ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

Τμήμα Ηλεκτρονικών Μηχανικών & Μηχανικών Υπολογιστών

«Υλοποίηση Μηχανής Αναζήτησης βασισμένης στο PageRank με χρήση του Hadoop»
« Implementation of a Pagerank-based Search Engine using Hadoop»

Διπλωματική Εργασία

Γιώργος Φραγκιαδουλάκης

Εξεταστική Επιτροπή

Καθηγητής Γαροφαλάκης Μίνως **(Επιβλέπων)**

Καθηγητής Πετράκης Ευριπίδης

Επ. Καθηγητής Δεληγιαννάκης Αντώνιος

Οκτώβριος 2013

Κατάλογος περιεχομένων

1. Εισαγωγή....7
1.1 Αντικείμενο και Συνεισφορά....7
1.2 Δομή της Εργασίας....9
2. Background/Related Work....11
2.1 Hadoop/Map-Reduce....11
2.1.1 Προγραμματιστικό Μοντέλο Map-Reduce....11
2.1.2 Hadoop Distributed File System (HDFS)....15
2.2 Web Crawling....15
2.2.1 Web Crawling – Θεωρητικό Υπόβαθρο....15
2.2.2 Apache Nutch....17
2.3 Indexing και Vector Space Model....20
2.3.1 Κατασκευή του Inverted Index....20
2.3.2 Vector Space Model....22
2.3.3 Συναφή Συστήματα....24
2.4 Personalized PageRank....27
2.4.1 Original PageRank – Θεωρητικό Υπόβαθρο....27
2.4.2 Personalized PageRank – Θεωρητικό Υπόβαθρο....30
2.4.3 Συναφείς Μελέτες....30
3. Indexing και Vector Space Model στο Hadoop Map-Reduce....32
3.1 Ανάλυση των απαιτήσεων....32
3.2 Φάση Crawling....33
3.3 Επεξεργασία των δεδομένων....34
3.4 Κατασκευή του Inverted Index σε Map-Reduce....36
3.5 Υπολογισμός των tf-idf scores σε Map-Reduce....37
3.5.1 Φάση “WordCount”....38
3.5.2 Φάση “WordCountPerDocument-Normalized”....38
3.5.3 Φάση “Tf-Idf Calculation”....39
3.6 Κατασκευή του βελτιωμένου Inverted Index σε Map-Reduce....41
3.7 Υπολογισμός Ομοιότητας Συνημιτόνου (Cosine Similarity) σε Map-Reduce....42
4. Personalized PageRank και Συνδυασμός Μεθόδων Scoring στο Hadoop Map-Reduce....47
4.1 Ανάλυση των απαιτήσεων....47
4.2 Κατασκευή γράφου διαδικτύου (web graph)....48
4.3 Προϋπάρχουσες Υλοποιήσεις – Original PageRank....51
4.3.1 Κατασκευή δομημένης αναπαράστασης του γράφου στο Hadoop....51
4.3.2 Υλοποίηση του αλγορίθμου PageRank στο Hadoop....53
4.4 Πρόσθετες Υλοποιήσεις – Personalized PageRank....55
4.4.1 Εφαρμογή του κριτηρίου σύγκλισης στον αλγόριθμο PageRank....55
4.4.2 Κατανομή βαρών σε σελίδες με βάση τις προτιμήσεις των χρηστών....56
4.4.3 Προσθήκη εξατομίκευσης (personalization) στον αλγόριθμο PageRank....58
4.5 Συνδυασμός των scores: PageRank και Cosine Similarity....59
4.5.1 Εξαγωγή των ζευγών <id κόμβου-τιμή pagerank> και εγγραφή στο HDFS....60
4.5.2 Υπολογισμός ενοποιημένου score Cosine Similarity-Personalized PageRank . .61
4.5.3 Μετατροπή των ids στα αντίστοιχα urls....64
5. Πειραματική Διαδικασία....67

5.1 Δεδομένα Εισόδου και Μεθοδολογία των Πειραμάτων.....67
5.2 Αξιολόγηση Πειραματικών Αποτελεσμάτων.....71
5.2.1 Αποτελέσματα Original PageRank.....71
5.2.2 Αποτελέσματα Personalized PageRank.....73
5.2.3 Αποτελέσματα συνδυασμού Cosine Similarity - PageRank.....78
5.2.4 Χρόνος Εκτέλεσης Online Διεργασιών.....106
6. Συμπεράσματα και Future Work.....111

Περίληψη

Οι μηχανές αναζήτησης αποτελούν δίχως αμφιβολία αναπόσπαστο κομμάτι του παγκόσμιου ιστού. Τα αποτελέσματα μίας αναζήτησης με βάση το επερώτημα (query) του εκάστοτε χρήστη προκύπτουν από τη συλλογή, την επεξεργασία και την αξιολόγηση τεράστιου όγκου δεδομένων. Επομένως, το μέγεθος της πληροφορίας που καλούμαστε να διαχειριστούμε μας οδηγεί στη χρησιμοποίηση του προγραμματιστικού μοτίβου Map-Reduce και της υλοποίησής του στο Hadoop, με σκοπό την παράλληλη εκτέλεση των απαιτούμενων διεργασιών. Μετά την ολοκλήρωση της συλλογής των δεδομένων χρησιμοποιώντας το Nutch ως web crawler, κατασκευάσαμε τη δομή Inverted Index στο κατανεμημένο σύστημα αρχείων HDFS. Εκμεταλλευόμενοι το λεκτικό περιεχόμενο των ιστοσελίδων, υπολογίσαμε scores όπως το Tf-Idf και η Ομοιότητα Συνημιτόνου (Cosine Similarity) ώστε να καταταχθούν οι σελίδες που προκύπτουν από queries διαφόρων κατηγοριών. Στη συνέχεια, προχωρήσαμε στην εξατομίκευση (personalization) του αλγορίθμου PageRank, ώστε να υπολογιστούν επιπλέον scores για όλες τις σελίδες της συλλογής μας, λαμβάνοντας υπόψιν τις προτιμήσεις των χρηστών του συστήματος. Οι τιμές PageRank είναι ανεξάρτητες από το εκάστοτε query και επηρεάζονται μόνο από τη δομή του γράφου (webgraph) που απεικονίζει τους συνδέσμους μεταξύ των σελίδων. Τέλος, έγινε ο συνδυασμός των παραπάνω μεθόδων αξιολόγησης, με αποτέλεσμα την εξαγωγή ενός ενοποιημένου score. Διεξήχθησαν πειραματικές αναζητήσεις με βάση διαφορετικές περιπτώσεις, ώστε να προκύψουν όσο το δυνατόν πιο χρήσιμα συμπεράσματα σχετικά με την υλοποίησή μας.

Abstract

Search engines are undoubtedly an integral part of the Web. The search results based on the query of each user are derived from the collection, processing and evaluation of huge amounts of data. Therefore, the size of the information we need to manage leads us to the use of MapReduce programming model and its implementation on Hadoop for parallel execution of the necessary processes. After completion of data collection using Nutch web crawler, the Inverted Index structure was constructed in the distributed file system HDFS. Taking advantage of the web pages' text content, we calculated scores such as Tf-Idf and Cosine Similarity in order to classify the pages resulting from queries of various categories. Afterwards, we proceeded to the personalization of PageRank algorithm to calculate additional scores for all pages of the collection, taking into account the system users' preferences. The PageRank values are independent of each query and they are only influenced by the structure of the webgraph that represents the links between the pages. Finally, the above ranking methods were combined, leading to the calculation of a unified score. Experimental searches were conducted under different conditions in order to obtain the most useful conclusions possible about our implementation.

1. Εισαγωγή

1.1 Αντικείμενο και Συνεισφορά

Η αύξηση των χρηστών του διαδικτύου τα τελευταία χρόνια είναι αλματώδης, ενώ καθημερινά η πληροφορία που διατίθεται μέσω των ιστοσελίδων (web pages) πολλαπλασιάζεται. Αναμφισβήτητα, κάθε χρήστης έχει διαφορετικό εύρος ενδιαφερόντων και προτιμήσεων και αυτό ανακλάται στο είδος των ιστοσελίδων που επισκέπτεται. Η ανάγκη για εύκολη πρόσβαση στο περιεχόμενο των ιστοσελίδων επέφερε τη δημιουργία συστημάτων ανάκτησης πληροφοριών (information retrieval systems), τα οποία ονομάζουμε μηχανές αναζήτησης διαδικτύου (web search engines).

