήμα δίνει μια γραφική αναπαράσταση της μορφής του forward index:

Document ID	**Words**
Doc1	the, elephant, is, happy
Doc2	it, is, too, late
Doc3	tomorrow, we, will, go, to ,the, beach

Σχήμα 10: Παράδειγμα forward index

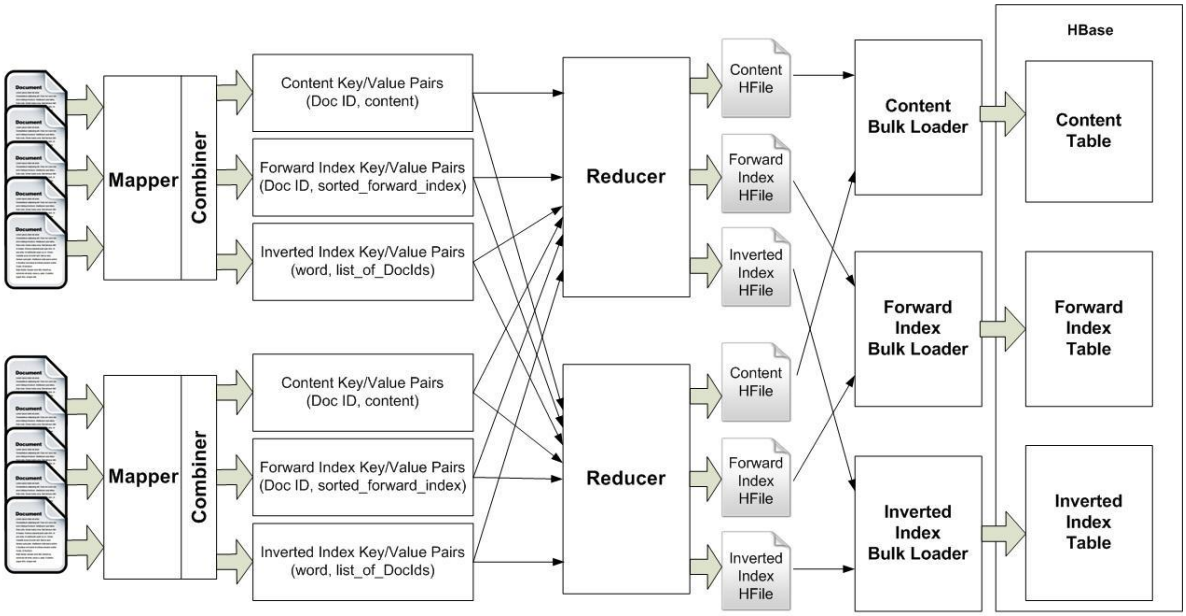
Η χρήση του forward index επιτρέπει την ανάκτηση των λέξεων της παλιάς έκδοσης του κειμένου, χωρίς να πρέπει να σαρωθεί ξανά η έκδοση αυτή, γεγονός που θα αύξανε το κόστος επεξεργασίας. Προκειμένου, μάλιστα, να επιταχύνουμε τη σύγκριση της παλαιάς έκδοσης του κάθε κειμένου με τη νεότερη επιλέγουμε να αποθηκεύσουμε στο forward index τις λέξεις κάθε κειμένου ταξινομημένες αλφαβητικά. Ο λόγος που έγινε αυτή η επιλογή, καθώς και ο ακριβής τρόπος σύγκρισης των δύο διαφορετικών εκδόσεων παρουσιάζονται αναλυτικά στην επόμενη ενότητα που αναφέρεται στην διαδικασία ενημέρωσης του ευρετηρίου. Η δημιουργία του forward index δεν απαιτεί σημαντική αύξηση της επεξεργασίας αφού κατά τη σάρωση των κειμένων έχουμε διαθέσιμες τις λέξεις κάθε κειμένου και είναι απλά αναγκαίο να τις ταξινομήσουμε, διαδικασία η οποία δεν αυξάνει σημαντικά την υπολογιστική πολυπλοκότητα αφού ο αριθμός των λέξεων που περιέχει ένα κείμενο είναι σχετικά μικρός (ειδικά στην περίπτωση που επιλέξουμε να μην συμπεριλάβουμε στο ευρετήριο όλα τα τμήματα, αλλά μόνο τον τίτλο ή τα ονόματα των συγγραφέων, για παράδειγμα). Ταυτόχρονα, οι απαιτήσεις σε αποθηκευτικό χώρο δεν αυξάνονται σημαντικά, για τον ίδιο ακριβώς λόγο, και έτσι το μέγεθος του forward index είναι αρκετά μικρότερο από το συνολικό μέγεθος των κειμένων της συλλογής. Το γεγονός αυτό καθιστά την χρήση του forward index μία καλή επιλογή για την επιτάχυνση της διαδικασίας ενημέρωσης του ανεστραμμένου ευρετηρίου. Το forward index που δημιουργείται αποθηκεύεται και αυτό στην HBase προκειμένου να έχουμε ταχύτερη πρόσβαση σε αυτό.

4.3.6 Τροποποιημένη διαδικασία αρχικής δημιουργίας του ευρετηρίου με στόχο την αποδοτική ενημέρωση του

Με βάση τις προδιαγραφές που αναλύθηκαν στις προηγούμενες ενότητες, τροποποιούμε την διαδικασία αρχικής δημιουργίας του ευρετηρίου, έχοντας ως στόχο τη δυνατότητα αποδοτικής ενημέρωσης του μετά από την μεταβολή του περιεχομένου ενός ποσοστού των

κειμένων της συλλογής και την προσθήκη νέων κειμένων. Η γενική ιδέα δεν διαφέρει σημαντικά σε σχέση με την αρχική διαδικασία, ωστόσο υπάρχουν ορισμένες καίριες αλλαγές που αφορούν κυρίως τον τρόπο αποθήκευσης του ευρετηρίου, αλλά και την δημιουργία του forward index.

Στο παρακάτω σχήμα παρουσιάζεται γραφικά η τροποποιημένη διαδικασία δημιουργίας του ευρετηρίου, η οποία αναλύεται διεξοδικά στη συνέχεια. Ο αριθμός των mappers και των reducers έχει επιλεγεί για την καλύτερη απεικόνιση της διαδικασίας και δεν αντιστοιχεί στην πραγματική εκτέλεση της εργασίας.



Σχήμα 11: Τροποποιημένη διαδικασία δημιουργίας ανεστραμμένων ευρετηρίων

Αναλυτικά η τροποποιημένη διαδικασία αρχικής κατασκευής του ανεστραμμένου ευρετηρίου είναι η εξής:

Πριν την εκκίνηση της επεξεργασίας, το MapReduce framework αναλαμβάνει να δημιουργήσει ένα map task για κάθε block δεδομένων από αυτά που αποτελούν τη συλλογή των κειμένων, το οποίο αναλαμβάνει την επεξεργασία των κειμένων που ανήκουν στο block αυτό. Τα map tasks μοιράζονται στους διαφορετικούς κόμβους της συστοιχίας.

Κάθε map task εκτελεί τον εξής αλγόριθμο:

Αλγόριθμος 7: Αλγόριθμος των mappers για τη δημιουργία ανεστραμμένου ευρετηρίου με τη δυνατότητα αποδοτικής ενημέρωσης

Σάρωση του block για την αναγνώριση των ανεξάρτητων κειμένων που ανήκουν σε αυτό και δημιουργία key/value pairs (ID κειμένου, περιεχόμενο);
Για κάθε key/value pair:
    Εξαγωγή ενός key/value pair(ID κειμένου, περιεχόμενο κειμένου);
    Δημιουργία ενός άδειου δέντρου ταξινόμησης;
    Σάρωση του περιεχομένου για την αναγνώριση των λέξεων που περιέχονται στο κείμενο και επιθυμούμε να συμπεριλάβουμε στο ευρετήριο;
    Για κάθε λέξη που αναγνωρίζεται:
        Εξαγωγή ενός key/value pair(λέξη, ID κειμένου);
        Προσθήκη της λέξης στο δέντρο ταξινόμησης;
    Διάσχιση του δέντρου ταξινόμησης σε αύξουσα σειρά των στοιχείων του;
    Για κάθε λέξη-στοιχείο του δέντρου:
        Προσθήκη της λέξης σε έναν buffer με όνομα sorted_forward_index;
    Εξαγωγή ενός key/value pair(ID κειμένου, sorted_forward_index);

Για κάθε key/value pair που εξάγεται, χρησιμοποιείται η συνάρτηση partitioning προκειμένου να προσδιοριστεί ο reducer ο οποίος είναι αρμόδιος για την επεξεργασία του συγκεκριμένου key. Μετά την ολοκλήρωση του κάθε map task εκτελείται ένας combiner για κάθε map task με σκοπό την μείωση των δεδομένων που πρέπει να μεταφερθούν από τον κάθε mapper προς τους reducers. Η διαφορά με την διαδικασία που περιγράψαμε αρχικά είναι ότι στον combiner φτάνουν και key/value pairs που αφορούν το forward index της συλλογής. Ωστόσο, κάθε ένα από τα key/value pairs αυτά έχει μοναδικό key, αφού έχουμε ένα pair για κάθε κείμενο, με αποτέλεσμα να μην απαιτείται η τροποποίηση του combiner για τη διαχείριση τους. Ο αλγόριθμος που χρησιμοποιείται από τους combiners, λοιπόν, είναι ίδιος με αυτόν που παρουσιάστηκε στην προηγούμενη ενότητα με αποτέλεσμα να μην είναι απαραίτητη η περιγραφή του.

Μετά την ολοκλήρωση των map tasks τα key/value pairs ομαδοποιούνται και ταξινομούνται ως προς το κλειδί τους και στη συνέχεια διαβάζονται από τους reducers που

έχει ορίσει η συνάρτηση partitioning για κάθε κλειδί. Οι reducers εκτελούν τον εξής αλγόριθμο:

Αλγόριθμος 8: Αλγόριθμος των reducers για τη δημιουργία ανεστραμμένου ευρετηρίου με τη δυνατότητα αποδοτικής ενημέρωσης

```
Για κάθε ξεχωριστό κλειδί που διαβάζεται:
    Αν το κλειδί αφορά το inverted index:
        Για κάθε key/value pair με τη συγκεκριμένη λέξη-κλειδί:
            Διάσχιση της λίστας από ID κειμένων;
            Για κάθε στοιχείο-ID της λίστας:
                Εξαγωγή ενός key/value pair (λέξη, ID κειμένου) στην έξοδο
                που αφορά το inverted index;
                //με συγκεκριμένη δομή ώστε το ID να φορτωθεί σαν το
                //όνομα της στήλης και όχι σαν τιμή κελιού
    Αν το κλειδί αφορά το forward index:
        Εξαγωγή ενός key/value pair (ID κειμένου, sorted_forward_index)
        στην έξοδο που αφορά το forward index;
        //με συγκεκριμένη δομή ώστε το sorted_forward_index να φορτωθεί σαν
        //τιμή κελιού μιας μοναδικής στήλης (με οποιοδήποτε όνομα) για το
        //αντίστοιχο ID-γραμμή
    Αλλιώς:
        Εξαγωγή ενός key/value pair (ID κειμένου, περιεχόμενο) στην έξοδο που
        αφορά το περιεχόμενο των κειμένων;
        //με συγκεκριμένη δομή ώστε το περιεχόμενο να φορτωθεί σαν
        //τιμή κελιού μιας μοναδικής στήλης (με οποιοδήποτε όνομα) για το
        //αντίστοιχο ID-γραμμή
```

Τέλος, χρησιμοποιείται η κατάλληλη συνάρτηση εξόδου, η οποία είναι διαθέσιμη από την HBase και είναι συμβατή με το Hadoop MapReduce, ώστε να αποθηκευτούν τα δεδομένα σε ειδικά αρχεία, τα οποία ονομάζονται HFiles και κατάλληλα για την μετέπειτα μαζική φόρτωσή τους (bulk loading) στην HBase. Προτιμάται η τεχνική του bulk loading διότι η σειριακή προσθήκη του κάθε key/value pair στη βάση χρειάζεται σημαντικά περισσότερο

χρόνο εξαιτίας της ενημέρωσης του index και άλλων στοιχείων της βάσης κατά την εισαγωγή κάθε στοιχείου.

Πρέπει να τονίσουμε ότι στη συγκεκριμένη διαδικασία χρησιμοποιούνται τρεις διαφορετικές ροές εξόδου. Μία για τα key/value pairs που αφορούν το ανεστραμμένο ευρετήριο, μία δεύτερη για τα key/value pairs που αφορούν το forward index και μία τρίτη για τα key/value pairs που αφορούν το περιεχόμενο των κειμένων. Τα key/value pairs εξόδου που αφορούν τη δημιουργία του ανεστραμμένου ευρετηρίου έχουν κατάλληλη δομή ώστε η λέξη (key) να αποτελέσει το κλειδί της γραμμής του πίνακα, το ID του κειμένου (value) να αποτελέσει το όνομα της στήλης, υποδηλώνοντας ότι η λέξη περιέχεται στο συγκεκριμένο κείμενο, και η τιμή του αντίστοιχου κελιού να είναι κενή για τη μείωση του απαιτούμενου αποθηκευτικού χώρου. Η συγκεκριμένη δομή της εξόδου διαφέρει σε σχέση με την διαδικασία δημιουργίας του ευρετηρίου που περιγράφτηκε σε προηγούμενη ενότητα, όπου η λέξη(key) αποτελούσε το κλειδί της γραμμής του πίνακα, αλλά η λίσταID(value) αποτελούσε την τιμή κελιού της μοναδικής στήλης που δημιουργούταν για κάθε γραμμή. Με τον τρόπο αυτό, για κάθε λέξη του ανεστραμμένου ευρετηρίου δημιουργούμε μία στήλη με όνομα αυτό του ID του αντίστοιχου κειμένου και επομένως ικανοποιούμε τις απαιτήσεις για την αποδοτική ενημέρωση του ευρετηρίου.

Πολυπλοκότητα

Η πολυπλοκότητα της τροποποιημένης διαδικασίας δεν διαφέρει σημαντικά σε σχέση με αυτή της διαδικασίας που περιγράφηκε σε προηγούμενη ενότητα. Η μόνη διαφορά έγκειται στη δημιουργία του forward index των κειμένων της συλλογής, η οποία αυξάνει την πολυπλοκότητα της διαδικασίας δημιουργίας ως εξής:

- Η πολυπλοκότητα από την ταξινόμηση των λέξεων του forward index του κάθε κειμένου, στους mappers, είναι O(n*logn), όπου n ο αριθμός των λέξεων του κειμένου που θα συμπεριληφθούν στο ευρετήριο, ο οποίος είναι σε γενικές γραμμές σχετικά μικρός για κάθε κείμενο με αποτέλεσμα να μην αυξάνει σημαντικά τη συνολική πολυπλοκότητα.
- Η πολυπλοκότητα της επεξεργασίας των ενδιάμεσων key/value pairs που αφορούν το forward index στους combiners είναι γραμμική ως προς τον αριθμό των κειμένων που αντιστοιχούν στο συγκεκριμένο mapper task, αφού σε κάθε key αντιστοιχεί ένα μοναδικό value.

- Για τον ίδιο λόγο, η πολυπλοκότητα της επεξεργασίας των key/value pairs που αφορούν το forward index στους reducers είναι γραμμική ως προς τον αριθμό των κειμένων που αντιστοιχούν στο συγκεκριμένο reduce task.

Η υπολογιστική πολυπλοκότητα της εφαρμογής εξακολουθεί να είναι χαμηλή, παρά την ταξινόμηση των λέξεων του κάθε κειμένου για τη δημιουργία του forward index. Το κόστος ανάκτησης και μεταφοράς των δεδομένων παραμένει υψηλό, με αποτέλεσμα να επηρεάζει σημαντικά το χρόνο δημιουργίας του ευρετηρίου και να καθιστά αμελητέα αυτή τη μικρή αύξηση της πολυπλοκότητας. Έτσι, όπως και στην προηγούμενη διαδικασία, αναμένεται γραμμική εξάρτηση του χρόνου εκτέλεσης της διαδικασίας σε σχέση με το μέγεθος της εισόδου.

Αύξηση των ενδιάμεσων και τελικών key/value pairs για τη δημιουργία του forward index

Η δημιουργία του forward index απαιτεί την εξαγωγή επιπλέον key/value pairs, τόσο από τους mappers, όσο και από του reducers. Τα key/value pairs αυτά πρέπει να ταξινομηθούν και να μεταφερθούν στους reducers, γεγονός που θα μπορούσε να επιβραδύνει την εκτέλεση της εφαρμογής, αν ο αριθμός ή ο όγκος τους ήταν μεγάλος. Ωστόσο, ο αριθμός των key/value pairs που αφορούν το forward index είναι ίσος με τον αριθμό των κειμένων της συλλογής, ο οποίος είναι αρκετά μικρότερος σε σχέση με τον αριθμό των λέξεων των κειμένων, οι οποίες αποτελούν τα key/value pairs για τη δημιουργία του ανεστραμμένου ευρετηρίου. Ακόμα, ο όγκος των δεδομένων που πρέπει να μεταφερθούν στους reducers δεν αυξάνεται σημαντικά, αφού οι λέξεις που συμπεριλαμβάνονται στο forward index είναι πολύ λιγότερες από τις λέξεις που περιέχουν τα κείμενα της συλλογής με αποτέλεσμα ο όγκος των αντίστοιχων key/value pairs να είναι σχετικά μικρός. Τέλος, τα key/value pairs που αφορούν το forward index δεν απαιτούν καμία επιπλέον επεξεργασία από τους reducers, πριν από αποθήκευσή τους στα κατάλληλα αρχεία εξόδου. Για τους λόγους αυτούς, παρά την αύξηση των ενδιάμεσων και τελικών key/value pairs, ο χρόνος εκτέλεσης της εφαρμογής δεν αυξάνεται σημαντικά και έτσι η χρήση του forward index, για την αποδοτικότερη ενημέρωση του ευρετηρίου, αποτελεί μια αποδεκτή επιλογή.