Τα πρώτα μεγάλα βήματα για την κατασκευή τους έγιναν στις αρχές της δεκαετίας του 90', ενώ η αρχή έγινε με τη λειτουργία των Lycos και WebCrawler το 1994. Το 1995 ιδρύθηκε η Yahoo!, ενώ ένα χρόνο αργότερα, αναπτύχθηκε η Altavista, η οποία στις πρώτες τρεις εβδομάδες λειτουργίας της, δεχόταν περίπου 2 εκατομμύρια επερωτήματα (queries) την ημέρα. Η πιο διαδεδομένη μηχανή αναζήτησης στο διαδίκτυο σήμερα, η Google, ιδρύθηκε το 1998 και αποτελεί μία από τις μεγαλύτερες εταιρίες παγκοσμίως.

Είναι απαραίτητο να αναφέρουμε τα κύρια στάδια λειτουργίας μίας τυπικής μηχανής αναζήτησης. Η πρωταρχική φάση αποτελείται από το Web Crawling, το οποίο ορίζεται ως η διαδικασία κατά την οποία συλλέγονται σελίδες από το διαδίκτυο ώστε να δεικτοδοτηθούν και να υποστηρίξουν τη μηχανή. Ο σκοπός του Web Crawling είναι να συλλεχθούν γρήγορα και αποτελεσματικά όσο το δυνατόν περισσότερες χρήσιμες ιστοσελίδες, καθώς επίσης και ο γράφος (webgraph) που περιλαμβάνει τις συνδέσεις (links) μεταξύ τους. Ένας Web Crawler πρέπει απαραίτητα να έχει τα εξής δύο χαρακτηριστικά: 1) Ευρωστία, που σημαίνει να είναι ανθεκτικός σε παγίδες όπως οι λεγόμενες spider traps, οι οποίες είναι γεννήτριες σελίδων που οδηγούν λανθασμένα τους crawlers στην επαναληπτική συλλογή σελίδων κάτω από μία ορισμένη διεύθυνση διαδικτύου (domain) και 2) "Ευγένεια", δηλαδή ο εκάστοτε crawler θα πρέπει να λαμβάνει υπόψιν τις πολιτικές των διακομιστών διαδικτύου (web servers) σχετικά με τη συχνότητα των επισκέψεων του σε αυτούς.

Η επόμενη απαραίτητη λειτουργία μίας μηχανής αναζήτησης είναι η Δεικτοδότηση (Indexing) των όρων που εμπεριέχονται στις ιστοσελίδες που συλλέχθηκαν από την παραπάνω διαδικασία του crawling. Στην ουσία, δημιουργείται ένα "ανεστραμμένο ευρετήριο" ή "ανεστραμμένο αρχείο"

(inverted index or inverted file), το οποίο τυπικά αποθηκεύει για κάθε όρο της συλλογής, το σύνολο των κειμένων (ιστοσελίδων) στα οποία τον συναντάμε καθώς επίσης και κάποιο βάρος (weight), όπως π.χ. το πλήθος των εμφανίσεων του όρου στο συγκεκριμένο κείμενο. Το πλεονέκτημα του Inverted Index είναι ότι εξασφαλίζει άμεση πρόσβαση στη λίστα όλων των ιστοσελίδων στις οποίες περιέχεται ένας όρος. Η παραπάνω μέθοδος είναι μία από τις πιο διαδεδομένες τεχνικές στην περιοχή της ανάκτησης πληροφορίας και χρησιμοποιείται από τις περισσότερες μηχανές αναζήτησης.

Με ποιο τρόπο όμως αξιολογεί μία μηχανή τη διαθέσιμη πληροφορία; Τα μοντέλα ανάκτησης πληροφορίας που έχουν επικρατήσει μέχρι τώρα είναι: το Boolean Model, το Vector Space Model (VSM) και το Probabilistic Model. Τα παραπάνω μοντέλα λειτουργούν με βάση το λεκτικό περιεχόμενο των ιστοσελίδων και εξάγουν τα πιο σχετικά αποτελέσματα, δεδομένου του επερωτήματος του χρήστη.

Πέρα από την αξιολόγηση της πληροφορίας που βρίσκεται μέσα στο εκάστοτε κείμενο μίας ιστοσελίδας, μπορούμε να εκμεταλλευτούμε τη δομή του γράφου που απεικονίζει τον τρόπο που συνδέονται οι σελίδες μεταξύ τους στον παγκόσμιο ιστό. Δύο αλγόριθμοι για την ταξινόμηση ιστοσελίδων με βάση τις συνδέσεις μεταξύ τους, οι PageRank και HITS (Hyperlink Induced Topic Search) αναπτύχθηκαν το 1996 στο Πανεπιστήμιο του Stanford και στην IBM Almaden και είχαν σκοπό να προσδώσουν μεγαλύτερη ακρίβεια στην αναζήτηση. Ο επαναληπτικός αλγόριθμος PageRank χρησιμοποιείται σήμερα από την Google, ενώ η βασική ιδέα είναι ότι κάθε σελίδα στο διαδίκτυο κατέχει ένα μέτρο "κύρους", το οποίο είναι ανεξάρτητο από τα επερωτήματα των χρηστών. Χονδρικά, το κύρος μίας σελίδας είναι ανάλογο του αθροίσματος των τιμών κύρους των σελίδων που περιέχουν συνδέσμους προς εκείνη.

Σχετικά με τον αλγόριθμο PageRank έχουν προταθεί επεκτάσεις οι οποίες βασίζονται στις προτιμήσεις του εκάστοτε χρήστη και με αυτό τον τρόπο συντελούν στην ανάπτυξη της ιδέας του Εξατομικευμένου (Personalized) PageRank. Το Personalized PageRank αναπτύχθηκε ως ιδέα από το Stanford University το 2002, ενώ κάποιες από τις παραλλαγές του οι οποίες είναι ευαίσθητες στην πληροφορία που περιέχει το επερώτημα του χρήστη ή/και το προφίλ του ίδιου του χρήστη είναι οι: Topic-Sensitive PageRank, Modular PageRank και BlockRank.

Πώς όμως ανταποκρίνεται ένα σύστημα αναζήτησης στις εκατομμύρια αιτήσεις για αναζήτηση πληροφορίας που υποβάλλουν καθημερινά οι χρήστες του διαδικτύου; Είναι φανερό ότι απαιτούνται αποθηκευτικοί χώροι που θα κρατούν τεράστιο όγκο πληροφοριών και μηχανήματα τα οποία θα έχουν την ικανότητα να εκτελούν με ταχύτητα τις απαραίτητες λειτουργίες. Αυτός είναι και ο λόγος που γεννήθηκε ένα σύστημα διαχείρισης μεγάλου όγκου δεδομένων, το Hadoop. Το Hadoop έχει τις ρίζες του στην open source μηχανή αναζήτησης Apache Nutch, η οποία άρχισε να κατασκευάζεται το 2002, ως μέρος της βιβλιοθήκης αναζήτησης κειμένου Apache Lucene. Το προγραμματιστικό μοτίβο Map-Reduce, στο οποίο στηρίζεται το Hadoop, μας δίνει τη δυνατότητα να διαχειριστούμε πληροφορία μεγάλου μεγέθους κατανέμοντας τις διεργασίες σε πολλούς κόμβους εργασίας παράλληλα με σκοπό την ανθεκτικότητα σε αστοχίες, ένα χαρακτηριστικό που το καθιστά ως ένα εξαιρετικά αξιόπιστο σύστημα.

Η βασική συνεισφορά της παρούσας διπλωματικής εργασίας η υλοποίηση ενός ολοκληρωμένου συστήματος μηχανής αναζήτησης βασισμένο στο PageRank, χρησιμοποιώντας το Hadoop. Επίσης, υλοποιείται ο Inverted Index και στη συνέχεια το VSM στο Hadoop, έτσι ώστε να εκμεταλλευτούμε το λεκτικό περιεχόμενο των σελίδων. Υπάρχει έτσι η δυνατότητα μελέτης της συμπεριφοράς του αλγορίθμου PageRank στο Hadoop και του τρόπου με τον οποίο το PageRank

score επηρεάζει τα αποτελέσματα της μηχανής αναζήτησης, αφού συνδυαστεί με το VSM.

Αρχικά, χρησιμοποιήσαμε το Apache Nutch ως crawler με σκοπό να λάβουμε πληροφορία για ένα ικανοποιητικό αριθμό ιστοσελίδων. Μετά από κατάλληλη επεξεργασία των δεδομένων, φτάσαμε στο σημείο να έχουμε στη διάθεση μας το λεκτικό περιεχόμενο των σελίδων, τη δομή του γράφου που όπως προαναφέραμε περιλαμβάνει όλες τις συνδέσεις μεταξύ των προκείμενων σελίδων, καθώς και πολλά άλλου είδους μεταδεδομένα, τα οποία μπορούν να φανούν χρήσιμα σε κάθε σύστημα επεξεργασίας πληροφοριών.

Στη συνέχεια, στηριχθήκαμε στο μοντέλο VSM και μέσα από συγκεκριμένα βήματα, προέκυψαν οι τιμές (scores) TF-IDF, οι οποίες δίνουν ένα μέτρο για τη σχέση ανάμεσα σε κάθε ζεύγος όρου και ιστοσελίδας-κείμενου της συλλογής μας. Με βάση τις παραπάνω τιμές, προχωρήσαμε στην κατασκευή του Inverted Index, ο οποίος συμπεριλαμβάνει και τα scores TF-IDF, ενώ το επόμενο βήμα ήταν να εξάγουμε το μέτρο γνωστό και ως Cosine Similarity, που καταδεικνύει το βαθμό σχετικότητας ανάμεσα σε ένα επερώτημα του χρήστη και στις διαθέσιμες ιστοσελίδες.

Συνεχίζοντας με το επόμενο κομμάτι της εργασίας μας που δεν είναι άλλο από την εκμετάλλευση της πληροφορίας που λαμβάνουμε από τον webgraph, υποθέσαμε ότι το σύστημα μας χρησιμοποιείται από 14 χρήστες, οι οποίοι έχουν διαφορετικά ενδιαφέροντα και προτιμούν να επισκέπτονται συγκεκριμένες ιστοσελίδες της συλλογής μας. Το γεγονός αυτό, μας οδήγησε στην παραλλαγή του αλγορίθμου PageRank και κατ' επέκταση στην εξαγωγή διαφορετικών εξατομικευμένων (personalized) scores για κάθε χρήστη.

Τελικά, επιχειρήσαμε να συνδυάσουμε τα scores που προέρχονται από το Cosine Similarity και το Personalized PageRank , δίνοντας επερωτήματα που περιέχουν συνήθεις ή/και μη συνήθεις όρους του index. Σκοπός των πειραμάτων είναι να εξετάσουμε κατά πόσο η μηχανή αναζήτησης που υλοποιήσαμε ταξινομεί σωστά τα εξαγόμενα αποτελέσματα ανάλογα με το εκάστοτε επερώτημα και τις προτιμήσεις του κάθε χρήστη.

Ένα αρκετά ενδιαφέρον ζήτημα προς εξέταση, λοιπόν, είναι ο τρόπος με τον οποίο μπορούμε να συνδυάσουμε διαφορετικές μεθόδους αξιολόγησης των αποτελεσμάτων της αναζήτησης και πιο συγκεκριμένα να εξάγουμε scores τα οποία θα συνδυάζουν το Vector Space Model και το Personalized PageRank. Αυτό έγινε εφικτό κάνοντας χρήση του Hadoop Cluster του εργαστηρίου SoftNet του Πολυτεχνείου Κρήτης.

1.2 Δομή της Εργασίας

Η διάρθρωση των κεφαλαίων που ακολουθούν έχει ως εξής: Το κεφάλαιο 2 ασχολείται με το θεωρητικό υπόβαθρο του Hadoop Map-Reduce (2.1) και του Web Crawling (2.2), καθώς επίσης με το θεωρητικό υπόβαθρο και τις συναφείς μελέτες όσον αφορά την κατασκευή του Inverted Index και το Vector Space Model (2.3) και το Personalized PageRank (2.4).

Στο κεφάλαιο 3, αφού γίνει η ανάλυση των απαιτήσεων (3.1), παρουσιάζεται η φάση του Crawling (3.2) και η επεξεργασία των δεδομένων που προκύπτουν από αυτό (3.3). Στη συνέχεια, αναλύεται η κατασκευή του Inverted Index, καθώς και ο υπολογισμός των tf-idf scores στο Hadoop (3.4, 3.5, 3.6), ενώ τελικά παρουσιάζεται ο τρόπος με τον οποίο υπολογίστηκε η Ομοιότητα Συνημιτόνου (Cosine Similarity) στο Hadoop (3.7).

Το κεφάλαιο 4 ασχολείται με την ανάλυση της υλοποίησης του αλγορίθμου PageRank (Original και Personalized) στο Hadoop (4.3, 4.4), αφού πρώτα παρουσιαστεί η ανάλυση των απαιτήσεων

(4.1) και η κατασκευή του γράφου διαδικτύου (4.2) που χρησιμοποιείται στον υπολογισμό του PageRank. Επιπρόσθετα, γίνεται αναλυτική περιγραφή της υλοποίησης που σχετίζεται με το συνδυασμό των μεθόδων Cosine Similarity και Personalized PageRank (4.5).

Στο κεφάλαιο 5 παρουσιάζεται αρχικά η μεθοδολογία των πειραμάτων που διεξήχθησαν (5.1) και στη συνέχεια γίνεται η αξιολόγηση και ο σχολιασμός των πειραματικών αποτελεσμάτων που εξήχθησαν (5.2), με βάση τις υλοποιήσεις που αναλύονται στα κεφάλαια 3 και 4.

Τέλος, στο κεφάλαιο 6 συνοψίζονται τα συμπεράσματα από την παρούσα διπλωματική εργασία και προτείνονται μελλοντικές επεκτάσεις που αφορούν αποδοτικότερες υλοποιήσεις του συστήματος.

2. Background/Related Work

2.1 Hadoop/Map-Reduce

Το Apache Hadoop [20] είναι μία βιβλιοθήκη λογισμικού γραμμένη σε Java που επιτρέπει την κατανεμημένη επεξεργασία μεγάλου όγκου δεδομένων μέσω μίας συστοιχίας υπολογιστών (cluster of computers) χρησιμοποιώντας ένα απλό προγραμματιστικό μοντέλο,το MapReduce. Το Hadoop δημιουργήθηκε από τον Doug Cutting και σήμερα χρησιμοποιείται ευρέως από εταιρίες όπως η Yahoo!,η Google, η Facebook και άλλες. Είναι σχεδιασμένο με τέτοιο τρόπο ώστε να λειτουργεί σε ένα ή σε χιλιάδες μηχανήματα, καθένα από τα οποία εκτελεί υπολογισμούς και αποθηκεύει δεδομένα σε τοπικό επίπεδο. Το μοτίβο MapReduce βοηθάει στην υλοποίηση εφαρμογών που επεξεργάζονται με ταχύτητα τεράστιο μέγεθος δεδομένων παράλληλα σε μεγάλα clusters από υπολογιστικούς κόμβους (compute nodes). Το κύριο σύστημα αποθήκευσης που χρησιμοποιείται από τις εφαρμογές σε Hadoop είναι το κατανεμημένο σύστημα αρχείων HDFS (Hadoop Distributed File System), το οποίο δημιουργεί πολλαπλά αντίγραφα των μπλοκ δεδομένων (data blocks) και τα διαμοιράζει στους υπολογιστικούς κόμβους μέσω του cluster για να επιτευχθούν αξιόπιστοι και εξαιρετικά γρήγοροι υπολογισμοί.

Κάποιες από τις σχετικές με το Hadoop εφαρμογές είναι οι: Avro, Cassandra, Chukwa, Hbase, Hive, Mahout, Pig και ZooKeeper.

2.1.1 Προγραμματιστικό Μοντέλο Map-Reduce

Το προγραμματιστικό μοντέλο Map-Reduce είναι σχεδιασμένο για την παράλληλη επεξεργασία μεγάλου όγκου δεδομένων σε cluster υπολογιστών.

Το Map-Reduce λειτουργεί χωρίζοντας την επεξεργασία σε 2 φάσεις: τη φάση Map και τη φάση Reduce, οι οποίες αντιστοιχούν στις συναρτήσεις map και reduce, των οποίων η λειτουργία εξαρτάται από τον εκάστοτε προγραμματιστή. Ως είσοδος δίνεται ένα σύνολο από ζευγάρια της μορφής κλειδί/τιμή (key/value), ενώ στην έξοδο παράγονται επίσης ζευγάρια της ίδιας μορφής.

Η συνάρτηση map λαμβάνει ως είσοδο ένα ζεύγος key/value και εξάγει ενδιάμεσα ζεύγη key/value. Όλες οι ενδιάμεσες τιμές (values) που αφορούν ένα κλειδί (key) ομαδοποιούνται και δίνονται ως είσοδος στη φάση reduce.

Η συνάρτηση reduce δέχεται σαν είσοδο ένα ενδιάμεσο κλειδί και ένα σύνολο από τιμές για αυτό το κλειδί. Όλες οι ενδιάμεσες τιμές για ένα συγκεκριμένο κλειδί παρέχονται μέσω ενός iterator. Αυτό μας επιτρέπει να χειριστούμε λίστες που είναι υπερβολικά μεγάλες για να χωρέσουν στην κύρια μνήμη. Τυπικά, σαν έξοδος παράγεται μία ή καμία τιμή σε κάθε κλήση της reduce.