4.3.7 Ομοιόμορφη κατανομή των key/value pairs που αφορούν τη δημιουργία του forward index - Τροποποίηση της συνάρτησης partitioning

Στην ανάλυση που προηγήθηκε θεωρήσαμε δεδομένη την δυνατότητα της συνάρτησης partitioning να κατανέμει ομοιόμορφα στους reducers όλα τα key/value pairs, και επομένως και αυτά που αφορούν τη δημιουργία του forward index. Ωστόσο, η συνάρτηση που χρησιμοποιήσαμε σε προηγούμενη ενότητα είναι σχεδιασμένη για να κατανέμει ομοιόμορφα μόνο τα key/value pairs που αφορούν το ανεστραμμένο ευρετήριο και το περιεχόμενο των κειμένων. Το γεγονός ότι τα key/value pairs που αφορούν τη δημιουργία του forward index έχουν ίδιου τύπου key με αυτά που αφορούν το περιεχόμενο των κειμένων διευκολύνει σημαντικά τη διαδικασία, αφού δεν απαιτείται νέος διαχωρισμός διαστημάτων για την ομοιόμορφη κατανομή τους στους reducers. Χρησιμοποιώντας τη συνάρτηση δημιουργίας εικονικών IDs, σε συνδυασμό με τα διαστήματα που έχουν δημιουργηθεί για τα key/value pairs του περιεχομένου των κειμένων, έχουμε τη δυνατότητα να κατανέμουμε ομοιόμορφα τα νέα key/value pairs, κατανέμοντας, έτσι, ομοιόμορφα το φορτίο επεξεργασίας στους reducers.

4.4 Διαδικασία ενημέρωσης του ανεστραμμένου ευρετηρίου

4.4.1 Γενική ιδέα για την ενημέρωση του ευρετηρίου

Έχοντας δημιουργήσει το ανεστραμμένο ευρετήριο με βάση την τροποποιημένη διαδικασία δημιουργίας που περιγράφθηκε στην προηγούμενη ενότητα, έχουμε στη διάθεσή μας:

- Το υπάρχον ανεστραμμένο ευρετήριο, αποθηκευμένο με τη δομή που επιτρέπει την αποδοτικότερη ενημέρωση του, σε έναν πίνακα της HBase.
- Το forward index της συλλογής, σε έναν δεύτερο πίνακα με κλειδί το ID των κειμένων.

Προκειμένου να καταλήξουμε σε μία ενημερωμένη έκδοση του ευρετηρίου, η οποία να είναι σύμφωνη με όλες τις τροποποιήσεις που έχουν γίνει στα κείμενα της υπάρχουσας συλλογής, είναι απαραίτητο, όπως παρουσιάσαμε και κατά την περιγραφή του προβλήματος, να διαγραφούν οι εγγραφές του ευρετηρίου που αφορούν κείμενα τα οποία έχουν τροποποιηθεί και να προστεθούν οι κατάλληλες εγγραφές για τις λέξεις που περιέχονται στις τροποποιημένες εκδόσεις των κειμένων αυτών. Όσον αφορά τα νέα κείμενα που προστίθενται στη συλλογή, η διαδικασία αυτή απλοποιείται, αφού η φάση της

διαγραφής δεν έχει κανένα απολύτως νόημα και είναι απαραίτητο απλά να προσθέσουμε τις κατάλληλες εγγραφές για τις λέξεις που περιέχονται στα νέα αυτά κείμενα.

Αυτό σημαίνει ότι κάθε mapper πρέπει να σαρώσει το νέο κείμενο, προκειμένου να εξάγει ένα key/value pair για κάθε λέξη που επιθυμούμε να συμπεριλάβουμε στο ευρετήριο, αλλά και να σαρώσει το forward index της παλιάς έκδοσης του κειμένου (προκειμένου να μην επεξεργαστεί ξανά την παλαιά έκδοση του κειμένου) και να εξάγει ένα key/value pair για κάθε λέξη που περιέχεται σε αυτό και άρα πρέπει να διαγραφεί από το ευρετήριο. Στη συνέχεια, οι reducers πρέπει να επεξεργαστούν και τα δύο είδη key/value pairs, ώστε να διαγράψουν τις παλιές εγγραφές από τη βάση δεδομένων και να προσθέσουν τις καινούργιες. Η διαδικασία αυτή επιβαρύνει σημαντικά την εκτέλεση της ενημέρωσης του ευρετηρίου, αφού:

- αυξάνεται σε μεγάλο βαθμό ο αριθμός των ενδιάμεσων key/value pairs που εξάγουν οι mappers, τα οποία πρέπει να ταξινομηθούν και να μεταφερθούν στους reducers
- ο αριθμός των προσβάσεων στη βάση δεδομένων, ειδικά για την διαγραφή των παλαιών εγγραφών, η οποία δεν μπορεί να γίνει με μαζικό τρόπο, όπως η φόρτωση δεδομένων, γίνεται εξαιρετικά μεγάλος

4.4.2 Σύγκριση των διαφορετικών εκδόσεων των κειμένων για την επιτάχυνση της διαδικασίας ενημέρωσης

Το γεγονός ότι ένα κείμενο έχει υποστεί κάποια τροποποίηση, σε σχέση με την μορφή που έχει συμπεριληφθεί στο ευρετήριο, δεν σημαίνει απαραίτητα ότι το κείμενο αυτό έχει αλλάξει σε μεγάλο βαθμό, ειδικά στην περίπτωση που οι ενημερώσεις του ευρετηρίου είναι συχνές. Για το λόγο αυτό αναπτύξαμε μια διαφορετική στρατηγική αντιμετώπισης των τροποποιημένων κειμένων, με σκοπό την ελαχιστοποίηση των αλλαγών που πρέπει να πραγματοποιηθούν στο ευρετήριο και επομένως την αντιμετώπιση των παραπάνω προβλημάτων.

Προκειμένου να επιταχύνουμε τη διαδικασία σύγκρισης επιλέγουμε οι λέξεις στο forward index κάθε κειμένου να είναι ταξινομημένες, κάτι το οποίο δεν ήταν αναγκαίο κατά τη διαγραφή όλων των παλαιών εγγραφών που περιγράφθηκε προηγουμένως.

Συγκεκριμένα κατά τη φάση του map:

- Σαρώνουμε την νέα έκδοση του κειμένου και εισάγουμε τις λέξεις που αναγνωρίζονται σε ένα δέντρο ταξινόμησης.
- Φορτώνουμε από τη βάση δεδομένων το forward index της παλαιάς έκδοσης του κειμένου.
- Διατρέχουμε ταυτόχρονα τα στοιχεία του δέντρου ταξινόμησης (σε αύξουσα σειρά) και τα στοιχεία του forward index της παλαιάς έκδοσης, τα οποία είναι επίσης ταξινομημένα, προκειμένου να καταγράψουμε τις λέξεις που δεν υπήρχαν στην παλιά έκδοση του κειμένου και προστέθηκαν στη νέα, καθώς και τις λέξεις που υπήρχαν στην παλιά έκδοση του κειμένου και διαγράφτηκαν στη νέα.

Κατά τη φάση του reduce:

- Προσθέτουμε στο ευρετήριο τις εγγραφές που ανήκουν στην πρώτη κατηγορία,
- Διαγράφουμε από το ευρετήριο τις εγγραφές που ανήκουν στη δεύτερη κατηγορία

Ορθότητα

Η ορθότητα της παραπάνω διαδικασίας δεν είναι δύσκολο να αποδειχθεί. Έστω ότι η παλιά έκδοση ενός κειμένου περιέχει ένα σύνολο λέξεων Α, ενώ η νέα περιέχει ένα σύνολο λέξεων Β. Το ευρετήριο περιέχει τις εγγραφές με βάση μόνο το σύνολο Α και επιθυμούμε τελικά να περιέχει τις εγγραφές μόνο του συνόλου Β. Η πρώτη διαδικασία που περιγράφθηκε, η οποία κρίνεται μη αποδοτική, απλά διέγραφε όλες τις εγγραφές που αφορούσαν το σύνολο Α και προσέθετε αυτές του συνόλου Β. Η τροποποιημένη διαδικασία που περιγράψαμε στηρίζεται στο γεγονός ότι η τομή $A \cap B$ των δύο συνόλων δεν αποτελεί το κενό σύνολο και μάλιστα ότι αποτελεί ένα σημαντικό τμήμα των συνόλων αυτών, προκειμένου να επιταχύνει τη διαδικασία της ενημέρωσης. Έτσι από τις εγγραφές του ευρετηρίου που αφορούν το σύνολο Α επιλέξαμε να διαγράψουμε τις εγγραφές που ανήκουν στο $A - A \cap B$ αφήνοντας στο ευρετήριο μόνο τις εγγραφές που αφορούν το σύνολο $A \cap B$. Στη συνέχεια, προστίθενται στο ευρετήριο οι εγγραφές που αφορούν το σύνολο $B - A \cap B$, με αποτέλεσμα το ευρετήριο να περιέχει τις λέξεις που αφορούν το σύνολο $A \cap B + B - A \cap B = B$. Ακόμα και αν αντιστρέψουμε τη σειρά εκτέλεσης των διαδικασιών προσθήκης και διαγραφής εγγραφών, η τελική μορφή του ευρετηρίου δεν αλλάζει. Το ευρετήριο αρχικά περιέχει τις εγγραφές που αφορούν το σύνολο Α, μετά την προσθήκη των εγγραφών του συνόλου $B - A \cap B$ περιέχει τις εγγραφές που αφορούν το

$A + B - A \cap B$, ενώ μετά τη διαγραφή των εγγραφών που ανήκουν στο $A - A \cap B$, το ευρετήριο καταλήγει και πάλι να περιέχει τις εγγραφές που αφορούν το σύνολο B.

Μπορούμε, λοιπόν, με ασφάλεια να χρησιμοποιήσουμε την τροποποιημένη αυτή διαδικασία που στηρίζεται στη σύγκριση των διαφορετικών εκδόσεων των κειμένων για την επιτάχυνση της διαδικασίας ενημέρωσης του ευρετηρίου.

Αλγόριθμος σύγκρισης των δύο εκδόσεων του κειμένου

Η χρήση του παρακάτω αλγορίθμου προϋποθέτει ότι οι λέξεις του forward index της παλιάς έκδοσης του κάθε κειμένου είναι ταξινομημένες και ότι κατά τη σάρωση του κειμένου έχουμε δημιουργήσει ένα δέντρο ταξινόμησης το οποίο περιέχει τις λέξεις της νέας έκδοσης του κειμένου. Συγκεκριμένα ο αλγόριθμος είναι ο εξής:

Αλγόριθμος 9: Αλγόριθμος σύγκρισης των δύο εκδόσεων των κειμένων

```
λέξη_FI=Ανάγνωση της πρώτης λέξης από το forward index;
λέξη_δέντρου=Ανάγνωση της πρώτης λέξης από το δέντρο ταξινόμησης;
Όσο (λέξη_FI!=null AND λέξη_δέντρου!=null)://μέχρι οι λέξεις ενός από τα δύο να τελειώσουν
        Αν (λέξη_δέντρου > λέξη_FI)
                Εξαγωγή key/value pair για διαγραφή(λέξη_FI, ID κειμένου);
                λέξη_FI =Ανάγνωση της επόμενης λέξης από το forward index;
        Αλλιώς Αν (λέξη_δέντρου < λέξη_FI)
                Εξαγωγή key/value pair για προσθήκη(λέξη_δέντρου, ID κειμένου);
                λέξη_δέντρου =Ανάγνωση της επόμενης λέξης από το δέντρο;
        Αλλιώς
                λέξη_FI =Ανάγνωση της επόμενης λέξης από το forward index;
                λέξη_δέντρου =Ανάγνωση της επόμενης λέξης από το δέντρο;

Όσο (λέξη_δέντρου!=null) //αν το δέντρο περιέχει ακόμα λέξεις
        Εξαγωγή key/value pair για προσθήκη(λέξη_δέντρου, ID κειμένου);
        λέξη_δέντρου =Ανάγνωση της επόμενης λέξης από το δέντρο;
Όσο (λέξη_FI!=null) //αν το forward index περιέχει ακόμα λέξεις
        Εξαγωγή key/value pair για διαγραφή(λέξη_FI, ID κειμένου);
        λέξη_FI =Ανάγνωση της επόμενης λέξης από το forward index;
```

Όφελος από την χρήση της διαδικασίας σύγκρισης των διαφορετικών εκδόσεων

Με τη βοήθεια του παραπάνω αλγορίθμου έχουμε καθορίσει τις λέξεις που δεν υπήρχαν στην παλιά έκδοση του κειμένου και προστέθηκαν στη νέα, καθώς και τις λέξεις που υπήρχαν στην παλιά έκδοση του κειμένου και διαγράφτηκαν, ενώ ταυτόχρονα έχουμε εξάγει τα key/value pairs και για τις δύο κατηγορίες λέξεων, με αποτέλεσμα να μπορούμε να εκτελέσουμε μόνο τις απολύτως αναγκαίες αλλαγές στο ευρετήριο.

Η πολυπλοκότητα του παραπάνω αλγορίθμου είναι γραμμική ως προς το μέγεθος των δύο εκδόσεων των κειμένων, ενώ η πολυπλοκότητα από την ταξινόμηση των λέξεων του forward index κάθε κειμένου είναι O(n*logn) όπου n ο αριθμός των λέξεων που περιέχονται στο κείμενο και θα συμπεριληφθούν στο ευρετήριο, με αποτέλεσμα να μην αυξάνεται σημαντικά ο χρόνος εκτέλεσης της ενημέρωσης. Για το λόγο αυτό η χρήση της παραπάνω διαδικασίας αποτελεί μία καλή επιλογή προκειμένου:

- Να μειωθεί ο αριθμός των ενδιάμεσων key/value pairs που εξάγουν οι mappers, τα οποία πρέπει να ταξινομηθούν και να μεταφερθούν στους reducers .
- Να μειωθεί ο αριθμός των προσβάσεων στη βάση δεδομένων για την διαγραφή των παλαιών εγγραφών, η οποία, μάλιστα δεν μπορεί να γίνει με μαζικό τρόπο, όπως η φόρτωση δεδομένων.
- Να μειωθεί ο αριθμός των εγγραφών που πρέπει να προστεθούν στο ευρετήριο, των οποίων η εισαγωγή, αν και πραγματοποιείται μαζικά, απαιτεί χρόνο για την ολοκλήρωση της.

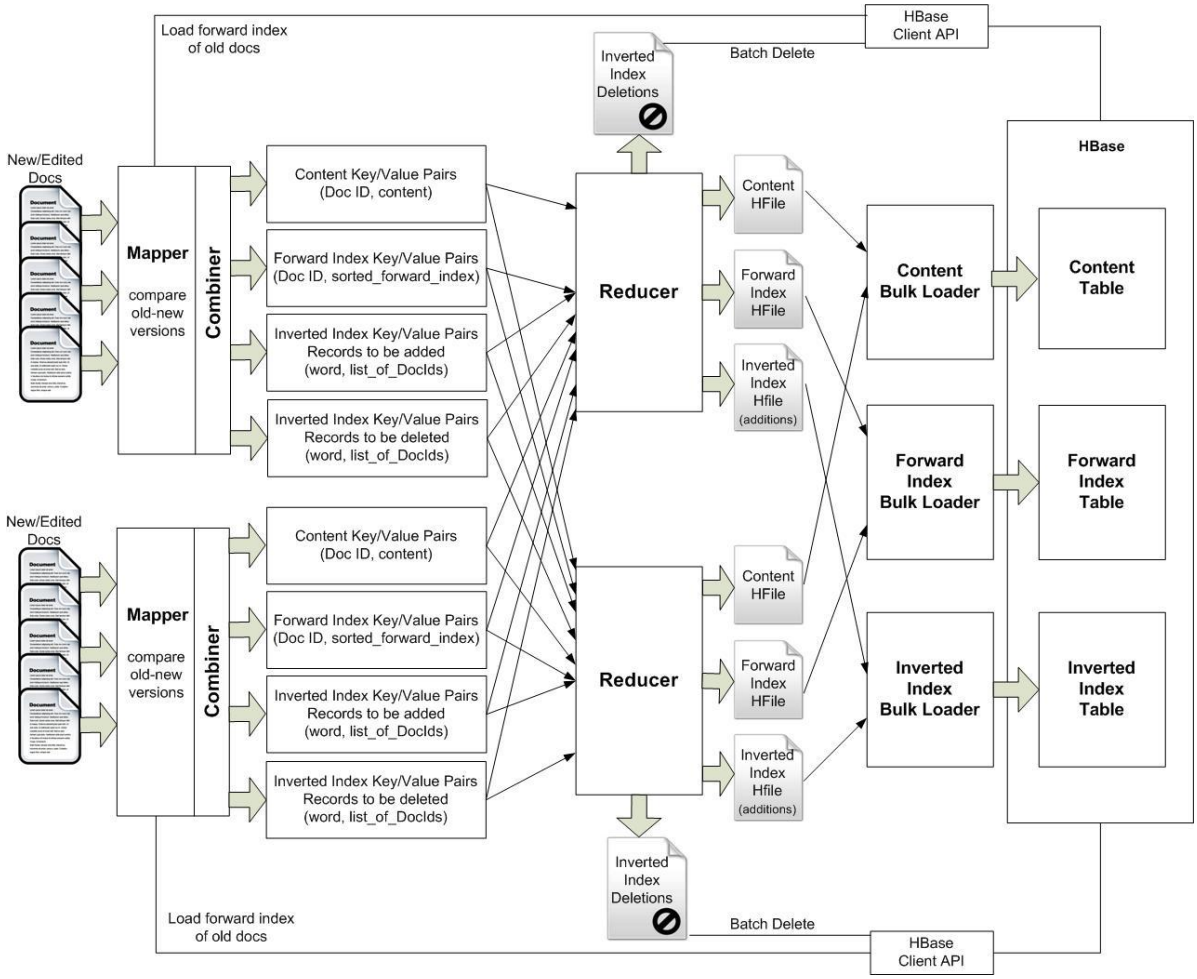
Οι διαδικασίες αυτές απαιτούν προσβάσεις στους δίσκους της συστοιχίας, των οποίων το κόστος είναι σημαντικά μεγαλύτερο σε σχέση με το κόστος της επιπλέον επεξεργασίας που εισάγεται από τη σύγκριση των εκδόσεων των κειμένων. Έτσι, με τη σύγκριση των διαφορετικών εκδόσεων των κειμένων έχουμε τη δυνατότητα να επιταχύνουμε την ενημέρωση του ευρετηρίου.