Ενδεικτικά, παρουσιάζεται η γενική μορφή των παραπάνω συναρτήσεων:

Συνάρτηση map : (key1,value1) → list(key2, value2)

Συνάρτηση reduce : (key2, list(value2)) → list(key3, value3)

όπου: key1,value1 οι τύποι των δεδομένων εισόδου της συνάρτησης map

key2,value2 οι τύποι των δεδομένων εξόδου της συνάρτησης map και εισόδου της συνάρτησης reduce

και key3,value3 οι τύποι των δεδομένων εξόδου της συνάρτησης reduce

Ακολουθεί ένα παράδειγμα ψευδοκώδικα σε λογική Map-Reduce, το οποίο μετράει το πλήθος των εμφανίσεων κάθε λέξης (word count) σε μία πλειάδα από αρχεία:

```
void map(String name, String document):
        // name: document name
        // document: document contents
        for each word w in document:
                EmitIntermediate(w, "1");


void reduce(String word, Iterator partialCounts):
        // word: a word
        // partialCounts: a list of aggregated partial counts
        int sum = 0;
        for each pc in partialCounts:
                sum += ParseInt(pc);
        Emit(word, AsString(sum));
```

Πίνακας 2.1: Παράδειγμα WordCount

Κάθε φορά που εκτελείται η συνάρτηση map και έχουμε ως είσοδο ένα αρχείο, αυτό διαχωρίζεται σε λέξεις και έτσι στην έξοδο παράγονται ζεύγη της μορφής (word1,”1”),(word2,”1”), (word2,”1”) κοκ. Η είσοδος στη φάση reduce είναι της μορφής (word,list(“1”)), άρα για παράδειγμα, για τη λέξη word2 , η είσοδος θα είναι (word2,(“1”,”1”)). Στο σώμα της συνάρτησης παρατηρούμε ότι για κάθε λέξη, αθροίζεται το πλήθος των εμφανίσεων της σε όλα τα αρχεία και εξάγεται ζεύγος της μορφής (word,SumOfCounts).

Τα παραπάνω καταδεικνύουν τη λογική που εμπεριέχει σε γενικές γραμμές το μοτίβο Map-Reduce. Παρακάτω παρουσιάζονται συνοπτικά τα στάδια που λαμβάνουν χώρα από τη στιγμή που ξεκινάει να εκτελείται μία διεργασία (job) Map-Reduce έως το τέλος:

- **RecordReader:** Τα δεδομένα εισόδου χωρίζονται σε κομμάτια (input splits), τυπικά από 16MB έως 128MB. Η κλάση RecordReader φορτώνει τα δεδομένα εισόδου και τα

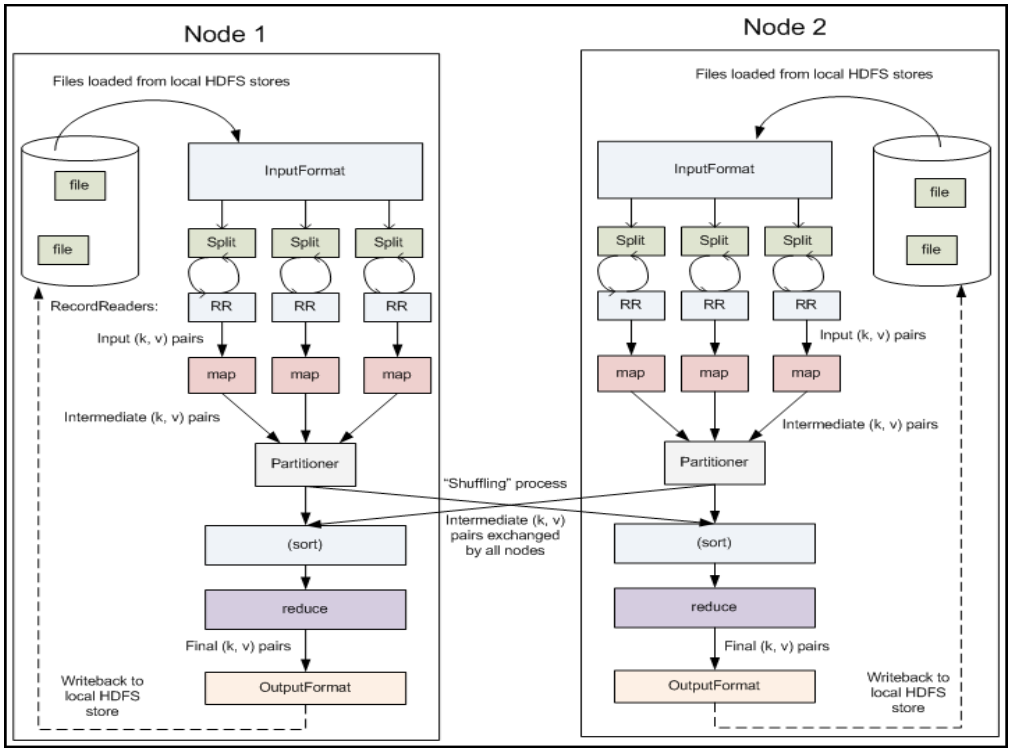
μετατρέπει σε ζευγάρια της μορφής (key,value), κατάλληλα για να διαβαστούν από τον Mapper στη συνέχεια.

- **Mapper:** Στην κλάση Mapper συναντάμε το πρώτο στάδιο του προγράμματος Map-Reduce στο οποίο οι λειτουργίες που θα εκτελεστούν ορίζονται από τον προγραμματιστή. Δεδομένου ενός ζεύγους (key/value), η μέθοδος map() δίνει στην έξοδο ένα ή περισσότερα ζεύγη (key,value), τα οποία με τη σειρά τους προωθούνται στους Reducers. Σε κάθε κλήση της map() αντιστοιχεί ξεχωριστό input split, το οποίο είναι μέρος του συνόλου των δεδομένων εισόδου και η επεξεργασία του γίνεται αποκλειστικά από το μηχάνημα-κόμβο στο οποίο εκτελείται η μέθοδος.
- **Combiner:** Ο Combiner αποτελεί ένα προαιρετικό στάδιο επεξεργασίας το οποίο εκτελείται αμέσως μετά τη φάση Map και χρησιμοποιείται για λόγους βελτιστοποίησης. Δέχεται σαν είσοδο τα δεδομένα εξόδου του Mapper και στην ουσία είναι μία διαδικασία "mini-reduce", η οποία επεξεργάζεται δεδομένα που προκύπτουν από ένα μηχάνημα.
- **Partition & Shuffle:** Αφού ολοκληρωθούν οι πρώτες map διεργασίες, τα ενδιάμεσα δεδομένα εξόδου αρχίζουν να δίνονται ως είσοδος στους reducers. Αυτή η διαδικασία ονομάζεται shuffling. Η είσοδος των reducers είναι στην ουσία "διαμερίσεις" (partitions) των ενδιάμεσων δεδομένων, καθεμία από τις οποίες αντιστοιχεί σε διαφορετικό κλειδί. Έτσι, όλες οι τιμές για ένα συγκεκριμένο κλειδί λαμβάνονται από τον ίδιο reducer. Η κλάση Partitioner καθορίζει ποια "διαμέριση" θα ακολουθήσει το κάθε ζεύγος (key/value). Μία τυπική συνάρτηση διαμέρισης κάνει χρήση μίας συνάρτησης κατακερματισμού (hash function) σε ένα κλειδί και ανάλογα με τον αριθμό των reducers, "στέλνει" τα δεδομένα για επεξεργασία στον αντίστοιχο reducer (π.χ. numOfParition = hash(key) %numOfReducers). Ο προεπιλεγμένος Partitioner του Hadoop είναι ο Hash Partitioner.
- **Sort:** Κάθε διεργασία reduce είναι υπεύθυνη για τις τιμές που σχετίζονται με τα διάφορα ενδιάμεσα κλειδιά. Το σύνολο των κλειδιών σε ένα κόμβο, ταξινομείται αυτόματα από το Hadoop πριν περάσουν ως είσοδος στον Reducer.
- **Reduce:** Είναι η φάση κατά την οποία στη μέθοδο reduce() λαμβάνεται ως είσοδος ένα κλειδί καθώς επίσης και οι τιμές οι οποίες αντιστοιχούν σε αυτό και δεν είναι απαραίτητα ταξινομημένες. Από τη φάση αυτή, παράγονται ζευγάρια (key,value), τα οποία αποτελούν την τελική έξοδο του προγράμματος.
- **RecordWriter:** Η κλάση RecordWriter χρησιμοποιείται για να εξάγει τα παραπάνω ζευγάρια ως διαφορετικές καταχωρήσεις οι οποίες τελικά αποθηκεύονται στο HDFS.

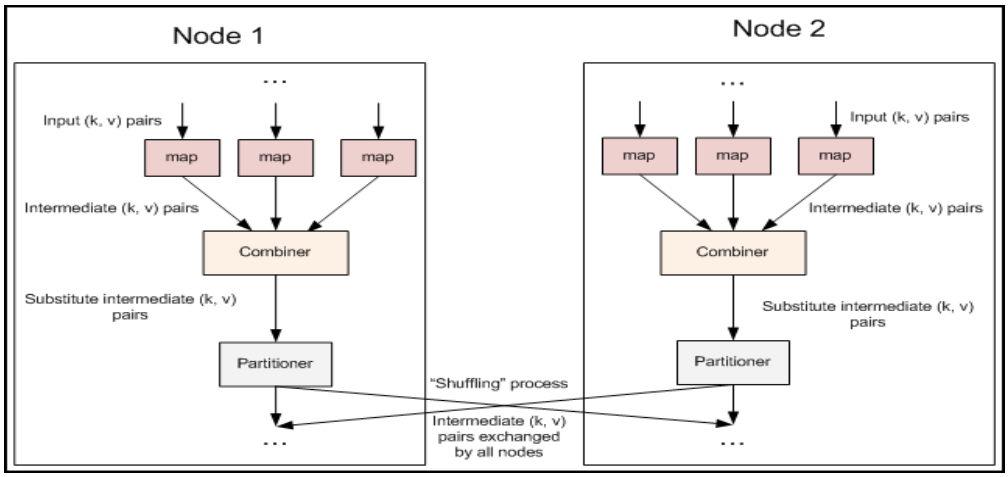
Σε αυτό το σημείο αξίζει να αναφέρουμε τη μορφή που μπορούν να έχουν τα αρχεία εισόδου και εξόδου που λαμβάνουν μέρος σε ένα πρόγραμμα Map-Reduce. Για τα αρχεία εισόδου, η μορφή τους ορίζεται από το InputFormat και οι κύριες μορφές είναι TextInputFormat, KeyValueInputFormat και SequenceFileInputFormat, ενώ για τα αρχεία εξόδου και αντίστοιχα το OutputFormat, αναφέρουμε ενδεικτικά τα TextOutputFormat, SequenceFileOutputFormat και NullOutputFormat.

Το διάγραμμα ροής δεδομένων του Map-Reduce και η χρήση των Combiners φαίνονται στις εικόνες 2.1 και 2.2 αντίστοιχα.

Τέλος, μερικοί από τους πιο συνηθισμένους τύπους δεδομένων του Hadoop είναι οι: Text, IntWritable, FloatWritable, ByteWritable, BooleanWritable.



Εικόνα 2.1: Διάγραμμα Ροής Δεδομένων στο Hadoop Map-Reduce



Εικόνα 2.2: Το προαιρετικό στάδιο Combiner

2.1.2 Hadoop Distributed File System (HDFS)

Το HDFS είναι ένα κατανεμημένο σύστημα αρχείων, το οποίο είναι σχεδιασμένο ώστε να αποθηκεύει μεγάλο όγκο πληροφορίας διαμέσου πολλών μηχανημάτων. Τα δεδομένα που βρίσκονται στο HDFS είναι προσπελάσιμα από πολλούς χρήστες μέσω ενός δικτύου. Το HDFS παρέχει ταχύτητα στην πρόσβαση των δεδομένων, ενώ είναι αξιόπιστο και ανεκτικό σε αστοχίες.

Τα αρχεία που βρίσκονται στο HDFS χωρίζονται σε blocks τυπικού μεγέθους 64MB, τα οποία αποθηκεύονται στους υπολογιστές του cluster, ενώ τα blocks του ίδιου αρχείου δε βρίσκονται απαραίτητα στον ίδιο υπολογιστή. Το κάθε ένα μηχάνημα-κόμβος του cluster ονομάζεται **DataNode**. Για ένα σύστημα αρχείων είναι σημαντικό να μπορεί να αποθηκεύει αξιόπιστα τα μεταδεδομένα του, όπως είναι τα ονόματα των αρχείων, τα δικαιώματα πρόσβασης σε αυτά καθώς επίσης και οι τοποθεσίες των blocks κάθε αρχείου. Γι' αυτό, όλα τα μεταδεδομένα του HDFS βρίσκονται σε ένα μηχάνημα-κόμβο, το **NameNode**. Λόγω του σχετικά μικρού μεγέθους της συγκεκριμένης πληροφορίας, είναι δυνατό να αποθηκεύεται στην κύρια μνήμη του NameNode, με σκοπό τη γρήγορη πρόσβαση στα μεταδεδομένα. Η πληροφορία που είναι αποθηκευμένη στον NameNode, θα πρέπει να διατηρείται ακόμα κι αν υπάρξουν σοβαρές αστοχίες στο συγκεκριμένο κόμβο· γι' αυτό το λόγο υπάρχουν παράπλευρα συστήματα τα οποία φροντίζουν για την ανάκτηση της "χαμένης" πληροφορίας.

Πώς λοιπόν επικοινωνεί ο NameNode με τους DataNodes; Για παράδειγμα, για το άνοιγμα ενός αρχείου καλείται ο NameNode ώστε να ανακτήσει μία λίστα από τοποθεσίες για τα blocks από τα οποία αποτελείται το αρχείο. Έτσι, τα δεδομένα (π.χ. σε ένα πρόγραμμα Map-Reduce) διαβάζονται άμεσα από τον τοπικό δίσκο του εκάστοτε μηχανήματος, καθώς η πληροφορία για τον DataNode στον οποίο βρίσκονται είναι πλέον διαθέσιμη.

2.2 Web Crawling

2.2.1 Web Crawling – Θεωρητικό Υπόβαθρο

Web Crawling ονομάζεται η διαδικασία κατά την οποία συλλέγονται σελίδες από το διαδίκτυο με σκοπό να αναλυθούν ώστε να συλλεχθούν περισσότερες σελίδες μέσω των συνδέσμων (links) προς εκείνες. Η συγκεκριμένη διεργασία επιτελείται από συστήματα τα οποία ονομάζονται Web Crawlers (ή Web Spiders, Web Robots).

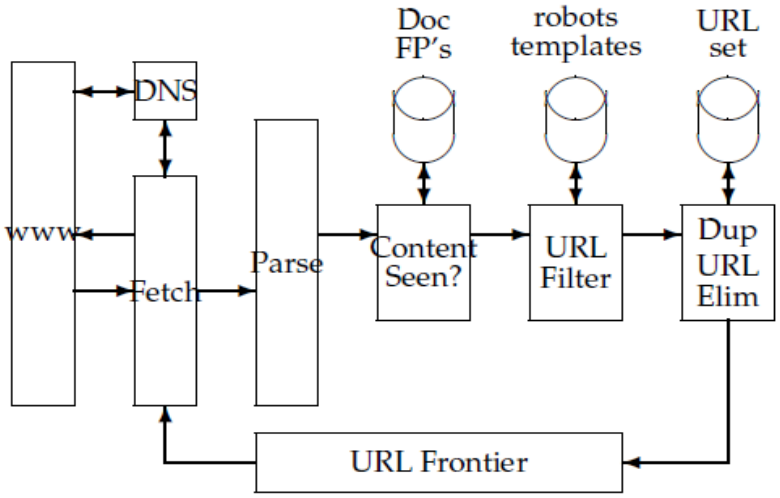
Ένας αξιόπιστος Web Crawler θα πρέπει να χαρακτηρίζεται από: 1) Ευρωστία, που σημαίνει να είναι ανθεκτικός σε παγίδες όπως οι λεγόμενες spider traps, οι οποίες είναι γεννήτριες σελίδων που οδηγούν λανθασμένα τους crawlers στην επαναληπτική συλλογή σελίδων κάτω από μία ορισμένη διεύθυνση διαδικτύου (domain) και 2) "Ευγένεια", δηλαδή ο εκάστοτε crawler θα πρέπει να λειτουργεί με βάση τις πολιτικές των διακομιστών διαδικτύου (web servers) σχετικά με τη συχνότητα των επισκέψεων του σε αυτούς. Άλλα επιθυμητά χαρακτηριστικά ενός Web Crawler είναι η ποιότητα των σελίδων που συλλέγει, η αποδοτικότητα, η επεκτασιμότητα, να έχει την ικανότητα λειτουργίας σε κατανεμημένο περιβάλλον, καθώς επίσης και η δυνατότητα να συλλέγει τα νεότερα αντίγραφα των παλιών σελίδων.