4.4.3 Διαδικασία ενημέρωσης του ευρετηρίου μετά από την τροποποίηση ενός ποσοστού των κειμένων της υπάρχουσας συλλογής και την προσθήκη νέων κειμένων

Η διαδικασία ενημέρωσης του ευρετηρίου που αναπτύξαμε βασίζεται στη γενική ιδέα ενημέρωσης που παρουσιάστηκε στην αρχή της ενότητας και, πιο συγκεκριμένα, υλοποιεί

την ιδέα για τη σύγκριση των διαφορετικών εκδόσεων των κειμένων προκειμένου να επιταχυνθεί η ενημέρωση του ευρετηρίου. Ενώ κατά τη δημιουργία του ευρετηρίου είχαμε μόνο προσθήκες εγγραφών στο ευρετήριο, κατά την ενημέρωση του είναι απαραίτητο να έχουμε και διαγραφές, με αποτέλεσμα η διαδικασία που θα χρησιμοποιήσουμε να διαφέρει σε κάποιο βαθμό από αυτή της δημιουργίας του ευρετηρίου. Η εισαγωγή νέων κειμένων στη συλλογή απαιτεί την τροποποίηση της διαδικασίας αυτής, αφού τα κείμενα μπορούν να αντιμετωπιστούν σαν κείμενα των οποίων η παλαιά έκδοση δεν περιείχε καμία απολύτως λέξη.

Στο παρακάτω σχήμα παρουσιάζεται γραφικά η διαδικασία ενημέρωσης ενός υπάρχοντος ανεστραμμένου ευρετηρίου, η οποία περιγράφεται αναλυτικά στη συνέχεια. Ο αριθμός των mappers και των reducers έχει επιλεγεί για την καλύτερη απεικόνιση της διαδικασίας και δεν αντιστοιχεί στην πραγματική εκτέλεση της εργασίας.



Σχήμα 12: Διαδικασία ενημέρωσης ενός υπάρχοντος ανεστραμμένου ευρετηρίου

Σαν είσοδο της MapReduce εργασίας για την ενημέρωση του ευρετηρίου θεωρούμε ένα σύνολο αρχείων, τα οποία περιέχουν τα κείμενα της συλλογής που έχουν υποστεί κάποια τροποποίηση, καθώς και τα νέα κείμενα που εισάγονται στη συλλογή.

Η διαδικασία ενημέρωσης του ευρετηρίου είναι η εξής:

Πριν την εκκίνηση της επεξεργασίας, το MapReduce framework αναλαμβάνει να δημιουργήσει ένα map task για κάθε block δεδομένων από αυτά που αποτελούν τη συλλογή των τροποποιημένων ή νέων κειμένων, το οποίο αναλαμβάνει την επεξεργασία των κειμένων που ανήκουν στο block αυτό. Τα map tasks μοιράζονται στους διαφορετικούς κόμβους της συστοιχίας.

Κάθε map task εκτελεί τον εξής αλγόριθμο:

Αλγόριθμος 10: Αλγόριθμος των mappers για την ενημέρωση ενός υπάρχοντος ανεστραμμένου ευρετηρίου

Σάρωση του block για την αναγνώριση των ανεξάρτητων κειμένων που ανήκουν σε αυτό και δημιουργία key/value pairs (ID κειμένου, περιεχόμενο);
Για κάθε key/value pair:
    Εξαγωγή ενός key/value pair(ID κειμένου, περιεχόμενο κειμένου);
    Δημιουργία ενός άδειου δέντρου ταξινόμησης;
    Σάρωση του περιεχομένου για την αναγνώριση των λέξεων που περιέχονται στο κείμενο και επιθυμούμε να συμπεριλάβουμε στο ευρετήριο;
    Για κάθε λέξη που αναγνωρίζεται:
        Προσθήκη της λέξης στο δέντρο ταξινόμησης;
    Φόρτωση από την HBase του forward index της παλαιάς έκδοσης του κειμένου;
    Χρήση του αλγορίθμου για τη σύγκριση των διαφορετικών εκδόσεων του κειμένου, χρησιμοποιώντας το forward index και το δέντρο ταξινόμησης, ο οποίος εξάγει και τα κατάλληλα key/value pairs, τόσο για τις προσθήκες όσο και για τις διαγραφές, και ταυτόχρονη προσθήκη της κάθε λέξης-στοιχείου του δέντρου σε έναν buffer με όνομα sorted_forward_index;
    Εξαγωγή ενός key/value pair(ID κειμένου, sorted_forward_index);

Για κάθε key/value pair που εξάγεται, χρησιμοποιείται η συνάρτηση partitioning προκειμένου να προσδιοριστεί ο reducer ο οποίος είναι αρμόδιος για την επεξεργασία του

συγκεκριμένου key. Μετά την ολοκλήρωση του κάθε map task εκτελείται ένας combiner, για κάθε map task, με σκοπό την μείωση των δεδομένων που πρέπει να μεταφερθούν από τους mappers προς τους reducers. Η διαφορά σε σχέση με την αρχική δημιουργία του ευρετηρίου είναι ότι κατά την ενημέρωση έχουμε key/value pairs τα οποία αφορούν τις διαγραφές που πρέπει να πραγματοποιηθούν στο ευρετήριο. Αυτή η διαφορά ωστόσο δεν επηρεάζει τη λειτουργία του combiner, ο οποίος χρησιμοποιεί τον αλγόριθμο που παρουσιάστηκε και προηγουμένως.

Μετά την ολοκλήρωση όλων των map tasks τα key/value pairs ομαδοποιούνται και ταξινομούνται ως προς το κλειδί τους και στη συνέχεια διαβάζονται από τους reducers που έχει ορίσει η συνάρτηση partitioning για κάθε κλειδί. Οι reducers εκτελούν τον εξής αλγόριθμο:

Αλγόριθμος 11: Αλγόριθμος των reducers για την ενημέρωση ενός υπάρχοντος ανεστραμμένου ευρετηρίου

```
Για κάθε ξεχωριστό κλειδί που διαβάζεται:
    Αν το κλειδί αφορά την προσθήκη εγγραφής στο ευρετήριο:
        Για κάθε key/value pair με τη συγκεκριμένη λέξη-κλειδί:
            Για κάθε στοιχείο-ID της λίστας:
                Εξαγωγή ενός key/value pair (λέξη, ID κειμένου) στην έξοδο
                που αφορά προσθήκες στο inverted index;
                //με συγκεκριμένη δομή ώστε το ID να φορτωθεί σαν το
                //όνομα της στήλης και όχι σαν τιμή κελιού
    Αν το κλειδί αφορά την διαγραφή εγγραφής από το ευρετήριο:
        Δημιουργία ενός αντικειμένου DELETE για τη συγκεκριμένη λέξη-κλειδί;
        Για κάθε key/value pair με τη συγκεκριμένη λέξη-κλειδί:
            Για κάθε στοιχείο-ID της λίστας:
                Προσθήκη του ID στο αντικείμενο DELETE για την
                διαγραφή της αντίστοιχης στήλης ;
        Εξαγωγή ενός key/value pair (λέξη, DELETE) στην έξοδο που αφορά τις
        διαγραφές στο inverted index;
    Αν το κλειδί αφορά το forward index:
        Εξαγωγή ενός key/value pair (ID κειμένου, sorted_forward_index) στην
        έξοδο που αφορά το forward index;
```

//με συγκεκριμένη δομή ώστε το sorted_forward_index να φορτωθεί σαν
//τιμή κελιού μιας μοναδικής στήλης (με οποιοδήποτε όνομα) για την
//αντίστοιχη λέξη-γραμμή

Αλλιώς:

Εξαγωγή ενός key/value pair (ID κειμένου, περιεχόμενο) στην έξοδο που αφορά το περιεχόμενο των κειμένων;
//με συγκεκριμένη δομή ώστε το περιεχόμενο να φορτωθεί σαν
//τιμή κελιού μιας μοναδικής στήλης (με οποιοδήποτε όνομα) για το
//αντίστοιχο ID-γραμμή

Τέλος, εκτός από τη συνάρτηση εξόδου που χρησιμοποιήθηκε στις προηγούμενες περιπτώσεις, προκειμένου να αποθηκευτούν τα δεδομένα σε αρχεία κατάλληλα για την μετέπειτα μαζική φόρτωσή τους στη βάση δεδομένων, χρησιμοποιείται και η κατάλληλη συνάρτηση εξόδου, ώστε να γίνει συλλογική (batch) διαγραφή των στοιχείων που περιγράφονται από τα key/value pairs τύπου Delete, μέσω του HBase Client API.

Πολυπλοκότητα

Η πολυπλοκότητα της διαδικασίας ενημέρωσης δεν διαφέρει σημαντικά σε σχέση με αυτή της αρχικής δημιουργίας του ευρετηρίου. Η βασική διαφορά έγκειται στη σύγκριση των δύο εκδόσεων κάθε κειμένου, με σκοπό των εντοπισμό των διαφορών τους, η οποία κατ' επέκταση μεταβάλει τον αριθμό των ενδιάμεσων και τελικών key/value pairs. Πρέπει να σημειώσουμε ότι ο αριθμός των διαφορών μεταξύ των δύο εκδόσεων ενός κειμένου είναι το πολύ ίσος με τον αριθμό των λέξεων των δύο εκδόσεων (στην περίπτωση που όλες οι λέξεις είναι διαφορετικές μεταξύ τους) και κατά συνέπεια εξαρτάται γραμμικά από το μέγεθος των δύο εκδόσεων του κειμένου.

Αναλύοντας την πολυπλοκότητα κάθε επιμέρους τμήματος της διαδικασίας, εξαιρώντας την πολυπλοκότητα επεξεργασίας από το ίδιο το framework, καταλήγουμε στα εξής:

- Η πολυπλοκότητα κατά την ανάγνωση των κειμένων είναι γραμμική ως προς το μέγεθος του αρχείου-block των κειμένων που αναλαμβάνει το κάθε mapper task.
- Η πολυπλοκότητα κατά τη σάρωση του κάθε κειμένου, για την αναγνώριση των λέξεων που αυτό περιέχει, είναι γραμμική ως προς το μέγεθος του κειμένου και επομένως η σάρωση των κειμένων που αντιστοιχούν σε ένα mapper task είναι γραμμική ως προς το μέγεθος του αρχείου κειμένων που του έχει ανατεθεί.

- Η πολυπλοκότητα από την ταξινόμηση των λέξεων του forward index του κάθε κειμένου είναι O(n*logn), όπου n ο αριθμός των λέξεων του κειμένου που θα συμπεριληφθούν στο ευρετήριο, ο οποίος είναι σε γενικές γραμμές σχετικά μικρός για κάθε κείμενο με αποτέλεσμα να μην αυξάνει σημαντικά τη συνολική πολυπλοκότητα.
- Η πολυπλοκότητα του αλγορίθμου σύγκρισης των δύο εκδόσεων κάθε κειμένου είναι γραμμική ως προς το μέγεθος των δύο εκδόσεων του κάθε κειμένου.
- Η πολυπλοκότητα της κατασκευής των επιμέρους λιστών από τους combiners είναι γραμμική ως προς τον αριθμό των διαφορών των κειμένων που αντιστοιχούν στο συγκεκριμένο mapper task και επομένως γραμμική ως προς το μέγεθος των παλαιών και νέων εκδόσεων των κειμένων. Για τα key/value pairs που αφορούν το περιεχόμενο των κειμένων ή το forward index, το κάθε key έχει μοναδικό value και επομένως η επεξεργασία τους είναι γραμμική ως προς τον αριθμό των κειμένων που αντιστοιχούν στο συγκεκριμένο mapper task.
- Η πολυπλοκότητα της επεξεργασίας σε κάθε reducer είναι γραμμική ως προς τον αριθμό των key/value pairs που του έχουν ανατεθεί, τα οποία αντιπροσωπεύουν τις διαφορές μεταξύ των εκδόσεων των κειμένων και συνεπώς ο αριθμός τους εξαρτάται γραμμικά από το μέγεθος των δύο εκδόσεων των κειμένων. Για τα key/value pairs που αφορούν το περιεχόμενο των κειμένων ή το forward index, το κάθε key έχει μοναδικό value και επομένως η επεξεργασία τους είναι γραμμική ως προς τον αριθμό των κειμένων της συλλογής που αντιστοιχούν στον κάθε reducer.

Όπως φαίνεται από την παραπάνω ανάλυση, η πολυπλοκότητα της εφαρμογής, με εξαίρεση την ταξινόμηση των λέξεων κάθε κειμένου για τη δημιουργία του forward index, η οποία ωστόσο δεν επηρεάζει σημαντικά το χρόνο εκτέλεσης, παραμένει γραμμική ως προς το μέγεθος της εισόδου. Ταυτόχρονα, η επεξεργασία και η μεταφορά των δεδομένων που εκτελείται από το MapReduce framework, με εξαίρεση την ταξινόμηση των ενδιάμεσων key/value pairs, η οποία, όπως αναφέραμε και προηγουμένως, δεν απαιτεί σημαντικό χρόνο διότι επεξεργάζεται μόνο τα keys των ενδιάμεσων key/value pairs, είναι επίσης γραμμική ως προς το μέγεθος της εισόδου. Έτσι, τελικά, αναμένεται γραμμική εξάρτηση του χρόνου εκτέλεσης της ενημέρωσης σε σχέση με το μέγεθος της συλλογής νέων ή τροποποιημένων κειμένων.

4.4.4 Τροποποίηση της διαδικασίας δειγματοληψίας και της συνάρτησης partitioning για την ομοιόμορφη κατανομή των key/value pairs

Στην ανάλυση που προηγήθηκε θεωρήσαμε δεδομένη την δυνατότητα της συνάρτησης partitioning να κατανέμει ομοιόμορφα στους reducers τα key/value pairs. Ωστόσο, τόσο η διαδικασία της δειγματοληψίας, όσο και η συνάρτηση partitioning που χρησιμοποιήσαμε σε προηγούμενη ενότητα, είναι σχεδιασμένες για να κατανέμουν ομοιόμορφα τα key/value pairs στην περίπτωση που έχουμε μόνο προσθήκη νέων κειμένων στη συλλογή. Είναι, λοιπόν, απαραίτητο να τροποποιήσουμε τη διαδικασία δειγματοληψίας, αλλά και τη συνάρτηση partitioning, προκειμένου να είναι συμβατές με τη διαδικασία ενημέρωσης του ευρετηρίου, η οποία επεξεργάζεται τα δεδομένα με διαφορετικό τρόπο και χρησιμοποιεί μια επιπλέον κατηγορία key/value pairs για την διαγραφή εγγραφών από το ευρετήριο.

Τα key/value pairs που αφορούν τη διαγραφή εγγραφών από τη βάση δεδομένων είναι ανεξάρτητα από αυτά που αφορούν την προσθήκη εγγραφών. Ταυτόχρονα, η διαδικασία με την οποία το σύστημα εκτελεί τις διαγραφές από τη βάση δεδομένων, σύμφωνα με τα key/value pairs εξόδου της εργασίας, διαφέρει σε σχέση με τη διαδικασία εγγραφής των key/value pairs σε ένα αρχείο, με σκοπό τη μετέπειτα μαζική φόρτωση του στη βάση, και ενδέχεται να απαιτεί περισσότερο χρόνο για την ολοκλήρωση της, λόγω των αλλαγών που απαιτούνται στη βάση μετά από τη διαγραφή κάθε στοιχείου της. Για το λόγο αυτό, προκειμένου να εξασφαλίσουμε ότι το φορτίο επεξεργασίας θα είναι περίπου ίδιο για όλους τους reducers, επιλέγουμε να θεωρήσουμε τη διανομή των key/value pairs σαν μια εντελώς ανεξάρτητη διαδικασία για κάθε μίας κατηγορία από key/value pairs. Οι κατηγορίες που χρησιμοποιούνται είναι οι εξής:

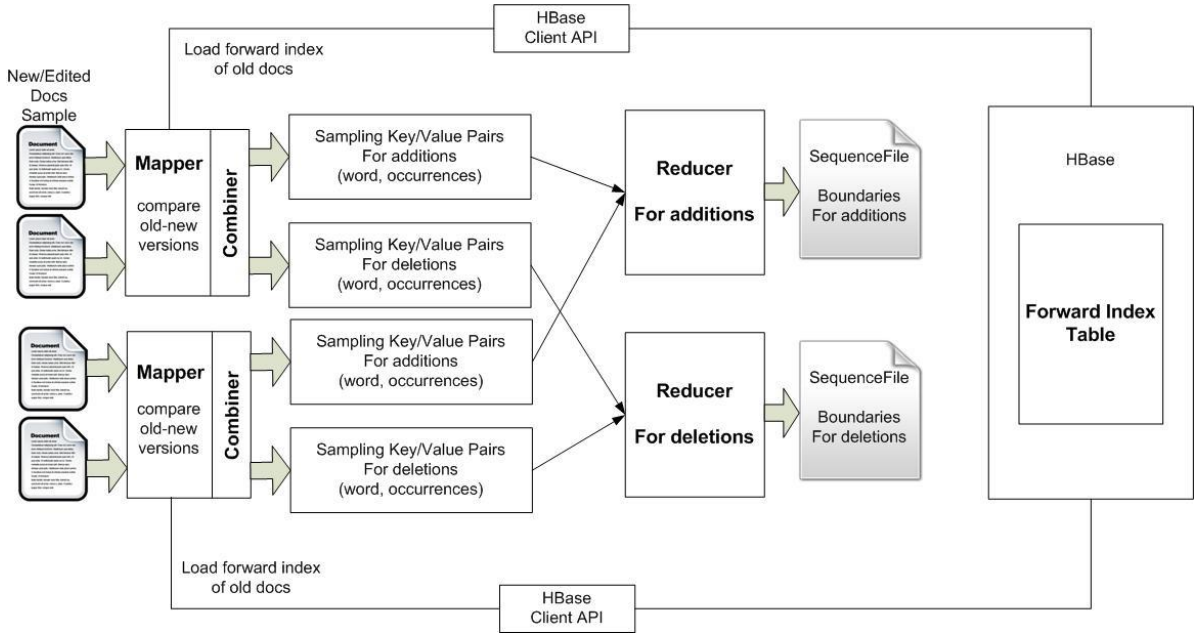
- Τα key/value pairs που αφορούν τη φόρτωση του περιεχομένου των κειμένων
- Τα key/value pairs που αφορούν τη δημιουργία του forward index
- Τα key/value pairs που αφορούν εισαγωγή εγγραφών στο ανεστραμμένο ευρετήριο
- Τα key/value pairs που αφορούν διαγραφή εγγραφών από το ανεστραμμένο ευρετήριοο

Οι ιδιότητες των τριών πρώτων κατηγοριών αναλύθηκαν στις προηγούμενες ενότητες. Τα key/value pairs που αφορούν τη διαγραφή εγγραφών από το ευρετήριο έχουν σαν key μια λέξη από αυτές που περιέχονται στα κείμενα της συλλογής. Το γεγονός αυτό δυσχεραίνει την ομοιόμορφη κατανομή τους στους reducers, αφού όπως αναλύσαμε, η Zipfian κατανομής της συχνότητας των λέξεων της αγγλικής γλώσσας δημιουργεί μεγάλη

διακύμανση στο πλήθος των pairs ανά key. Για το λόγο αυτό είναι απαραίτητη η χρήση δειγματοληψίας για την ομοιόμορφη κατανομή του φορτίου στους reducers, όπως χρειάστηκε και κατά την αρχική δημιουργία του ευρετηρίου.