Η βασική λειτουργία που εκτελεί ένας τυπικός Web Crawler είναι η εξής: Ο crawler διαλέγει ένα url από μία αρχική λίστα με urls (seed list) με σκοπό να συλλέξει το περιεχόμενο της σελίδας με το συγκεκριμένο url. Η σελίδα εκείνη αναλύεται έτσι ώστε να εξαχθεί το λεκτικό περιεχόμενό της και

οι σύνδεσμοι από εκείνη προς άλλες σελίδες στον παγκόσμιο ιστό. Τα urls που εξήχθησαν τοποθετούνται σε ένα *URL frontier,* στο οποίο αρχίκα βρισκόταν η seed list. Έτσι, ο Web Crawler προγραμματίζει τις επισκέψεις του στις σελίδες που αντιστοιχούν στα νέα urls, ενώ μετά από κάθε επίσκεψη, το url διαγράφεται από τη λίστα. Η διαδικασία του crawling συνεχίζεται επαναληπτικά και σταματάει ή διακόπτεται ανάλογα με παραμέτρους που τίθενται στο εκάστοτε σύστημα.

Ακολουθεί η αρχιτεκτονική ενός Web Crawler (εικόνα 2.3), η οποία περιλαμβάνει διάφορα τμήματα:

1. Το URL frontier, το οποίο περιλαμβάνει τα urls που δεν έχει επισκεφθεί ακόμα ο crawler
2. Το τμήμα *DNS resolution*, που καθορίζει τον web server από τον οποίο θα αποκτηθεί η σελίδα που αντιστοιχεί σε ένα url.
3. Ένα τμήμα (fetch module) που χρησιμοποιεί το πρωτόκολλο http ώστε να ανακτήσει τη σελίδα για ένα url.
4. Ένα τμήμα ανάλυσης (parsing module) της σελίδας, το οποίο εξάγει πληροφορία για το κείμενο της καθώς επίσης και για τους συνδέσμους προς άλλες σελίδες.
5. Ένα τμήμα αφαίρεσης αντιγράφων (duplicate elimination module), το οποίο καθορίζει εάν κάποιος σύνδεσμος από το παραπάνω βήμα υπάρχει ήδη στο URL frontier ή αν έχει γίνει πρόσφατα επίσκεψη σε αυτόν.

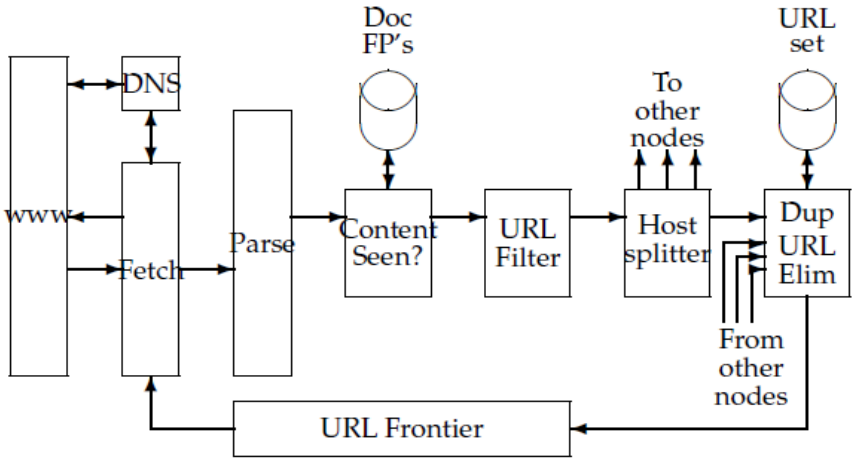


Εικόνα 2.3: Η βασική αρχιτεκτονική ενός web crawler

Θα παρακολουθήσουμε τη πορεία ενός url από τη στιγμή που θα επιλεχθεί για επεξεργασία ακολουθώντας διάφορα στάδια ελέγχων μέχρις ότου τελικά επιστραφεί στο URL frontier για μελλοντική επίσκεψη: Αρχικά, συλλέγεται η σελίδα που αντιστοιχεί στο url χρησιμοποιώντας το πρωτόκολλο http. Στη συνέχεια αναλύεται με σκοπό να εξαχθεί το κείμενο της και οι σύνδεσμοι προς άλλες σελίδες, όπως αναφέραμε και παραπάνω και τα οποία χρησιμοποιούνται για δεικτοδότηση (indexing) με σκοπό την ταξινόμηση-βαθμολόγηση (ranking) της εκάστοτε σελίδας.

Το περιεχόμενο της σελίδας ελέγχεται ώστε να εξασφαλιστεί το γεγονός ότι δεν έχει συλλεχθεί ακριβώς το ίδιο στο παρελθόν από κάποιο άλλο url (συνήθως μέσω κάποιου "αποτυπώματος"-fingerprint, "Doc FP's" στην εικόνα 2.3). Στην επόμενη φάση, χρησιμοποιείται ένα φίλτρο URL (URL filter), το οποίο βασίζεται σε διάφορους ελέγχους (π.χ. αν το συγκεκριμένο website δεν επιτρέπει τη συλλογή σελίδων, ο crawler θα πρέπει να συμμορφώνεται με αυτό) και καθορίζει αν το τρέχον url θα πρέπει να παραμείνει ή όχι στο URL frontier. Τέλος, ένα url πρέπει να ελέγχεται ώστε να αφαιρούνται τυχών διπλοτυπίες (αν ένα url υπάρχει ήδη στο URL frontier, τότε δεν ξαναπροστίθεται σε αυτό).

Η διαδικασία που περιγράφηκε μπορεί να εκτελεστεί ταυτόχρονα σε πολλά μηχανήματα, συνιστώντας έτσι την αρχή του κατανεμημένου (distributed) crawling (εικόνα 2.4). Με την παραλληλοποίηση των διεργασιών στους κόμβους ενός κατανεμημένου συστήματος, επιτυγχάνεται καλύτερη απόδοση και φυσικά μεγαλύτερη ανοχή σε σφάλματα. Η παραπάνω ιδέα μπορεί να εφαρμοστεί σε έναν *γεωγραφικά* κατανεμημένο crawler, όπου κάθε κόμβος θα είναι υπεύθυνος για εξυπηρετητές (hosts) που βρίσκονται "κοντά" σε αυτόν. Για παράδειγμα, ένας κόμβος του συστήματος που βρίσκεται στην Ευρώπη, θα επικεντρώνεται σε ευρωπαικά domains. Σε ένα κατανεμημένο σύστημα crawling, θα προσθέταμε ένα στάδιο όπου τα urls θα διαμερίζονταν με βάση π.χ. τον εξυπηρετητή στον οποίο ανήκουν, οπότε θα προχωρούσαν και στον αντίστοιχο κόμβο του cluster. Τέλος, ο έλεγχος για αντίγραφα των urls που υπάρχουν ήδη στο URL frontier γίνεται λαμβάνοντας τα αποτελέσματα από όλα τα μηχανήματα που συνιστούν τον κατανεμημένο crawler.



Εικόνα 2.3: Κατανεμημένο σύστημα web crawler

2.2.2 Apache Nutch

Στην εργασία μας, χρησιμοποιήσαμε το web crawling κομμάτι του **Apache Nutch** [21], το ανοιχτού κώδικα λογισμικό της Apache για υπηρεσίες μηχανής αναζήτησης. Το Nutch είναι γραμμένο σε Java, χρησιμοποιεί το Apache Solr [27] για τις λειτουργίες δεικτοδότησης (indexing) και αναζήτησης (searching), ενώ οι λειτουργίες του ως web crawler έχουν δημιουργηθεί από το μηδέν. Το πλεονέκτημά του είναι ότι λειτουργεί σε κατανεμημένο περιβάλλον μέσω του Hadoop και του Map-Reduce, γι' αυτό και το επιλέξαμε για την υλοποίηση της παρούσας διπλωματικής

εργασίας.

Το Nutch παρέχει λειτουργίες ώστε κάθε προγραμματιστής να έχει τη δυνατότητα να το χρησιμοποιήσει για να δημιουργήσει "τη δική του" μηχανή αναζήτησης. Για ποιο λόγο όμως θα το έκανε κάποιος αυτό από τη στιγμή που υπάρχουν μηχανές αναζήτησης όπως η Google; Υπάρχουν τουλάχιστον τρεις λόγοι:

1. **Διαφάνεια (Transparency)**: Στο Nutch, ο καθένας έχει στη διάθεσή του τους αλγόριθμους που χρησιμοποιούνται για ταξινόμηση (ranking) των αποτελεσμάτων της αναζήτησης, σε αντίθεση με τις εμπορικές μηχανές αναζήτησης όπου πλέον υπάρχει η δυνατότητα πληρωμής ώστε κάποιες σελίδες να εμφανίζονται υψηλότερα στην κατάταξη.
2. **Κατανόηση (Understanding)**: Λόγω του ότι ο πηγαίος κώδικας του Nutch είναι διαθέσιμος στο ευρύ κοινό, είναι ενδιαφέρον να εξετάσουμε πώς λειτουργεί μία μεγάλης κλίμακας μηχανή αναζήτησης. Το Nutch λειτουργεί έχοντας αντλήσει ιδέες από ακαδημαϊκό αλλά και από βιομηχανικό περιβάλλον.
3. **Επεκτασιμότητα (Extensibility)**: Το Nutch δίνει τη δυνατότητα σε κάθε προγραμματιστή να υλοποιήσει ένα σύστημα μηχανής αναζήτησης βασισμένο στις ανάγκες και στις προτιμήσεις του, καθώς μπορεί να προσαρμοστεί ώστε να λειτουργεί σε κάθε εφαρμογή.

Τα δεδομένα εξόδου από τη λειτουργία crawling του Nutch είναι:

- Η **βάση δεδομένων crawl** ή **crawldb**, η οποία περιέχει πληροφορία για κάθε url που λαμβάνει μέρος στη διαδικασία, όπως είναι το αν συλλέχθηκε το περιεχόμενο της αντίστοιχης σελίδας και το πότε έγινε αυτό, καθώς και άλλα μεταδεδομένα.
- Η **βάση δεδομένων link** ή **linkdb**, η οποία περιέχει τη λίστα όλων των γνωστών συνδέσμων (links) προς κάθε url, συμπεριλαμβανομένου του url προέλευσης (source url) και του αντίστοιχου anchor text (κείμενο του συνδέσμου).
- Ένα σύνολο από **τμήματα (segments)**, καθένα από τα οποία περιλαμβάνει πληροφορία για ένα σύνολο από urls που συλλέχθηκαν στην ίδια επανάληψη. Τα segments είναι directories που αποτελούνται από τα ακόλουθα subdirectories:
    - *crawl_generate:* ένα σύνολο από urls που πρόκειται να συλλεχθούν
    - *crawl_fetch:* περιέχει την κατάσταση κάθε url που συλλέγεται
    - *content:* περιέχει το ακατέργαστο περιεχόμενο που ανακτάται από κάθε url
    - *parse_text:* περιέχει το κείμενο που προέκυψε από την ανάλυση κάθε url
    - *parse_data:* περιέχει τους εξω-συνδέσμους (outlinks) ή λίστα γειτνίασης (adjacency list) και μεταδεδομένα που εξάγονται από κάθε url
    - *crawl_parse:* περιέχει τους εξω-συνδέσμους των urls που χρησιμοποιούνται για να ενημερώσουν την crawldb

Παρουσιάζονται τα βήματα που ακολουθεί το Nutch ώστε να εκτελεστεί η διαδικασία του crawling (εικόνα 2.4):

1. Αρχικοποίηση της crawldb με μία λίστα από root urls

   εντολή: *bin/nutch **inject** <basedir>/crawldb urls*

2. Παραγωγή μίας λίστας (fetchlist) urls από τη crawldb. Επιλέγονται προς συλλογή τα urls με το μεγαλύτερο score, δημιουργούνται segments, ενώ οι διαμερίσεις των urls δημιουργούνται με βάση τους εξυπηρετητές (hosts)

   εντολή: *bin/nutch **generate** <basedir>/crawldb <basedir>/segments -topN <numdoc>*

3. Γίνεται η συλλογή του περιεχομένου των σελίδων από τα urls στην fetchlist

   εντολή: *bin/nutch **fetch** <basedir>/segments/ -threads <threads>*

4. Γίνεται ανάλυση των δεδομένων

   εντολή: *bin/nutch **parse** <basedir>/segments/*

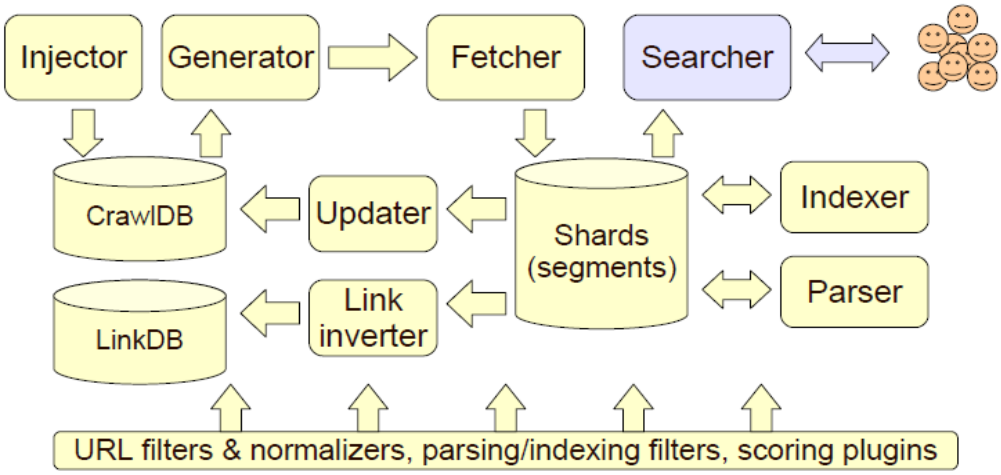
5. Γίνεται ενημέρωση της crawldb. Τα δεδομένα του τρέχοντος segment ενσωματώνονται στη βάση δεδομένων

   εντολή: *bin/nutch **updatedb** <basedir>/crawldb <basedir>/segments/ -filter*

6. Όλοι οι σύνδεσμοι "αντιστρέφονται" (εντολή invertlinks) και δημιουργείται η βάση δεδομένων linkdb, ώστε να είναι δυνατή η εξαγωγή του κειμένου των συνδέσμων (anchor text) που συνοδεύει τις σελίδες

   εντολή: *bin/nutch **invertlinks** <basedir>/linkdb <basedir>/segments/**

Τα βήματα 2 έως 5 εκτελούνται επαναληπτικά, με τον αριθμό των επαναλήψεων να απεικονίζει το βάθος στο οποίο θα "τρέξει" ο crawler. Οι λειτουργίες των indexing και searching έχουν πλέον ανατεθεί στο **Apache Solr**, το οποίο χρησιμοποιεί και επεκτείνει τη βιβλιοθήκη **Apache Lucene** [22].



Εικόνα 2.4: Η αρχιτεκτονική του Nutch

Εκτός από το Nutch, έχουν αναπτυχθεί και άλλα ανοιχτού κώδικα (open source) συστήματα web

crawling που χρησιμοποιούν το Hadoop, όπως τα Bixo[28] και NutchWax[29]. Το Bixo είναι ένα εργαλείο για εξόρυξη στο διαδίκτυο (web mining) βασισμένο στο Hadoop, ενώ το NutchWax περιλαμβάνει το βήμα *fetch* του Nutch και χρησιμοποιείται για τη συλλογή αρχείων (web archives) στο διαδίκτυο.

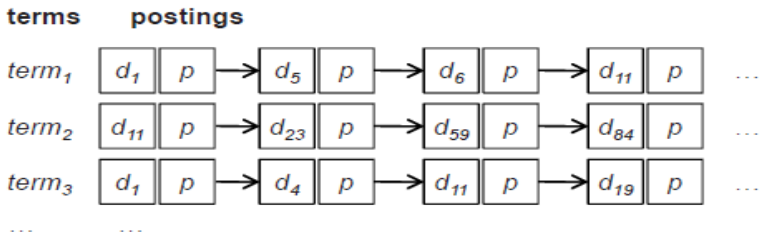
2.3 Indexing και Vector Space Model

Το επόμενο στάδιο μετά το crawling για μία μηχανή αναζήτησης είναι η διαδικασία του **Indexing**. Στην παρούσα ενότητα θα παρακολουθήσουμε τον τρόπο με τον οποίο γίνεται η κατασκευή του Inverted Index σε κατανεμημένο περιβάλλον (Hadoop Map-Reduce). Επίσης, γίνεται η θεωρητική ανασκόπηση του **Vector Space Model (VSM)**, το οποίο χρησιμοποιείται ευρέως στην ανάκτηση πληροφορίας στο διαδίκτυο.