Δεδομένου ότι η επεξεργασία των δεδομένων εισόδου κατά την ενημέρωση του ευρετηρίου διαφέρει σε σχέση με την επεξεργασία κατά την αρχική δημιουργία του ευρετηρίου, είναι αναγκαίο να τροποποιήσουμε την MapReduce εργασία δειγματοληψίας προκειμένου να είναι σύμφωνη με τη νέα μορφή επεξεργασίας, αλλά και τη νέα κατηγορία key/value pairs, η οποία αφορά την διαγραφή εγγραφών από τη βάση δεδομένων.

Στο παρακάτω σχήμα παρουσιάζεται γραφικά η τροποποιημένη διαδικασία δειγματοληψίας για την ενημέρωση του ευρετηρίου. Ο αριθμός των mappers του σχήματος είναι ενδεικτικός για την καλύτερη απεικόνιση της διαδικασίας. Αντίθετα, ο αριθμός των reducers αντιστοιχεί στον πραγματικό αριθμό των reducers της εργασίας.



Σχήμα 13: Διαδικασία δειγματοληψίας κατά την ενημέρωση του ευρετηρίου

Αναλυτικότερα, η τροποποιημένη διαδικασία δειγματοληψίας είναι η εξής:

- Κάθε mapper αναλαμβάνει ένα τμήμα από το σύνολο Α των κειμένων και σαρώνει ένα υποσύνολο Β των κειμένων του τμήματος αυτού. Επειδή κάθε mapper αναλαμβάνει ένα ανεξάρτητο και συγκεκριμένου μεγέθους υποσύνολο του αρχικού συνόλου κειμένων Α, τα δείγματα που επεξεργάζονται συνολικά οι mappers είναι

ομοιόμορφα κατανεμημένα μέσα στο σύνολο Α και έτσι το δείγμα μπορεί να θεωρηθεί αντιπροσωπευτικό.

- Ο mapper εκτελεί τον αλγόριθμο σύγκρισης των δύο εκδόσεων και για κάθε λέξη που πρέπει να προστεθεί στο ευρετήριο εξάγει ένα key/value pair της μορφής (λέξη, 1) το οποίο σημαίνει ότι είχαμε μία εμφάνιση της συγκεκριμένη λέξης, ενώ για κάθε λέξη η οποία πρέπει να διαγραφεί από το ευρετήριο εξάγει ένα key/value pair της μορφής (λέξη_delete, 1).
- Παράλληλα με την αναγνώριση των λέξεων του κειμένου, κάθε mapper καταγράφει το συνολικό αριθμό των λέξεων που πρέπει να προστεθούν, καθώς και το συνολικό αριθμό των λέξεων που πρέπει να διαγραφούν, ενώ όταν ολοκληρωθεί η σάρωση του κειμένου εξάγει ένα key/value pair της μορφής (πολύ_μικρό_κλειδί, αριθμός λέξεων) και ένα key/value pair της μορφής (πολύ_μικρό_κλειδί_delete, αριθμός λέξεων). Το πολύ_μικρό_κλειδί αναπαριστά μια συμβολοσειρά αλφαβητικά μικρότερη από κάθε πιθανή λέξη που μπορεί να αναγνωριστεί στα κείμενα της συλλογής και έχει σκοπό να ενημερώσει τον reducer που εκτελεί των διαχωρισμό των τμημάτων για το συνολικό αριθμό των λέξεων των κειμένων. Με τον τρόπο αυτό δεν είναι αναγκαία η διπλή σάρωση των key/value pairs που φτάνουν στον reducer, με αποτέλεσμα να επιταχύνεται η διαδικασία.
- Ο combiner αθροίζει τις τιμές των values για κάθε κλειδί, και εξάγει ένα key/value pair της μορφής (λέξη ή πολύ_μικρο_κλειδί, άθροισμα values) για κάθε κλειδί. Με τον τρόπο αυτό μειώνεται ο όγκος των δεδομένων που πρέπει να μεταφερθούν στον reducer, καθώς και ο αριθμός των αθροισμάτων που θα πρέπει αυτός να εκτελέσει.
- Η συνάρτηση partitioning διαχωρίζει τα key/value pairs σε αυτά των οποίων το key δεν περιέχει τη συμβολοσειρά '_delete', τα οποία προωθεί στον πρώτο από τους δύο reducers και σε αυτά των οποίων το key περιέχει τη συμβολοσειρά '_delete', τα οποία προωθεί στον δεύτερο reducer.
- Ο ένα από τους δύο reducers είναι υπεύθυνος για το διαχωρισμό των διαστημάτων που σχετίζονται με την προσθήκη εγγραφών, ενώ ο δεύτερος για το διαχωρισμό των διαστημάτων που σχετίζονται με την διαγραφή εγγραφών. Η λειτουργία του κάθε reducer, ωστόσο, είναι ακριβώς ίδια με αυτή που παρουσιάστηκε όταν είχαμε μόνο προσθήκες εγγραφών. Ο πρώτος reducer αρχικά λαμβάνει τα key/value pairs (πολύ_μικρό_κλειδί, αριθμός λέξεων) ώστε να υπολογίσει τον συνολικό αριθμό των λέξεων των κειμένων του δείγματος. Διαιρώντας τον αριθμό αυτό με τον αριθμό των

reducers που θα χρησιμοποιηθούν στην εφαρμογή κατασκευής του ευρετηρίου υπολογίζεται ο αριθμός των λέξεων που πρέπει να αναλάβει κάθε reducer. Καθώς διαβάζονται τα key/value pairs για τις λέξεις των κειμένων, ο reducer μετράει αθροιστικά τις εμφανίσεις των λέξεων μέχρι αυτές να ξεπεράσουν το όριο λέξεων ανά reducer. Όταν συμβεί αυτό ο reducer επιλέγει αν θα συμπεριλάβει την τελευταία λέξη στο διάστημα που δημιουργεί, εξάγει ένα key/value pair με την τιμή του ορίου του συγκεκριμένου διαστήματος και στη συνεχίζει την επεξεργασία των επόμενων λέξεων ώστε να ορίσει και τα επόμενα διαστήματα. Αντίστοιχα, ο δεύτερος reducer εκτελεί την ίδια διαδικασία αλλά για τα key/value pairs των οποίων το key περιέχει τη συμβολοσειρά '_delete'.

Ο ακριβής αλγόριθμος που εκτελείται είναι όμοιος με τον αλγόριθμο που παρουσιάσαμε στην προηγούμενη διαδικασία δειγματοληψίας και έτσι δεν είναι απαραίτητο να επαναληφθεί.

Με την εκτέλεση της παραπάνω MapReduce εργασίας δειγματοληψίας, έχουμε καθορίσει τα όρια των διαστημάτων στα οποία πρέπει να διαιρεθεί το εύρος τιμών των λέξεων προκειμένου να έχουμε ομοιόμορφη κατανομή των key/value pairs στους reducers, τόσο για τα key/value pairs που αφορούν την προσθήκη εγγραφών στο ευρετήριο, όσο και για αυτά που αφορούν τη διαγραφή εγγραφών. Η συνάρτηση partitioning αρχικά διαβάζει τα όρια και για τις δύο κατηγορίες και, στη συνέχεια, για κάθε κλειδί αναγνωρίζει την κατηγορία στην οποία αυτό αντιστοιχεί και επιλέγει τον reducer που έχει αναλάβει την επεξεργασία του διαστήματος όπου ανήκει το κλειδί αυτό. Συγκεκριμένα για R reducers διαχωρίζουμε το εύρος τιμών της κάθε κατηγορίας key/value pairs σε R διαστήματα, τα οποία ορίζονται από R-1 όρια, τα οποία συμβολίζουμε με Bound[i] και BoundDelete[i] αντίστοιχα. Η συνάρτηση partitioning αναγνωρίζει την κατηγορία στην οποία ανήκει το key/value pair και, με βάση τον παραπάνω συμβολισμό, αν πρόκειται για key/value pair που αφορά την προσθήκη εγγραφής στο ευρετήριο, παίρνει την εξής μορφή:

$$f(key) = \begin{cases} i-1, & \alpha v\ key < Bound[i-1], i \in [1, R-2] \\ R-1, & \alpha v\ key > Bound[R-1] \end{cases}$$

Όπου key η τιμή του κλειδιού για το οποίο αναζητείται ο αρμόδιος reducer.

Ενώ αν πρόκειται για key/value pair που αφορά τη διαγραφή εγγραφής από το ευρετήριο η μορφή της είναι η εξής:

$$f(key) = \begin{cases} i-1, & \alpha v\ key < BoundDelete[i-1], i \in [1, R-2] \\ R-1, & \alpha v\ key > BoundDelete[R-1] \end{cases}$$

Όπου key η τιμή του κλειδιού για το οποίο αναζητείται ο αρμόδιος reducer.

Η τροποποιημένη αυτή διαδικασία δειγματοληψίας οδηγεί στα επιθυμητά αποτελέσματα, και επιτυγχάνει τον διαχωρισμό του εύρους τιμών των λέξεων σε διαστήματα τέτοια ώστε το φορτίο των αντίστοιχων reducers να είναι περίπου ίδιο. Η αποτελεσματικότητα της αποδεικνύεται και πειραματικά, αφού παρά την Zipfian κατανομή των λέξεων, ο χρόνος ολοκλήρωσης των reduce tasks κατά την ενημέρωση του ευρετηρίου είναι περίπου ίδιος για όλα τα tasks.

Τέλος, είναι αναγκαίο να υπενθυμίσουμε ότι τα key/value pairs που αφορούν τη φόρτωση του περιεχομένου των κειμένων και την κατασκευή του forward index διανέμονται στους reducers με βάση τη διαδικασία που περιγράψαμε στην ενότητα που αφορά την αρχική δημιουργία του ευρετηρίου και δεν απαιτούν τη χρήση δειγματοληψίας.

4.5 Συμπεράσματα – Αποτελεσματικότητα της διαδικασίας δημιουργίας και ενημέρωσης του ευρετηρίου

Προκειμένου να αξιολογήσουμε θεωρητικά την αποτελεσματικότητα της διαδικασίας δημιουργίας και ενημέρωσης του ευρετηρίου, όπως αυτή παρουσιάστηκε στις προηγούμενες ενότητες, είναι απαραίτητο να εξετάσουμε το βαθμό στον οποίο η διαδικασία αυτή ικανοποιεί τις απαιτήσεις που καθορίστηκαν κατά την περιγραφή του προβλήματος.

Οι βασικές απαιτήσεις για τη ταχύτερη κατασκευή και ενημέρωση του ευρετηρίου, καθώς και για την ταχύτερη επεξεργασία των ερωτημάτων των χρηστών ήταν οι εξής:

- Η αποθήκευση του ευρετηρίου πρέπει να γίνεται με τρόπο που να επιτρέπει την ανάκτηση μικρών τμημάτων της λίστας κάθε λέξης, με στόχο την μείωση του χρόνου ανάκτησης των δεδομένων.
- Η διαδικασία ανάκτησης των δεδομένων για την επιστροφή αποτελεσμάτων στους χρήστες είναι απαραίτητο να είναι κατανεμημένη, με στόχο το διαμοιρασμό του μεγάλου φορτίου ερωτημάτων.
- Οι διαδικασίες κατασκευής, ενημέρωσης πρέπει να είναι κατανεμημένες και να κλιμακώνουν για μεγάλο όγκο δεδομένων, καθώς και μεγάλο αριθμό υπολογιστών.

- Είναι απαραίτητο ο χρόνος εκτέλεσης της ενημέρωσης του ευρετηρίου να είναι σημαντικά μικρότερος σε σχέση με την εξ αρχής δημιουργία του, αφού σε αντίθετη περίπτωση η όλη διαδικασία δεν έχει κανένα πρακτικό αποτέλεσμα.
- Η διαδικασία ενημέρωσης πρέπει να έχει πολυπλοκότητα, αλλά και χρόνο εκτέλεσης, ανεξάρτητο από το μέγεθος του υπάρχοντος ευρετηρίου, με στόχο την δυνατότητα συχνών ενημερώσεων του ευρετηρίου ακόμα και για μικρές συλλογές τροποποιημένων και νέων κειμένων.
- Η διαδικασία ενημέρωσης πρέπει να οδηγεί το ευρετήριο σε συνεπή κατάσταση, πανομοιότυπη με αυτή που θα είχε αν είχε δημιουργηθεί απευθείας από την τροποποιημένη συλλογή κειμένων και δεν είχε πραγματοποιηθεί καμία ενημέρωση. Ακόμα, πρέπει να απελευθερώνεται ο αποθηκευτικός χώρος που δεν περιέχει χρήσιμα δεδομένα. Με τον τρόπο αυτό εξασφαλίζεται τόσο η λειτουργικότητα του συστήματος, όσο και η σταθερότητα της απόδοσης του, η οποία δεν επηρεάζεται από τον αριθμό των ενημερώσεων.

Η διαδικασία κατασκευής και ενημέρωσης που σχεδιάσαμε ικανοποιεί όλες τις παραπάνω απαιτήσεις και άρα μπορεί να θεωρηθεί αποτελεσματική. Αναλυτικότερα:

- Όπως περιγράφθηκε στην αντίστοιχη ενότητα, ο τρόπος αποθήκευσης του ευρετηρίου που επιλέχθηκε, σύμφωνα με τον οποίο κάθε στήλη μιας γραμμής αναπαριστά ένα στοιχείο της λίστας της αντίστοιχης λέξης, επιτρέπει την ανάκτηση οποιουδήποτε αριθμού στοιχείων της λίστας και άρα ενός οσοδήποτε μικρού τμήματος αυτής.
- Με την αποθήκευση του ευρετηρίου στην HBase, η ανάκτηση των δεδομένων γίνεται κατανεμημένα, με αποδοτικό τρόπο, αξιοποιώντας τις δυνατότητες της κατανεμημένης αυτής βάσης δεδομένων.
- Με τη χρήση του MapReduce framework οι διαδικασίες δημιουργίας και ενημέρωσης είναι κατανεμημένες και έχουν τη δυνατότητα να κλιμακώνουν για μεγάλο όγκο δεδομένων και μεγάλο αριθμό υπολογιστών, χωρίς να απαιτείται η τροποποίηση των αλγορίθμων που χρησιμοποιούνται
- Η διαδικασία ενημέρωσης επεξεργάζεται μόνο τα τροποποιημένα ή νέα κείμενα με αποτέλεσμα ο όγκος των δεδομένων εισόδου και επομένως και ο απαιτούμενος χρόνος επεξεργασίας να μειώνονται σημαντικά. Ακόμα, με τη χρήση του αλγορίθμου σύγκρισης, η ενημέρωση του ευρετηρίου επιταχύνεται σε μεγάλο

βαθμό. Τα πειραματικά αποτελέσματα της επόμενης ενότητας επιβεβαιώνουν την αποδοτικότητα της διαδικασίας ενημέρωσης.

- Από την αναλυτική περιγραφή της διαδικασίας ενημέρωσης του ευρετηρίου, είναι φανερό ότι η διαδικασία αυτή δεν εξαρτάται σε γενικές γραμμές από το μέγεθος του υπάρχοντος ευρετηρίου, αφού η επεξεργασία αφορά μόνο τα τροποποιημένα ή νέα κείμενα. Τα μόνα σημεία, όπου το μέγεθος του υπάρχοντος ευρετηρίου επηρεάζει το χρόνο εκτέλεσης της ενημέρωσης, είναι αυτά όπου έχουμε ανάκτηση δεδομένων από τη βάση δεδομένων, δηλαδή κατά την φόρτωση του forward index για κάθε τροποποιημένο κείμενο, αλλά και κατά την εκτέλεση των διαγραφών των απαραίτητων στηλών. Ωστόσο, η HBase είναι σχεδιασμένη να λειτουργεί αποδοτικά για ιδιαίτερα μεγάλο αριθμό γραμμών αλλά και στηλών, με αποτέλεσμα η αύξηση του χρόνου ανάκτησης των δεδομένων, εξαιτίας της αύξησης του μεγέθους των πινάκων, να είναι αμελητέα, και άρα η διαδικασία ενημέρωσης να μπορεί να θεωρηθεί πρακτικά ανεξάρτητη από το μέγεθος του υπάρχοντος ευρετηρίου. Η ιδιότητα αυτή αποδεικνύεται και πειραματικά στην επόμενη ενότητα.
- Από την απόδειξη της ορθότητας της διαδικασίας ενημέρωσης που παρουσιάστηκε σε προηγούμενη ενότητα γίνεται κατανοητό ότι η διαδικασία ενημέρωσης οδηγεί το ευρετήριο σε συνεπή μορφή. Ταυτόχρονα, με τη διαγραφή των στηλών που αντιστοιχούν σε εγγραφές οι οποίες δεν συμφωνούν με τις νέες εκδόσεις των κειμένων, ο δεσμευμένος αποθηκευτικός χώρος, που δεν περιέχει πλέον χρήσιμη πληροφορία, απελευθερώνεται και, σε συνδυασμό με την προσθήκη των απαραίτητων στηλών που αφορούν νέες εγγραφές του ευρετηρίου, το ευρετήριο παίρνει ακριβώς της μορφή που θα είχε, αν είχε μόλις δημιουργηθεί εξ αρχής με βάση τη νέα συλλογή κειμένων.

Τα παραπάνω χαρακτηριστικά του συστήματος αποδεικνύουν ότι η διαδικασία κατασκευής και ενημέρωσης που προτείνεται ικανοποιεί τις απαιτήσεις που είχαμε προδιαγράψει και οδηγεί στα επιθυμητά αποτελέσματα. Τα πειραματικά δεδομένα που παρουσιάζονται στην επόμενη ενότητα είναι σύμφωνα με την θεωρητική αυτή ανάλυση, επιβεβαιώνοντας, με τον τρόπο αυτό, ότι η συγκεκριμένη διαδικασία είναι ιδιαίτερα αποτελεσματική.

Μειονεκτήματα

Παρά τα θετικά χαρακτηριστικά της διαδικασίας που παρουσιάσαμε, τα οποία οδήγησαν και στην επιλογή της ως καταλληλότερη, η διαδικασία εμφανίζει και ορισμένα αρνητικά χαρακτηριστικά τα οποία είναι απαραίτητο να παρουσιαστούν. Τα βασικά αρνητικά χαρακτηριστικά της μεθόδου είναι τα εξής:

- Ο τρόπος αποθήκευσης του ευρετηρίου που παρουσιάσαμε οδηγεί στην αύξηση του απαιτούμενου αποθηκευτικού χώρου του πίνακα στη βάση δεδομένων. Αυτό συμβαίνει διότι για κάθε στήλη που δημιουργείται σε μία γραμμή, η HBase αποθηκεύει επιπλέον πληροφορίες, όπως το timestamp του αντίστοιχου κελιού, και ανανεώνει το ευρετήριο της ώστε να συμπεριλάβει τη νέα αυτή στήλη. Έτσι, ο απαιτούμενος χώρος για την αποθήκευση του πίνακα αυξάνεται σε σχέση με την περίπτωση που χρησιμοποιούμε μία μόνο στήλη για την αποθήκευση ολόκληρης της λίστας της κάθε λέξης.
- Η αποθήκευση του ευρετηρίου με το συγκεκριμένο τρόπο επιβραδύνει την ανάκτηση των δεδομένων στην περίπτωση όπου απαιτείται η ανάκτηση ενός ιδιαίτερα μεγάλου τμήματος ή και ολόκληρης της λίστας μίας λέξης. Στην περίπτωση αυτή, ο νέος τρόπος αποθήκευσης, ο οποίος είναι κατάλληλα σχεδιασμένος για την ταχύτερη ανάκτηση μικρών τμημάτων της λίστας, καθιστά απαραίτητη τη ανάκτηση ενός μεγάλου αριθμού από στήλες, η οποία είναι πιο αργή σε σχέση με την περίπτωση όπου είχαμε ολόκληρη τη λίστα αποθηκευμένη σε μία μόνο στήλη.
- Τέλος, για την αποθήκευση του forward index της συλλογής απαιτείται επιπλέον αποθηκευτικός χώρος. Η χρήση το forward index θα μπορούσε βέβαια να παραληφθεί, αλλά αυτό θα καθιστούσε απαραίτητη τη σάρωση της παλιάς έκδοσης κάθε τροποποιημένου κειμένου για κάθε ενημέρωση, και θα επιβράδυνε σημαντικά τη διαδικασία ενημέρωσης.

Κεφάλαιο 5

Πειραματικά αποτελέσματα

Προκειμένου να ελέγξουμε την αποδοτικότητα των διαδικασιών δημιουργίας και ενημέρωσης ενός ανεστραμμένου ευρετηρίου, οι οποίες παρουσιάστηκαν στις προηγούμενες ενότητες, αλλά και να επιβεβαιώσουμε τα χαρακτηριστικά της κάθε διαδικασίας, τα οποία αναλύθηκαν θεωρητικά, υλοποιήσαμε ένα σύστημα κατανεμημένης δημιουργίας και ενημέρωσης ευρετηρίων και εκτελέσαμε έναν αριθμό πειραμάτων για δεδομένα μεταβαλλόμενου όγκου και για διαφορετικό αριθμό κόμβων. Τα αποτελέσματα των πειραμάτων αυτών μας επιτρέπουν να καταλήξουμε σε χρήσιμα συμπεράσματα, καθώς και να αξιολογήσουμε τα πλεονεκτήματα που προκύπτουν από τη χρήση των διαδικασιών που παρουσιάστηκαν στην εργασία αυτή σε ένα πραγματικό σύστημα.

Για την εκτέλεση των πειραμάτων χρησιμοποιήσαμε μια συστοιχία 9 υπολογιστών. Προκειμένου να εξασφαλίσουμε τη σωστή λειτουργία του συστήματος[22], ένας από τους κόμβους αυτούς είχε το ρόλο του Master-συντονιστή του συστήματος και λειτουργούσε ως NameNode και JobTracker, όσον αφορά τη λειτουργία του Hadoop DFS και MapReduce αντίστοιχα, αλλά και ως HMaster, όσον αφορά την HBase. Οι υπόλοιποι 8 κόμβοι λειτουργούσαν ως DataNodes, TaskTrackers και RegionServers. Η επιλογή αυτή έγινε προκειμένου να διατηρηθεί σε χαμηλά επίπεδα το φορτίο επεξεργασίας στον Master node, αφού η επιβάρυνση του θα επιβράδυνε τη λειτουργία του συστήματος συνολικά. Οι υπολογιστές που χρησιμοποιήθηκαν αποτελούνται από δύο τετραπύρηνους επεξεργαστές Intel Xeon στα 2GHz με 12MB L2 cache και διέθεταν 8GB μνήμης. Το λειτουργικό των υπολογιστών ήταν Debian Linux 64bit. Το δίκτυο της συστοιχίας ήταν τύπου Ethernet με εύρος 1Gbps.

Σε κάθε κόμβο, εκτός του Master, επιλέξαμε να έχουμε μέγιστο όριο 6 map tasks και 6 reduce tasks, το οποίο, αν συνυπολογίσουμε τις διεργασίες που αφορούν τους DataNode, TaskTracker και RegionServer, συνεπάγεται ότι έχουμε περίπου δύο διεργασίες ανά πυρήνα. Με τον τρόπο αυτό αυξήσαμε τη χρησιμοποίηση των υπολογιστικών πόρων, επικαλύπτοντας το χρόνο αναμονής για ανάγνωση και εγγραφή δεδομένων[23]. Με τη ρύθμιση αυτή, έχοντας 8 υπολογιστές να λειτουργούν ως TaskTrackers, το framework έχει

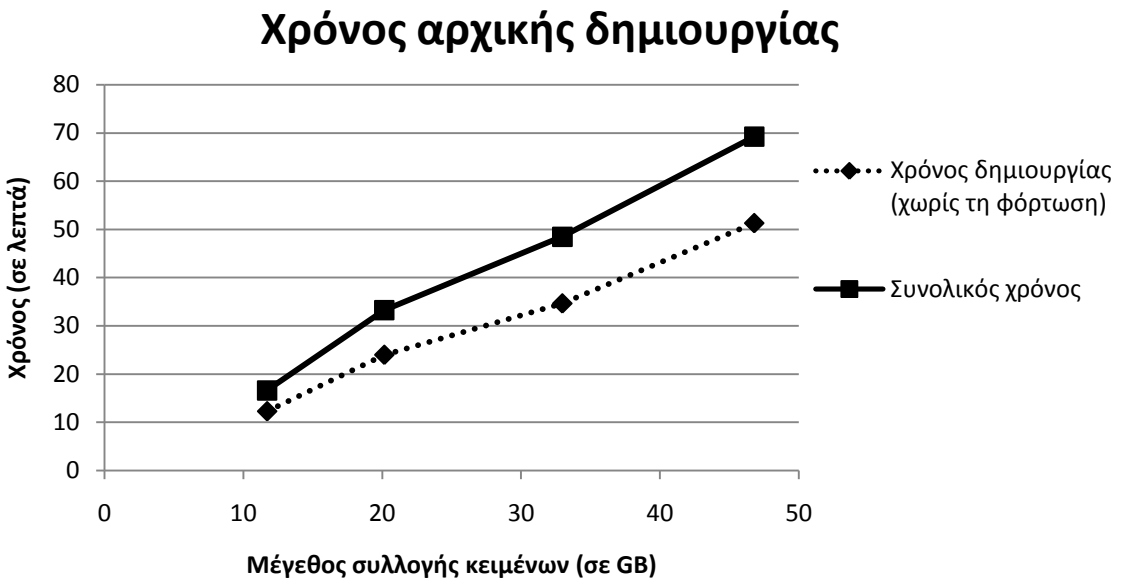
τη δυνατότητα να εκτελεί ταυτόχρονα 48 map tasks και 48 reduce tasks. Προκειμένου επιτύχουμε καλύτερη κατανομή του φορτίου στους reducers, επιλέξαμε σε κάθε εργασία να έχουμε αριθμό reducers περίπου ίσο με 1,75*48. Η επιλογή αυτή μας επιτρέπει να αποφύγουμε προβλήματα που προκύπτουν από τις ιδιομορφίες του hardware ορισμένων κόμβων, οι οποίες μπορεί να επιβραδύνουν την εκτέλεση της διαδικασίας. Ακόμα αυξήσαμε το μέγεθος του Java heap για τους reducers προκειμένου να μπορέσουν να διαχειριστούν το μεγάλο όγκο δεδομένων, χωρίς να υπάρχουν προβλήματα μνήμης.

Τα δεδομένα που χρησιμοποιήθηκαν για την εκτέλεση των πειραμάτων προέρχονται από τη Wikipedia και, πιο συγκεκριμένα, αποτελούν αντίγραφα όλων των κειμένων της Wikipedia σε μία συγκεκριμένη ημερομηνία. Για τη δημιουργία του ευρετηρίου χρησιμοποιήσαμε δεδομένα που αφορούσαν το περιεχόμενο της Wikipedia στις 5/4/2011, ενώ για την ενημέρωση του ευρετηρίου χρησιμοποιήσαμε δεδομένα που αφορούσαν το περιεχόμενο της Wikipedia στις 26/5/2011, ώστε να ενημερώσουμε το υπάρχον ευρετήριο με βάση τα κείμενα που έχουν τροποποιηθεί, αλλά και τα νέα κείμενα που έχουν εισαχθεί, στο χρονικό αυτό διάστημα.

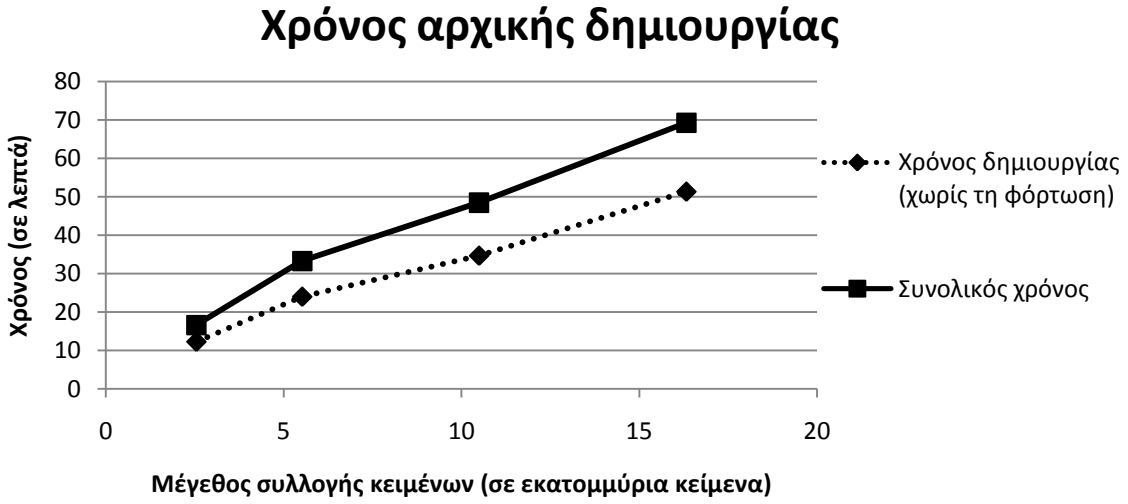
5.1 Αρχική δημιουργία του ανεστραμμένου ευρετηρίου

Σε αυτή την ομάδα πειραμάτων εξετάζουμε τη λειτουργία του συστήματος κατά τη δημιουργία ενός ανεστραμμένου ευρετηρίου, αυξάνοντας σταδιακά τον όγκο της συλλογής κειμένων που χρησιμοποιείται ως είσοδος.

Στα παρακάτω διαγράμματα παρουσιάζεται (α) ο χρόνος που απαιτήθηκε για να δημιουργηθεί το ευρετήριο, αλλά και (β) ο συνολικός χρόνος για να δημιουργηθεί και να φορτωθεί το ευρετήριο στη βάση δεδομένων, ώστε να μπορεί να χρησιμοποιηθεί άμεσα για την επεξεργασία των ερωτημάτων των χρηστών, σε συνάρτηση με τον όγκο της συλλογής, αλλά και τον αριθμό των κειμένων της συλλογής:



Σχήμα 14: Χρόνος δημιουργίας του ευρετηρίου σε συνάρτηση με το μέγεθος της συλλογής σε GB



Σχήμα 15: Χρόνος δημιουργίας του ευρετηρίου σε συνάρτηση με το μέγεθος της συλλογής σε εκατομμύρια κειμένων

Όπως φαίνεται από τα παραπάνω διαγράμματα, ο χρόνος που απαιτείται για τη δημιουργία του ευρετηρίου εξαρτάται γραμμικά από το μέγεθος της συλλογής των κειμένων. Το αποτέλεσμα αυτό είναι σύμφωνο με τη θεωρητική ανάλυση που έχει προηγηθεί, αφού η επεξεργασία των δεδομένων είναι γραμμική, με εξαίρεση την ταξινόμηση των λέξεων για τη δημιουργία του ανεστραμμένου ευρετηρίου, η οποία ωστόσο απαιτεί αμελητέο χρόνο σε σχέση με τη διάρκεια της υπόλοιπης επεξεργασίας. Ακόμα, η επεξεργασία και η μεταφορά των δεδομένων που εκτελείται από το MapReduce framework, με εξαίρεση την ταξινόμηση των key/value pairs πριν από τη φάση του reduce, είναι επίσης γραμμική ως προς το

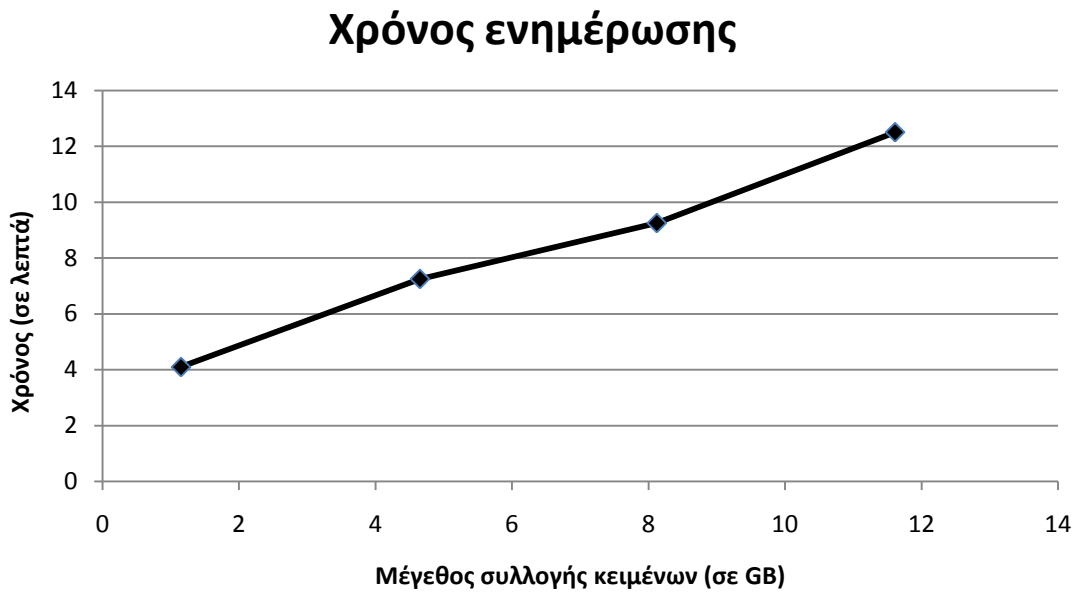
μέγεθος της εισόδου. Έτσι, παρότι απαιτείται η ταξινόμηση των key/value pair, τελικά ο χρόνος ολοκλήρωσης της διαδικασίας δημιουργίας παραμένει γραμμικός σε σχέση με το μέγεθος της εισόδου. Αυτό συμβαίνει διότι ο υπόλοιπος χρόνος επεξεργασίας και μεταφοράς των δεδομένων είναι σημαντικά μεγαλύτερος σε σχέση με το χρόνο που απαιτείται για την ταξινόμηση από το framework, το οποίο επεξεργάζεται τα δεδομένα ως key/value pairs, χωρίς να εκτελεί χρονοβόρες διαδικασίες στο εσωτερικό τους.

Παρατηρούμε ακόμα ότι ο χρόνος για τη φόρτωση του ευρετηρίου στη βάση δεδομένων αυξάνεται με την αύξηση του μεγέθους της συλλογής. Αυτό είναι λογικό αφού ο χρόνος φόρτωσης εξαρτάται από το μέγεθος του ευρετηρίου και κατ' επέκταση από το μέγεθος της συλλογής. Για το λόγο αυτό η γραμμή που προσεγγίζει το συνολικό χρόνο δημιουργίας και φόρτωσης του ευρετηρίου εμφανίζει μεγαλύτερη κλίση σε σχέση με αυτή του χρόνου δημιουργίας χωρίς τη φόρτωση.

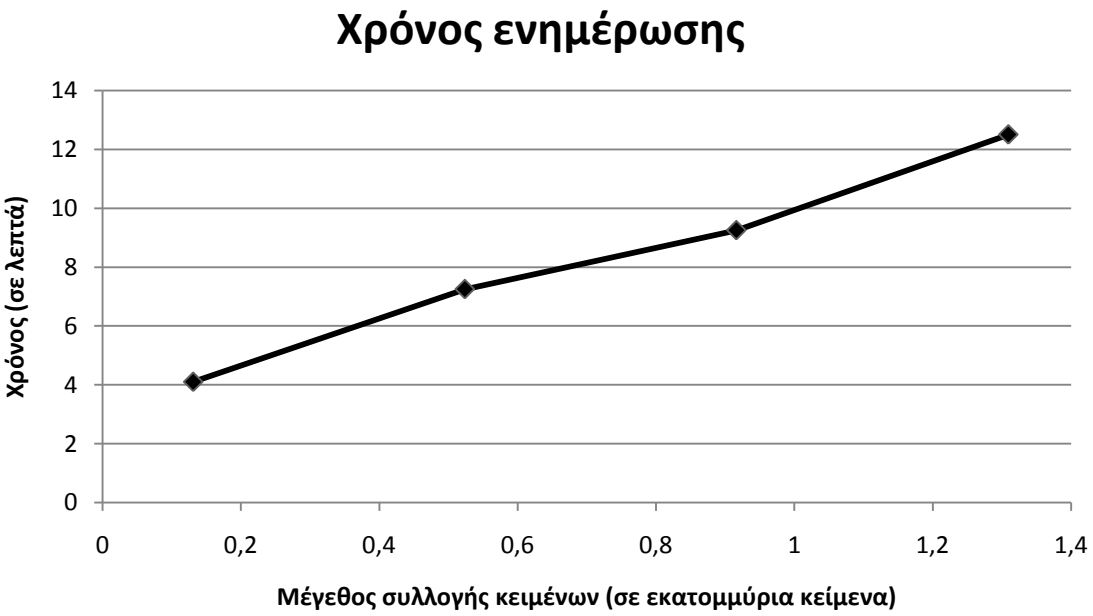
5.2 Ενημέρωση του ανεστραμμένου ευρετηρίου

Σε αυτή την ομάδα πειραμάτων εξετάζουμε τη λειτουργία του συστήματος κατά την ενημέρωση ενός ανεστραμμένου ευρετηρίου, διατηρώντας τη συλλογή με βάση την οποία δημιουργήθηκε το υπάρχον ευρετήριο σταθερή στα 46,8GB (και 16,3 εκατομμύρια κείμενα) και αυξάνοντας σταδιακά τον όγκο της συλλογής των τροποποιημένων ή νέων κειμένων.

Στα παρακάτω διαγράμματα παρουσιάζεται ο χρόνος που απαιτήθηκε για να ενημερωθεί το ευρετήριο και γίνουν οι απαραίτητες αλλαγές στη βάση δεδομένων, ώστε να μπορεί να χρησιμοποιηθεί άμεσα για την επεξεργασία των ερωτημάτων των χρηστών, σε συνάρτηση με τον όγκο της συλλογής, αλλά και τον αριθμό των κειμένων αυτής. Επιλέγουμε να μην παρουσιάσουμε ξεχωριστά το χρόνο ενημέρωσης χωρίς τη φόρτωση των δεδομένων, όπως στην περίπτωση της δημιουργίας, αφού ο χρόνος φόρτωσης, κατά την ενημέρωση, είναι εξαιρετικά μικρός και δεν προκύπτει ουσιαστική διαφορά στο διάγραμμα.



Σχήμα 16: Χρόνος ενημέρωσης του ευρετηρίου σε συνάρτηση με το μέγεθος της συλλογής νέων ή τροποποιημένων κειμένων σε GB



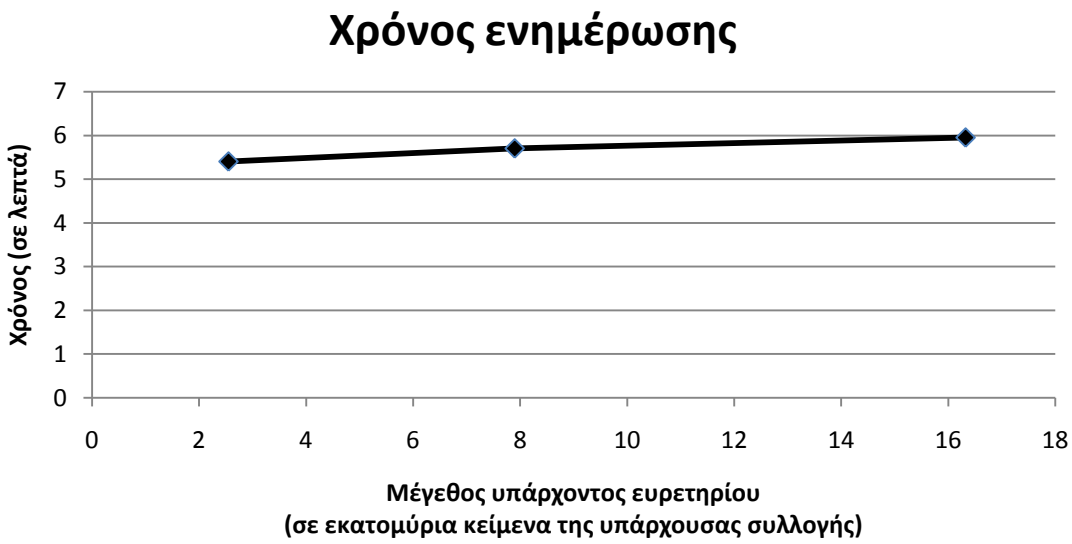
Σχήμα 17: Χρόνος ενημέρωσης του ευρετηρίου σε συνάρτηση με το μέγεθος της συλλογής νέων ή τροποποιημένων κειμένων σε εκατομμύρια κείμενα

Όπως φαίνεται από τα παραπάνω διαγράμματα, ο χρόνος που απαιτείται για την ενημέρωση του ευρετηρίου εξαρτάται γραμμικά από το μέγεθος της συλλογής των νέων ή τροποποιημένων κειμένων. Το αποτέλεσμα αυτό είναι σύμφωνο με τη θεωρητική ανάλυση της διαδικασίας. Όπως και κατά τη δημιουργία του ευρετηρίου, οι μη γραμμικές διαδικασίες δεν φαίνεται να επηρεάζουν σημαντικά το χρόνο εκτέλεσης του ευρετηρίου, ο

οποίος, για τους λόγους που αναφέρθηκαν προηγουμένως, είναι τελικά γραμμικός ως προς το μέγεθος της εισόδου. Ακόμα πρέπει να σημειώσουμε ότι στην περίπτωση αυτή η φόρτωση των δεδομένων στη βάση δεν είναι χρονοβόρα, αφού πρέπει να φορτωθούν μόνο οι νέες εγγραφές και όχι ολόκληρο το ευρετήριο όπως προηγουμένως.

Κατά την περιγραφή της διαδικασίας ενημέρωσης αναφέρθηκε ότι η πολυπλοκότητα της διαδικασίας είναι ανεξάρτητη του μεγέθους του υπάρχοντος ευρετηρίου, και επομένως, για σταθερό μέγεθος της συλλογής νέων ή τροποποιημένων κειμένων, ο χρόνος εκτέλεσης της ενημέρωσης θα πρέπει να είναι σταθερός ανεξάρτητα από το μέγεθος του αρχικού ευρετηρίου. Προκειμένου να επιβεβαιώσουμε τον ισχυρισμό αυτό εκτελέσαμε μετρήσεις διατηρώντας σταθερό το μέγεθος της νέας συλλογής κειμένων, καθώς και τον αριθμό των αλλαγών, αλλά μεταβάλλοντας το μέγεθος της αρχικής συλλογής.

Στο διάγραμμα που ακολουθεί παρουσιάζεται ο χρόνος που απαιτήθηκε προκειμένου να ενημερωθεί το ευρετήριο και να φορτωθούν οι αλλαγές στη βάση δεδομένων για σταθερό μέγεθος της συλλογής νέων ή τροποποιημένων κειμένων ίσο με 5,14GB (και περίπου 400 χιλιάδες κείμενα) σε συνάρτηση με τον όγκο της αρχικής συλλογής:



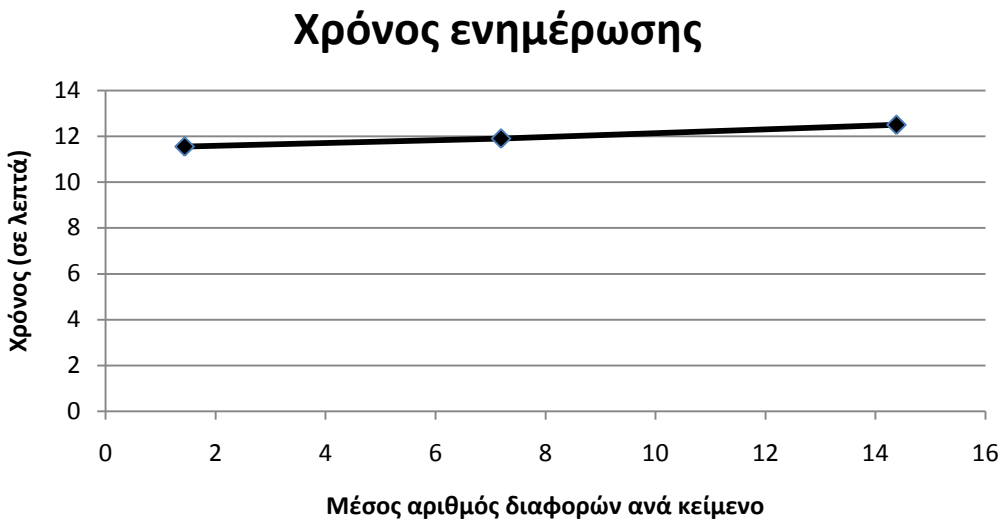
Σχήμα 18: Χρόνος ενημέρωσης του ευρετηρίου σε συνάρτηση με το μέγεθος του υπάρχοντος ευρετηρίου σε εκατομμύρια κείμενα

Από το παραπάνω διάγραμμα φαίνεται ότι ο χρόνος ενημέρωσης του ευρετηρίου είναι πρακτικά ανεξάρτητος από το μέγεθος του υπάρχοντος ευρετηρίου. Για 8 φορές μεγαλύτερο μέγεθος της αρχικής συλλογής, έχουμε μία αύξηση περίπου 9% στον χρόνο ενημέρωσης, γεγονός που μας επιτρέπει να θεωρήσουμε ότι το μέγεθος της αρχικής συλλογής δεν

επηρεάζει τη διαδικασία ενημέρωσης. Αυτό σημαίνει ότι η διαδικασία ενημέρωσης εξαρτάται ουσιαστικά μόνο από το μέγεθος της συλλογής νέων και τροποποιημένων κειμένων, επιτρέποντας έτσι την ταχεία ενημέρωση ευρετηρίων ιδιαίτερα μεγάλων ευρετηρίων. Η μικρή αύξηση που παρατηρείται οφείλεται στην σημαντική αύξηση του όγκου των δεδομένων που βρίσκονται αποθηκευμένα στη βάση δεδομένων. Όπως περιγράφθηκε κατά την ανάλυση της διαδικασίας ενημέρωσης, προκειμένου να εκτελεστεί η ενημέρωση, απαιτείται (α) η ανάκτηση του forward index της παλιάς έκδοσης κάθε κειμένου που έχει τροποποιηθεί, αλλά και (β) η διαγραφή από τη βάση δεδομένων όλων των εγγραφών που αφορούν τα κείμενα της υπάρχουσας συλλογής και είναι ασύμβατες με τα τροποποιημένα κείμενα. Η αύξηση του μεγέθους της βάσης επιβραδύνει ελαφρά την ανάκτηση και διαγραφή των δεδομένων, με αποτέλεσμα εξαιτίας του μεγάλου αριθμού των προσβάσεων στη βάση να εμφανίζεται αυτή η μικρή αύξηση του χρόνου ενημέρωσης.

Τέλος, όπως αναλύθηκε σε προηγούμενη ενότητα, η διαδικασία ενημέρωσης που παρουσιάζεται σε αυτή την εργασία βασίζεται στη σύγκριση των διαφορετικών εκδόσεων των κειμένων. Για το λόγο αυτό, είναι απαραίτητο να μελετήσουμε κατά πόσο η αύξηση των διαφορών μεταξύ των κειμένων της παλιάς και νέας συλλογής επηρεάζει το χρόνο εκτέλεσης της διαδικασίας ενημέρωσης. Δεδομένου ότι τα κείμενα των συλλογών που χρησιμοποιήσαμε έχουν χρονική διαφορά ενάμισι μήνα μεταξύ τους, ενώ ταυτόχρονα η Wikipedia αποτελεί ένα ιδιαίτερα δημοφιλές site με συνεχείς ανανεώσεις, μπορούμε να θεωρήσουμε ότι ο μέσος αριθμός των διαφορών ανά κείμενο, που εμφανίζεται στο χρονικό αυτό διάστημα, θα είναι μεγαλύτερος από τον μέσο αριθμό διαφορών ανά κείμενο που θα χρειαστεί να αντιμετωπίσουμε, σε πραγματικές συνθήκες, κατά την ενημέρωση ενός ανεστραμμένου ευρετηρίου. Έτσι επιλέξαμε να μην αυξήσουμε τεχνητά τον αριθμό των διαφορών, αλλά να περιορίσουμε τον αριθμό τους, εξαναγκάζοντας το σύστημα να απορρίπτει ένα ποσοστό των διαφορών κατά τη σύγκριση των κειμένων.

Στο παρακάτω διάγραμμα παρουσιάζεται ο χρόνος εκτέλεσης της ενημέρωσης του ευρετηρίου σε συνάρτηση με το μέσο αριθμό διαφορών ανά κείμενο που ανιχνεύτηκαν μεταξύ των κειμένων της παλιάς και νέας συλλογής:



Σχήμα 19: Χρόνος ενημέρωσης του ευρετηρίου σε συνάρτηση με το μέσο αριθμό διαφορών ανά κείμενο μεταξύ των δύο εκδόσεων των κειμένων

Όπως αναμενόταν, ο αριθμός των διαφορών φαίνεται να επηρεάζει τον χρόνο ολοκλήρωσης της ενημέρωσης. Παρόλα αυτά, η αύξηση του χρόνου ενημέρωσης είναι αμελητέα ακόμα και για μεγάλη αύξηση του αριθμού διαφορών. Το γεγονός αυτό οφείλεται στο ότι η επεξεργασία των τροποποιημένων κειμένων και η σύγκριση τους με τα κείμενα της υπάρχουσας συλλογής είναι ιδιαίτερα χρονοβόρες, ενώ ταυτόχρονα ο αριθμός των διαφορών ανά κείμενο είναι ιδιαίτερα μικρός. Έτσι, παρότι η αύξηση των αλλαγών συνεπάγεται αύξηση των ενδιάμεσων key/value pairs, τα οποία πρέπει να ομαδοποιηθούν και να ταξινομηθούν από το framework, καθώς και τον τελικών key/value pairs, τα οποία χρησιμοποιούνται για την προσθήκη και διαγραφή εγγραφών από τη βάση δεδομένων, ο συνολικός αριθμός τους παραμένει σε κάθε περίπτωση χαμηλός με αποτέλεσμα, ο χρόνος εκτέλεσης της ενημέρωσης να μην εμφανίζει σημαντική αύξηση.

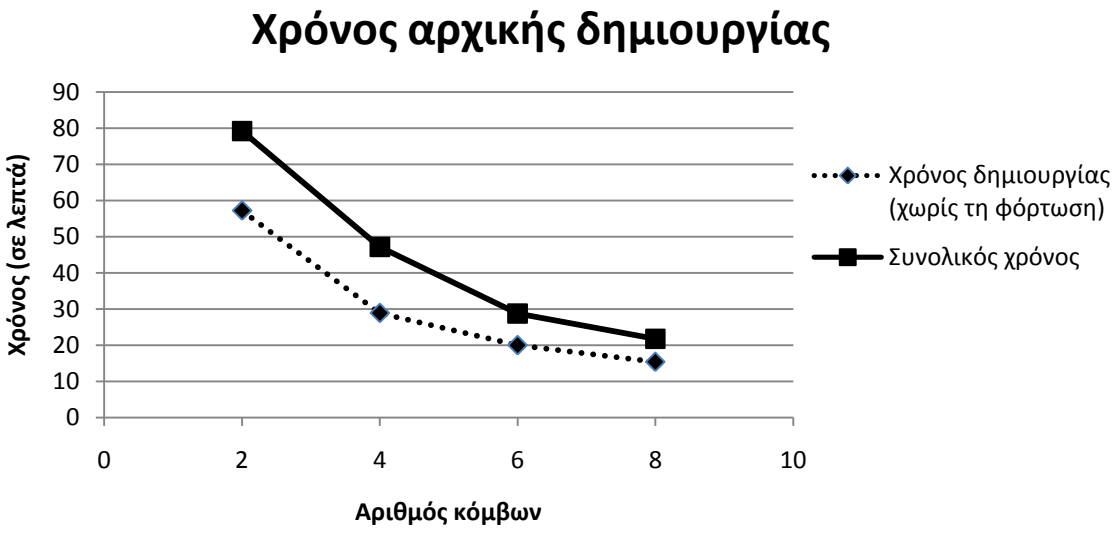
5.3 Κλιμακωσιμότητα του συστήματος

Ένα από τα βασικότερα χαρακτηριστικά μιας κατανεμημένης εφαρμογής είναι η δυνατότητα της να κλιμακώνει για μεγαλύτερους αριθμούς κόμβων. Σε γενικές γραμμές, οι εφαρμογές οι οποίες ταιριάζουν στο μοντέλο MapReduce έχουν τη δυνατότητα αυτή, αφού περιλαμβάνουν την ανεξάρτητη επεξεργασία των δεδομένων εισόδου. Ωστόσο, το όφελος από την αύξηση του αριθμού των κόμβων δεν είναι το ίδιο για κάθε εφαρμογή. Για το λόγο αυτό μελετήσαμε τη μεταβολή του χρόνου ολοκλήρωσης τόσο της διαδικασίας αρχικής δημιουργίας του ευρετηρίου, όσο και της διαδικασίας ενημέρωσης του, σε συνάρτηση με

τον αριθμό των κόμβων του συστήματος. Δυστυχώς, τα μηχανήματα του εργαστηρίου που διέθεταν την απαιτούμενη μνήμη ήταν λίγα, με αποτέλεσμα να περιοριστούμε σε μετρήσεις με μέχρι 8 κόμβους επεξεργασίας (με ένα επιπλέον κόμβο ως Master). Πρέπει να τονίσουμε ότι στα διαγράμματα που ακολουθούν ο αριθμός των κόμβων αναφέρεται στον αριθμό των υπολογιστικών κόμβων, δηλαδή των κόμβων που λειτουργούν ως TaskTrackers. Σε όλες τις περιπτώσεις χρησιμοποιούταν ένας επιπλέον κόμβος ως Master.

5.3.1 Αρχική δημιουργία του ευρετηρίου

Στο παρακάτω διάγραμμα παρουσιάζεται ο χρόνος δημιουργίας του ευρετηρίου, αλλά και ο συνολικός χρόνος για τη δημιουργία και τη φόρτωση του στη βάση δεδομένων:



Σχήμα 20: Χρόνος δημιουργίας του ευρετηρίου σε συνάρτηση με τον αριθμό των υπολογιστικών κόμβων

Όπως φαίνεται από το διάγραμμα, ο χρόνος δημιουργίας του ευρετηρίου μειώνεται σημαντικά καθώς αυξάνεται ο αριθμός των υπολογιστικών κόμβων. Ο διπλασιασμός των κόμβων του συστήματος οδήγησε σχεδόν σε υποδιπλασιασμό, τόσο του χρόνου δημιουργίας του ευρετηρίου, όσο και του συνολικού χρόνου, συμπεριλαμβανομένης της φόρτωσης των δεδομένων στη βάση.

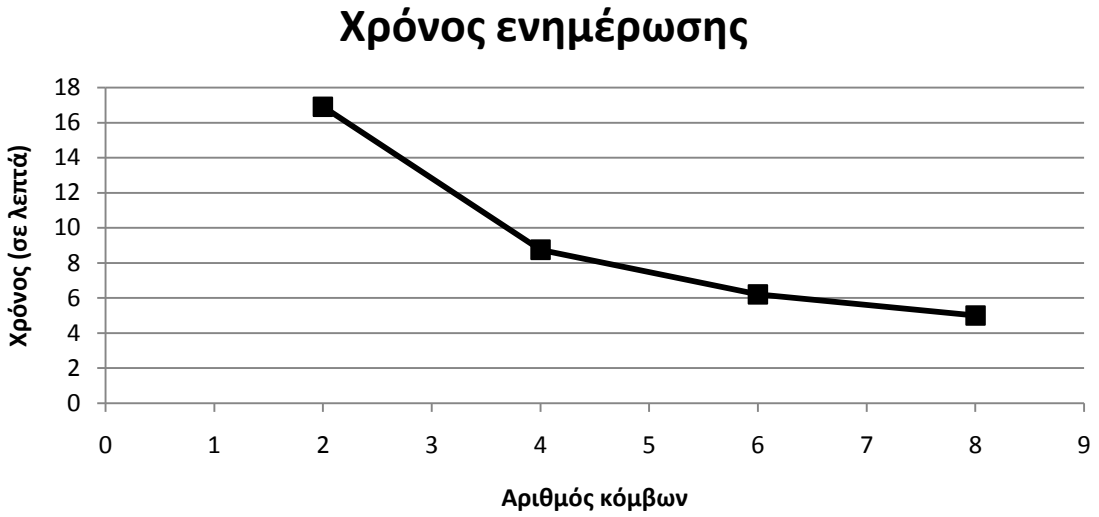
Στην εφαρμογή που εξετάζουμε, τα δεδομένα εισόδου, δηλαδή τα κείμενα της συλλογής, είναι ανεξάρτητα μεταξύ τους. Αυτό σημαίνει ότι, με τη αύξηση των υπολογιστικών κόμβων έχουμε τη δυνατότητα να επεξεργαστούμε παράλληλα τα δεδομένα αυτά κατά τη φάση του map. Έτσι, κατά τη φάση αυτή, ο χρόνος ολοκλήρωσης της επεξεργασίας είναι πρακτικά αντιστρόφως ανάλογος του αριθμού των υπολογιστικών κόμβων. Το ίδιο ισχύει

και για την φάση της ταξινόμησης των ενδιάμεσων key/value pairs, αλλά και για τη φάση του reduce, όπου και εκεί υπάρχει ανεξαρτησία των δεδομένων του κάθε reducer, με την προϋπόθεση βέβαια ότι χρησιμοποιείται μια καλά σχεδιασμένη συνάρτηση partitioning, ώστε κάθε reducer να αναλάβει ένα περίπου ίσο τμήμα των δεδομένων. Από την άλλη πλευρά, η αύξηση των κόμβων του συστήματος αυξάνει το κόστος επικοινωνίας στο στάδιο που παρεμβάλλεται ανάμεσα στις φάσεις map και reduce. Τα ενδιάμεσα δεδομένα, που προέκυψαν ως έξοδος της φάσης map, πρέπει να μεταφερθούν στους κατάλληλους reducers. Η αύξηση των κόμβων του συστήματος συνεπάγεται αύξηση του αριθμού των reducers που εκτελούνται σε απομακρυσμένα μηχανήματα και κατ' επέκταση αύξηση του όγκου των μεταφερόμενων δεδομένων, με αποτέλεσμα ο συνολικός χρόνος της διαδικασίας να μην είναι ακριβώς αντιστρόφως ανάλογος του αριθμού των κόμβων.

Όσον αφορά τη διαδικασία φόρτωσης των δεδομένων στη βάση, η αύξηση των κόμβων του συστήματος συνεπάγεται αύξηση των DataNodes και RegionServers του συστήματος. Αυτό σημαίνει ότι κάθε DataNode και κάθε RegionServer αναλαμβάνει να φορτώσει στη βάση ένα μικρότερο τμήμα των δεδομένων, με αποτέλεσμα ο χρόνος φόρτωσης να μειώνεται.

5.3.2 Ενημέρωση του ευρετηρίου

Στο παρακάτω διάγραμμα παρουσιάζεται ο χρόνος ενημέρωσης του συστήματος σε σχέση με τον αριθμό των υπολογιστικών κόμβων:



Σχήμα 21: Χρόνος ενημέρωσης του ευρετηρίου σε συνάρτηση με τον αριθμό των υπολογιστικών κόμβων

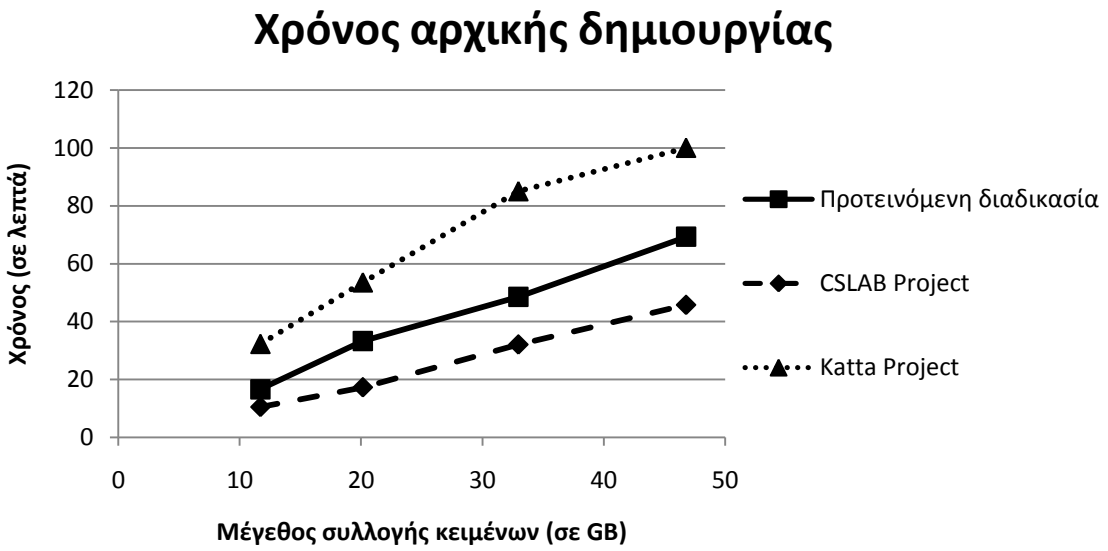
Και στην περίπτωση της ενημέρωσης του ευρετηρίου παρατηρούμε σημαντική μείωση του χρόνου ολοκλήρωσης της διαδικασίας, καθώς αυξάνεται ο αριθμός των κόμβων του συστήματος. Ο χρόνος ενημέρωσης είναι ουσιαστικά αντιστρόφως ανάλογος του αριθμού των υπολογιστικών κόμβων. Η αύξηση του όγκου των δεδομένων που πρέπει να μεταφερθούν μεταξύ των κόμβων κατά τη διαδικασία ενημέρωσης είναι λιγότερο σημαντική σε σχέση με τη διαδικασία δημιουργίας του ευρετηρίου, αφού ο αριθμός των ενδιάμεσων αποτελεσμάτων είναι μικρότερος. Έτσι, η εφαρμογή έχει τη δυνατότητα να κλιμακώνει με επιτυχία για μεγαλύτερο αριθμό κόμβων, γεγονός που την καθιστά κατάλληλη για μεγάλα συστήματα πολλών κόμβων.

5.4 Σύγκριση με άλλα συστήματα

Προκειμένου να ελέγξουμε τις επιδόσεις του συστήματος που σχεδιάσαμε και υλοποιήσαμε, επιλέξαμε να τις συγκρίνουμε με τις επιδόσεις άλλων σχετικών συστημάτων, τα οποία είναι ελεύθερα διαθέσιμα στο κοινό. Δυστυχώς, τη χρονική περίοδο που έγιναν τα πειράματα δεν υπήρχε διαθέσιμο κάποιο σύστημα ελεύθερου λογισμικού για την κατανεμημένη ενημέρωση ανεστραμμένων ευρετηρίων και για το λόγο αυτό θα συγκρίνουμε το σύστημα μας μόνο ως προς την αρχική δημιουργία ενός τέτοιου ευρετηρίου.

Το πρώτο σύστημα το οποίο χρησιμοποιήσαμε για την αξιολόγηση του συστήματος μας έχει αναπτυχθεί από την ομάδα των Κατανεμημένων Συστημάτων του εργαστηρίου Υπολογιστικών Συστημάτων (CSLAB) του Ε.Μ.Π.[11]. Το σύστημα αυτό αποτελεί υλοποίηση της διαδικασίας δημιουργίας που περιγράψαμε στην ενότητα 4.3.1, ως βάση για την παρουσίαση της διαδικασίας που προτείνεται στην παρούσα εργασία. Στο σύστημα αυτό έχουν γίνει κάποιες τροποποιήσεις για τον συνδυασμό της φόρτωσης του περιεχομένου των κειμένων, αλλά και της δημιουργίας του ανεστραμμένου ευρετηρίου σε μία MapReduce εργασία, με σκοπό την επιτάχυνση της όλης διαδικασίας. Το δεύτερο σύστημα που χρησιμοποιήσαμε είναι το Katta το οποίο παρουσιάστηκε στην ενότητα Κεφάλαιο 2, και αποτελεί project του sourceforge.net. Κανένα από τα δύο αυτά συστήματα δεν διαθέτει τη δυνατότητα ενημερώσεων και επομένως θα εκτελεστούν πειράματα μόνο όσον αφορά τη δημιουργία του ευρετηρίου.

Στο παρακάτω διάγραμμα παρουσιάζεται ο χρόνος δημιουργίας του ευρετηρίου, συμπεριλαμβάνοντας όλες τις απαραίτητες διαδικασίες, ώστε αυτό να είναι τελικά έτοιμο να δεχτεί τα ερωτήματα των χρηστών:



Σχήμα 22: Σύγκριση του συστήματος ως προς το χρόνο δημιουργίας του ευρετηρίου, σε σχέση με δύο άλλα διαθέσιμα συστήματα ελεύθερου λογισμικού

Όπως είχαμε αναλύσει και στην ενότητα 4.3.1, η διαδικασία δημιουργίας του ευρετηρίου που χρησιμοποιείται από το project του CSLAB είναι αποδοτικότερη από την διαδικασία που παρουσιάζεται στην παρούσα εργασία. Αυτό συμβαίνει διότι η διαδικασία αυτή δεν περιλαμβάνει τη δημιουργία του forward index των κειμένων της συλλογής και ταυτόχρονα ο τρόπος αποθήκευσης που χρησιμοποιεί οδηγεί στη μείωση του όγκου των τελικών αποτελεσμάτων και κατ' επέκταση του χρόνου φόρτωσης τους στη βάση δεδομένων. Το βασικό μειονέκτημά της διαδικασίας αυτής είναι ότι καθιστά αδύνατη την αποδοτική ενημέρωση του ευρετηρίου, με αποτέλεσμα να απαιτείται η ανακατασκευή του ευρετηρίου με στόχο την ενημέρωση του, η οποία είναι ιδιαίτερα χρονοβόρα ειδικά όταν το μέγεθος της συλλογής είναι μεγάλο. Στο παραπάνω διάγραμμα παρατηρούμε ότι η διαδικασία δημιουργίας ολοκληρώνεται περίπου στο 65% του χρόνου που απαιτείται από την προτεινόμενη διαδικασία. Η διαφορά αυτή είναι σημαντική, ωστόσο πρέπει να υπενθυμίσουμε ότι η προτεινόμενη διαδικασία έχει ως κύριο στόχο την αποδοτικότερη ενημέρωση του ευρετηρίου και όχι την ταχύτερη δημιουργία του. Η δημιουργία του ευρετηρίου μπορεί να πραγματοποιηθεί μόνο μία φορά και, στη συνέχεια, να εκτελούνται συνεχείς ενημερώσεις, με αποτέλεσμα ο χρόνος αρχικής δημιουργίας να μην είναι ιδιαίτερα σημαντικός. Παρατηρούμε ότι, παρότι το project του CSLAB έχει τη δυνατότητα να

δημιουργεί ταχύτερα το ευρετήριο, ο χρόνος δημιουργίας του ευρετηρίου, ακόμα και για μικρά μεγέθη της συλλογής, είναι αρκετά υψηλότερος σε σχέση με το χρόνο που απαιτείται για την ενημέρωση του, με βάση την προτεινόμενη διαδικασία. Συμπεραίνουμε, λοιπόν, ότι στην περίπτωση που απαιτείται σχετικά συχνή ενημέρωση του ευρετηρίου ή το μέγεθος του είναι μεγάλο, η προτεινόμενη διαδικασία υπερτερεί σημαντικά, καθιστώντας εφικτή την γρήγορη ενημέρωση του.

Από την άλλη πλευρά, το Katta χρειάζεται σημαντικά περισσότερο χρόνο για τη δημιουργία του ευρετηρίου. Αναλυτικότερα, το Katta αρχικά διαιρεί τη συλλογή των κειμένων σε ίσα τμήματα, τα οποία μοιράζει στους mappers ώστε αυτοί χρησιμοποιώντας το Apache Lucene να κατασκευάσουν το ευρετήριο για το μικρό αυτό τμήμα δεδομένων που τους ανατέθηκε. Το κάθε ένα από τα επιμέρους αυτά ευρετήρια ονομάζεται shard. Η διαδικασία αυτή είναι ιδιαίτερα γρήγορη αφού τα αποτελέσματα αποθηκεύονται απευθείας από τους mappers και δεν απαιτείται ταξινόμηση και μεταφορά των δεδομένων. Ακόμα το Apache Lucene είναι μια ισχυρή εφαρμογή δημιουργίας ευρετηρίων για μεγάλες συλλογές κειμένων, με αποτέλεσμα η δημιουργία των επιμέρους ευρετηρίων να είναι εξαιρετικά γρήγορη. Στη συνέχεια, ωστόσο, είναι απαραίτητο τα επιμέρους αυτά ευρετήρια να συνενωθούν, ώστε να σχηματιστεί το τελικό ευρετήριο με βάση το οποίο θα απαντώνται αποδοτικά τα ερωτήματα των χρηστών. Συγκεκριμένα, ο KattaMaster αναθέτει στους KattaNodes ένα σύνολο από shards και στη συνέχεια αυτοί κατεβάζουν από το HDFS τα shards που τους αντιστοιχούν στο τοπικό σύστημα αρχείων τους, τα ενσωματώνουν στον Lucene content server που διαθέτουν και τέλος ενημερώνουν τον KattaMaster, ώστε αυτός να γνωρίζει ποιο τμήμα του ευρετηρίου διαθέτει ο κάθε KattaNode και να μπορεί να χρησιμοποιήσει την πληροφορία αυτή κατά την απάντηση των ερωτημάτων των χρηστών. Η διαδικασία αυτή είναι εξαιρετικά αργή, αφού απαιτεί τη μεταφορά ενός μεγάλου όγκου δεδομένων από το HDFS στα τοπικά συστήματα αρχείων και ταυτόχρονα την συνένωση των επιμέρους ευρετηρίων. Έτσι, τελικά, ο συνολικός χρόνος για τη δημιουργία του ευρετηρίου είναι σημαντικά μεγαλύτερος σε σχέση με τις δύο άλλες μεθόδους. Τέλος, το Katta δεν υποστηρίζει ενημερώσεις του ευρετηρίου, με αποτέλεσμα να είναι απαραίτητο να δημιουργούμε εξ αρχής το ευρετήριο στην περίπτωση που επιθυμούμε να ανανεώσουμε το περιεχόμενο του, γεγονός που το καθιστά ακατάλληλο για μεγάλες συλλογές κειμένων, οι οποίες ενημερώνονται σχετικά συχνά.

Κεφάλαιο 6

Συμπεράσματα και μελλοντική έρευνα

6.1 Σύνοψη και συμπεράσματα

Τα τελευταία χρόνια, τα δεδομένα που είναι διαθέσιμα στο διαδίκτυο αυξάνονται με εκρηκτικούς ρυθμούς, ενώ ταυτόχρονα η συμμετοχή των απλών χρηστών στη δημιουργία νέων δεδομένων, μέσω των social networks για παράδειγμα, έχει ως αποτέλεσμα τη συνεχή ανανέωση του περιεχομένου του διαδικτύου. Για το λόγο αυτό είναι πλέον αναγκαία η διαρκής ενημέρωση των ανεστραμμένων ευρετηρίων των μηχανών αναζήτησης, ώστε τα αποτελέσματα των αναζητήσεων να περιλαμβάνουν τα νέα αυτά δεδομένα.

Στην παρούσα διπλωματική εργασία παρουσιάσαμε ένα σύστημα κατανεμημένης δημιουργίας και ενημέρωσης ανεστραμμένων ευρετηρίων, το οποίο αξιοποιεί τις δυνατότητες των κατανεμημένων αρχιτεκτονικών και του Cloud προκειμένου να επιταχύνει τις διαδικασίες δημιουργίας και ενημέρωσης για συλλογές κειμένων μεγάλης κλίμακας. Ακόμα, με σκοπό την αντιμετώπιση του αυξημένου φόρτου ερωτημάτων των χρηστών, το ευρετήριο που δημιουργείται αποθηκεύεται σε μία κατανεμημένη βάση δεδομένων, γεγονός που επιτρέπει την κατανομή του φόρτου ερωτημάτων σε ένα μεγάλο αριθμό κόμβων.

Δεδομένου του ταχύτατου ρυθμού με τον οποίον τροποποιούνται τα δεδομένα στο διαδίκτυο, καθώς και της απαίτησης των χρηστών για ενημερωμένα δεδομένα, θέσαμε ως κύριο στόχο του συστήματος την αποδοτική ενημέρωση του ευρετηρίου. Για το λόγο αυτό, η διαδικασία δημιουργίας, καθώς και ο τρόπος αποθήκευσης του ευρετηρίου, σχεδιάστηκαν με στόχο την επιτάχυνση της διαδικασίας ενημέρωσης.

Αναλυτικότερα, οι βασικές επιλογές είναι οι εξής:

- Αποθήκευση της λίστας από IDs κειμένων για κάθε λέξη του ευρετηρίου χρησιμοποιώντας μία γραμμή για κάθε λέξη και μια στήλη για κάθε ID της λίστας, γεγονός που μας επιτρέπει να αφαιρέσουμε απευθείας ένα κείμενο από τη λίστα μιας λέξης χωρίς να απαιτείται να διατρέξουμε τη λίστα αυτή. Ο συγκεκριμένος τρόπος αποθήκευσης βασίζεται στη δυνατότητα των NoSQL βάσεων δεδομένων να αποθηκεύουν και να διαχειρίζονται αποδοτικά έναν μεγάλο αριθμό στηλών, ο οποίος μπορεί μάλιστα να μεταβάλλεται ανάλογα με τη γραμμή. Η επιλογή αυτή

αυξάνει τις ανάγκες για αποθηκευτικό χώρο, αφού για κάθε στήλη είναι απαραίτητο να αποθηκευτούν ορισμένα επιπλέον δεδομένα στη βάση δεδομένων.

- Αποθήκευση του forward index για κάθε κείμενο της συλλογής, ώστε να μην είναι απαραίτητη η σάρωση της παλιάς έκδοσης κάθε κειμένου κατά τη διαδικασία ενημέρωσης. Η επιλογή αυτή αυξάνει, επίσης, τον απαιτούμενο αποθηκευτικό χώρο.

Οι επιλογές αυτές επιβάρυναν σε κάποιο βαθμό τη διαδικασία δημιουργίας και οδήγησαν σε αύξηση του απαιτούμενου αποθηκευτικού χώρου. Παρόλα αυτά, η διαδικασία ενημέρωσης του ευρετηρίου εκτελείται πολύ συχνότερα σε σχέση με τη διαδικασία δημιουργίας, η οποία μπορεί να εκτελεστεί μία μονό φορά. Ταυτόχρονα, το κόστος του αποθηκευτικού χώρου είναι πλέον σχετικά χαμηλό. Για τους λόγους αυτούς τα μειονεκτήματα που παρουσιάζουν οι παραπάνω επιλογές μπορούν να θεωρηθούν ήσσονος σημασίας σε σχέση με τα οφέλη των επιλογών αυτών για τη διαδικασία ενημέρωσης.

Η διαδικασία ενημέρωσης που σχεδιάσαμε εκμεταλλεύεται τις δυνατότητες που προκύπτουν από τις επιλογές αυτές και, όπως αναλύθηκε θεωρητικά, αλλά και επιβεβαιώθηκε από τα αποτελέσματα των αντίστοιχων πειραμάτων, διαθέτει ορισμένα σημαντικά χαρακτηριστικά, τα οποία την καθιστούν εξαιρετικά αποδοτική για την ενημέρωση ανεστραμμένων ευρετηρίων για μεγάλης κλίμακας συλλογές κειμένων.

Συγκεκριμένα, τα επιθυμητά χαρακτηριστικά που παρουσιάζει η προτεινόμενη μέθοδος ενημέρωσης είναι τα εξής:

- Ο χρόνος ενημέρωσης, αλλά και η υπολογιστική πολυπλοκότητα της διαδικασίας ενημέρωσης, είναι πρακτικά ανεξάρτητοι από το μέγεθος της υπάρχουσας συλλογής. Ο χρόνος ενημέρωσης επηρεάζεται ουσιαστικά μόνο από το μέγεθος της συλλογής των νέων ή τροποποιημένων κειμένων. Η μικρή αύξηση του χρόνου ενημέρωσης, η οποία εμφανίστηκε στα πειραματικά αποτελέσματα, οφείλεται, όπως περιγράφθηκε, στην ταχύτητα ανάκτησης των δεδομένων από τη βάση και όχι στον αλγόριθμο που χρησιμοποιήθηκε. Το γεγονός αυτό καθιστά την ενημέρωση του ευρετηρίου εξαιρετικά ταχύτερη σε σχέση με την αρχική δημιουργία του ευρετηρίου και επιτρέπει τη συχνότερη εκτέλεση των ενημερώσεων, ακόμα και για ευρετήρια που έχουν δημιουργηθεί από μεγάλες συλλογές κειμένων.
- Η μορφή του ευρετηρίου παραμένει αμετάβλητη ανεξάρτητα από τον αριθμό των ενημερώσεων που έχουν πραγματοποιηθεί. Με τον τρόπο αυτό εξασφαλίζεται η αξιοπιστία του συστήματος, αλλά και η σταθερότητα της απόδοσης του, αφού το

ευρετήριο έχει σε κάθε στιγμή τη μορφή που θα είχε αν είχε δημιουργηθεί από την τρέχουσα συλλογή κειμένων.

- Ο χρόνος ενημέρωσης εξαρτάται γραμμικά από το μέγεθος της συλλογής νέων ή τροποποιημένων κειμένων, γεγονός που καθιστά δυνατή την επεξεργασία μεγάλου όγκου δεδομένων και την ταχύτερη ενημέρωση του ευρετηρίου.
- Το σύστημα έχει τη δυνατότητα να κλιμακώνει ιδιαίτερα αποτελεσματικά σε μεγάλο αριθμό κόμβων με σκοπό τη μείωση του απαιτούμενου χρόνου για την ενημέρωση.

Τα χαρακτηριστικά αυτά, με εξαίρεση το δεύτερο το οποίο αναλύθηκε θεωρητικά στην ενότητα που αφορά τη διαδικασία ενημέρωσης, παρουσιάζονται και με γραφικό τρόπο από τα διαγράμματα που προέκυψαν με βάση τα πειραματικά αποτελέσματα. Για παράδειγμα, όπως φαίνεται από τα αντίστοιχα διαγράμματα, η αρχική δημιουργία ενός ανεστραμμένου ευρετηρίου από μία συλλογή μεγέθους 47GB, συμπεριλαμβανομένης της φόρτωσης του ευρετηρίου στη βάση δεδομένων, διαρκεί περίπου 70 λεπτά. Αντίθετα η ενημέρωση του ευρετηρίου αυτού για 11,6GB νέων και τροποποιημένων κειμένων, συμπεριλαμβανομένης της φόρτωσης των νέων δεδομένων στη βάση, διήρκησε μόνο 12,5 λεπτά. Ταυτόχρονα, από το διάγραμμα που παρουσιάζει το χρόνο ενημέρωσης για σταθερό μέγεθος της συλλογής νέων ή τροποποιημένων κειμένων και μεταβλητό μέγεθος της αρχικής συλλογής, φαίνεται ότι ο χρόνος ενημέρωσης ήταν περίπου σταθερός ανεξάρτητα από το μέγεθος της αρχικής συλλογής, σε αντίθεση με το χρόνο αρχικής δημιουργίας του ευρετηρίου ο οποίος εξαρτάται γραμμικά από το μέγεθος αυτό. Τέλος, από το διάγραμμα χρόνου δημιουργίας αλλά και ενημέρωσης του ευρετηρίου σε σχέση με τον αριθμό των υπολογιστικών κόμβων που χρησιμοποιήθηκαν, είναι εμφανές ότι το σύστημα έχει τη δυνατότητα να κλιμακώνει με μεγάλη επιτυχία για μεγαλύτερο αριθμό κόμβων, γεγονός που το καθιστά κατάλληλο για την επεξεργασία ιδιαίτερα μεγάλων συλλογών κειμένων σε κατανεμημένα περιβάλλοντα πολλών κόμβων.

Με βάση τη θεωρητική ανάλυση, αλλά και τα πειραματικά αποτελέσματα, που παρουσιάστηκαν, αποδεικνύεται η καταλληλότητα της προτεινόμενης διαδικασίας για την ενημέρωση ευρετηρίων μεγάλων συλλογών κειμένων, όπου η εξ αρχής δημιουργία του ευρετηρίου είναι εξαιρετικά χρονοβόρα. Αξιοποιώντας τις δυνατότητες των shared-nothing αρχιτεκτονικών πολλών κόμβων, όπως το Cloud, το σύστημα που παρουσιάστηκε καθιστά εφικτή την συχνότερη ενημέρωση των ανεστραμμένων ευρετηρίων με στόχο την επιστροφή πιο έγκυρων αποτελεσμάτων στους τελικούς χρήστες.

6.2 Προτάσεις για μελλοντική έρευνα

Παρότι το σύστημα που παρουσιάσαμε εκτελεί αποδοτικά την ενημέρωση ανεστραμμένων ευρετηρίων τα οποία απλά καταγράφουν τις λέξεις που περιέχονται σε κάθε κείμενο, η λειτουργία του ενδεχομένως να μην είναι το ίδιο αποδοτική στην περίπτωση όπου καταγράφεται και η θέση της κάθε λέξης μέσα στο κάθε κείμενο. Αυτό συμβαίνει διότι με την προσθήκη ή διαγραφή λέξεων, η θέση των υπόλοιπων λέξεων αλλάζει και επομένως, κατά τη διαδικασία ενημέρωσης, θα ήταν απαραίτητο να τροποποιηθούν οι εγγραφές του ευρετηρίου που αφορούν και άλλες λέξεις του κειμένου. Το γεγονός αυτό θα επιβάρυνε τη διαδικασία ενημέρωσης και, ενδεχομένως, θα την καθιστούσε αναποτελεσματική. Για την αντιμετώπιση του συγκεκριμένου προβλήματος έχουν προταθεί ορισμένες τεχνικές[24] για την κωδικοποίηση της θέσης κάθε λέξης με τρόπο τέτοιο ώστε η εισαγωγή ή η διαγραφή λέξεων του κειμένου να απαιτεί όσο το δυνατόν λιγότερες αλλαγές στο ευρετήριο. Θα ήταν, λοιπόν, ενδιαφέρον να χρησιμοποιηθούν τέτοιου είδους τεχνικές και στη συνέχεια να εξεταστεί εκ νέου η αποδοτικότητα της διαδικασίας ενημέρωσης.

Ακόμα, το ευρετήριο που δημιουργείται μέσω της διαδικασίας που παρουσιάστηκε δεν περιλαμβάνει καμία πληροφορία όσον αφορά την σχετικότητα κάθε κειμένου ως προς κάθε λέξη που αυτό περιέχει. Έτσι, κατά την αναζήτηση μίας ή περισσοτέρων λέξεων, δεν υπάρχει η δυνατότητα κατάταξης των αποτελεσμάτων ανάλογα με το πόσο αυτά σχετίζονται με τους όρους της αναζήτησης. Θα ήταν, λοιπόν, ενδιαφέρον να σχεδιαστεί μία μέθοδος αξιολόγησης της σχετικότητας κάθε κειμένου ως προς κάθε λέξη που αυτό περιέχει, με στόχο την κατάταξη των αποτελεσμάτων των αναζητήσεων, η οποία, όμως, να επιτρέπει την αποδοτική και κατανεμημένη ενημέρωση του ευρετηρίου.

Τέλος, θα ήταν ενδιαφέρον να εξετάσουμε τον τρόπο με τον οποίο η διαδικασία ενημέρωσης επηρεάζει το ρυθμό ανάκτησης δεδομένων από τη βάση, για την επεξεργασία των ερωτημάτων των χρηστών, καθώς και κατά πόσο θα μπορούσαμε να εκμεταλλευτούμε την ελαστικότητα του Cloud, προκειμένου να αντιμετωπίσουμε τυχόν μείωση του ρυθμού αυτού κατά τη διάρκεια της ενημέρωσης του ευρετηρίου, αυξάνοντας τον αριθμό των κόμβων ή τους διαθέσιμους πόρους του συστήματος.

Κεφάλαιο 7

Βιβλιογραφία

[1] "Inverted index," *http://en.wikipedia.org/wiki/Inverted_index*.

[2] S. Melink, S. Raghavan, B. Yang, and H. Garcia-Molina, "Building a distributed full-text index for the web," *ACM Trans. Inf. Syst.*, vol. 19, Jul. 2001, pp. 217–241.

[3] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, Jan. 2008, pp. 107–113.

[4] "Google Caffeine," *http://googleblog.blogspot.com/2010/06/our-new-search-index-caffeine.html*.

[5] "Twitter's New Search Architecture," *http://engineering.twitter.com/2010/10/twitters-new-search-architecture.html*.

[6] "Apache Lucene," *http://lucene.apache.org/*.

[7] "Katta - distributed lucene," *http://katta.sourceforge.net/*.

[8] "Apache Solr," *http://lucene.apache.org/solr/*.

[9] J. Lin, D. Metzler, T. Elsayed, and L. Wang, "Of Ivory and Smurfs: Loxodontan MapReduce experiments for web search," *Proc. of TREC*, 2009.

[10] N. Li, J. Rao, E. Shekita, and S. Tata, "Leveraging a scalable row store to build a distributed text index," *Proceeding of the first international workshop on Cloud data management*, Hong Kong, China: ACM, 2009, pp. 29–36.

[11] I. Konstantinou, E. Angelou, D. Tsoumakos, and N. Koziris, "Distributed indexing of web scale datasets for the cloud," *Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud*, Raleigh, North Carolina: ACM, 2010, pp. 1:1–1:6.

[12] "Hadoop Distributed File System," *http://hadoop.apache.org/hdfs/*.

[13] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," *Proceedings of the nineteenth ACM symposium on Operating systems principles*, Bolton Landing, NY, USA: ACM, 2003, pp. 29–43.

[14] "Hadoop MapReduce," *http://hadoop.apache.org/mapreduce/*.

[15] "HBase," *http://hbase.apache.org/*.

[16] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R.E. Gruber, "Bigtable: A Distributed Storage System for Structured Data," *ACM Trans. Comput. Syst.*, vol. 26, Jun. 2008, pp. 4:1–4:26.

[17] "Apache ZooKeeper," *http://zookeeper.apache.org/*.

[18] J. Lin, "The Curse of Zipf and Limits to Parallelization: A Look at the Stragglers Problem in MapReduce," *Proc. LSDS-IR*, 2009, pp. 57–62.

[19] "Zipf's law," *http://en.wikipedia.org/wiki/Zipf%27s_law*.

[20] A. Cary, Z. Sun, V. Hristidis, and N. Rishe, "Experiences on Processing Spatial Data with MapReduce," *Scientific and Statistical Database Management*, M. Winslett, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 302-319.

[21] Owen O'Malley, "TeraByte Sort on Apache Hadoop," 2008.

[22] "Hadoop Cluster Setup," *http://hadoop.apache.org/common/docs/current/cluster_setup.html*.

[23] T. White, *Hadoop: The Definitive Guide*, O'Reilly Media, Inc., 2010.

[24] L. Lim, M. Wang, S. Padmanabhan, J.S. Vitter, and R. Agarwal, "Dynamic maintenance of web indexes using landmarks," *Proceedings of the 12th international conference on World Wide Web*, Budapest, Hungary: ACM, 2003, pp. 102–111.