2.3.1 Κατασκευή του Inverted Index

Ο Inverted Index σε μία μηχανή αναζήτησης αποτελεί τη δομή η οποία αποθηκεύει για κάθε όρο (term) της συλλογής των κειμένων-σελίδων (documents), μία λίστα των κειμένων στα οποία βρίσκεται, καθώς επίσης και πρόσθετη πληροφορία, όπως διάφορα scores, αριθμό εμφανίσεων του όρου κ.α.. Το πιο συνηθισμένο score είναι η συχνότητα εμφάνισης *tf* (term frequency) κάποιου όρου στη συλλογή των κειμένων, ενώ μπορεί επίσης να αποθηκευτεί πληροφορία για τη θέση του όρου μέσα στο εκάστοτε κείμενο (positional information). Τα κείμενα απεικονίζονται στον Inverted Index με ένα μοναδικό id για το καθένα. Στην εικόνα 2.5 απεικονίζεται ένας τυπικός Inverted Index. Ο σκοπός της συγκεκριμένης δομής είναι η γρήγορη ανάκτηση των κειμένων που αφορούν τον εκάστοτε όρο του λεξικού (dictionary). Τα κύρια βήματα για την κατασκευή του Inverted Index είναι τα εξής:

1. Η συλλογή των κειμένων που θα περάσουν στη φάση του Indexing
2. Ο διαχωρισμός των όρων του κάθε κειμένου (text tokenization)
3. Η γλωσσική προεπεξεργασία των όρων, όπως είναι η αφαίρεση των stopwords (λέξεις που χρησιμοποιούνται συχνά, αλλά δεν επηρεάζουν το νόημα του κειμένου όπως π.χ. a,the,is,it κ.α.), το Stemming (π.χ. οι όροι “stopping”, ”stopped”, “stop” μετατρέπονται όλοι σε έναν όρο “stop”) και η μετατροπή των κεφαλαίων σε πεζά.
4. Δημιουργία των posting lists, που περιέχουν τα ids των κειμένων στα οποία εμφανίζεται κάθε όρος της συλλογής.

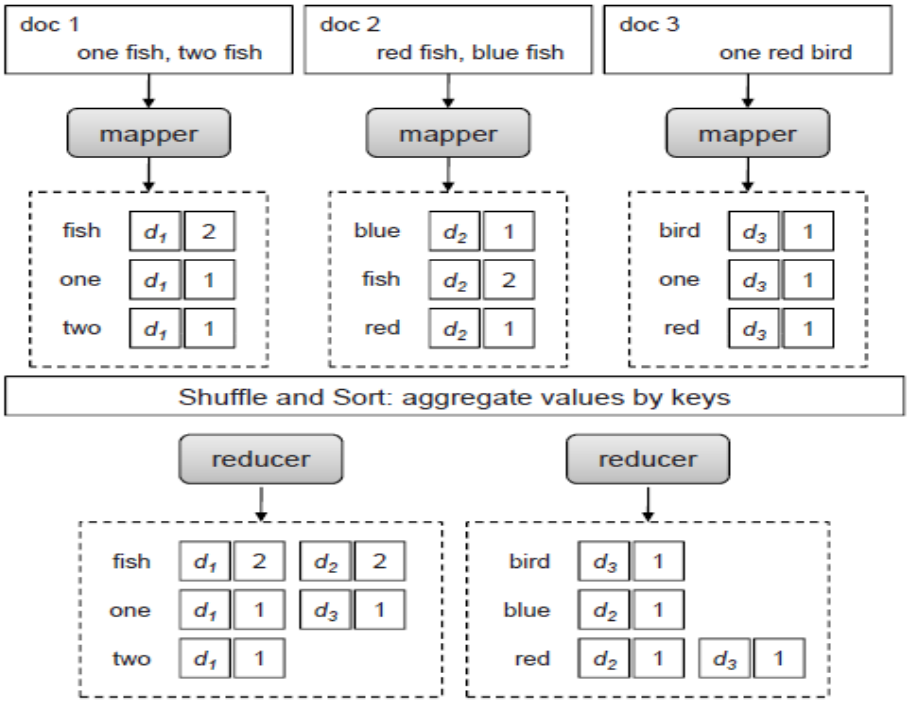


Εικόνα 2.5: Απλή απεικόνιση ενός Inverted Index. Κάθε όρος σχετίζεται με μία posting list, η οποία περιλαμβάνει τα ids των documents και ένα βάρος p για κάθε ζεύγος <term,document>.

Στο παράδειγμα της εικόνας 2.5, παρατηρούμε ότι ο όρος term1 βρίσκεται στα {d1, d5, d6,

d11, . . .}, ο όρος term2 στα {d11, d23, d59, d84, . . .} και ο όρος term3 στα {d1, d4, d11, d19, . . .}. Σε κάθε document ανατίθεται ένας μοναδικός ακέραιος από 1 έως n, όπου n το πλήθος των documents της συλλογής μας. Γενικά, οι posting lists είναι ταξινομημένες με βάση το score που έχει επιλεγεί, με βάση το document id ή οποιοδήποτε άλλο κανόνα.

Οι συλλογές των documents συνήθως έχουν απαγορευτικά μεγάλο μέγεθος ώστε ο index να κατασκευαστεί αποδοτικά σε ένα μηχάνημα. Γι' αυτό το λόγο, οι μηχανές αναζήτησης χρησιμοποιούν clusters υπολογιστών, οπότε και η διαδικασία του indexing γίνεται σε κατανεμημένο περιβάλλον. Έτσι, ο Inverted Index διαμερίζεται στα διάφορα μηχανήματα είτε με βάση τον όρο (term partitioning), είτε με βάση το document (document partitioning) με σκοπό την όσο το δυνατόν πιο αποδοτική κατανομή του μεγέθους του σε κάθε κόμβο του cluster. Η κατασκευή του Index σε κατανεμημένο περιβάλλον μπορεί να επιτευχθεί με τη χρήση του μοντέλου Map-Reduce, όπου οι λειτουργίες που περιγράψαμε παραπάνω εκτελούνται παράλληλα σε πολλά μηχανήματα. Αξίζει να σημειώσουμε ότι η μη χρήση του Map-Reduce, θα απαιτούσε από τον προγραμματιστή να ομαδοποιήσει τα δεδομένα με βάση τον όρο, δεδομένων των περιορισμών σε μνήμη και δίσκο. Αντίθετα, το Map-Reduce παρέχει αρκετές διευκολύνσεις, όπως είναι η ταξινόμηση και η ομαδοποίηση των δεδομένων.



Εικόνα 2.6: Απλή απεικόνιση του αλγορίθμου κατασκευής του Inverted Index σε Map-Reduce, χρησιμοποιώντας 3 mappers και 2 reducers. Οι posting lists αποτελούνται από ζεύγη της μορφής <docid,tf>.

Στην εικόνα 2.6 απεικονίζεται ένα απλό παράδειγμα κατασκευής Inverted Index σε Map-Reduce, σύμφωνα με τον αλγόριθμο που προτάθηκε από τους Dean και Ghemawat[1], τον οποίο και ακολουθήσαμε στη φάση Indexing της παρούσας εργασίας. Κάθε document λαμβάνεται ως είσοδος από ξεχωριστό Mapper, ο οποίος παράγει ζεύγη της μορφής <key,value> με key έναν όρο

και value τον αριθμό των εμφανίσεων του όρου στο συγκεκριμένο document. Στη συνέχεια, τα ενδιάμεσα δεδομένα διαμερίζονται με βάση τον όρο και δίνονται ως είσοδος στους reducers. Τελικά, παρατηρούμε ότι η έξοδος των reducers περιλαμβάνει τον κατανεμημένο Inverted Index με key τον όρο και value την αντίστοιχη posting list, η οποία αποτελείται από ζεύγη της μορφής <docid,"συχνότητα εμφάνισης">.

2.3.2 Vector Space Model

Το Vector Space Model (VSM) είναι ένα αλγεβρικό μοντέλο που χρησιμοποιείται στην περιοχή της ανάκτησης πληροφορίας και μας βοηθάει στον υπολογισμό ενός score για τη σχέση ανάμεσα σε ένα document και ένα δεδομένο query. Τα documents και τα queries αναπαρίστανται ως διανύσματα (vectors) όπου κάθε διάσταση αντιστοιχεί σε ένα όρο του λεξικού και απεικονίζει ένα βάρος w, το οποίο είναι μη μηδενικό εάν ο όρος εμφανίζεται μέσα στο document ή στο query αντίστοιχα τουλάχιστον μία φορά:

$$d_j = (w_{1,j}, w_{2,j}, \ldots, w_{t,j}) \;,\; q = (w_{1,q}, w_{2,q}, \ldots, w_{t,q})$$

Τα πιο απλό βάρος που θα μπορούσε να χρησιμοποιηθεί είναι για παράδειγμα η συχνότητα των εμφανίσεων του όρου μέσα στο κείμενο (term frequency ή tf). Το συγκεκριμένο όμως δημιουργεί ένα σημαντικό πρόβλημα: κάποιοι όροι που εμφανίζονται με πολύ μεγάλη συχνότητα μέσα στα κείμενα της συλλογής μας, θα λαμβάνουν και μεγαλύτερο βάρος ανεξάρτητα από το αν είναι σημαντικοί ή όχι, δεδομένου του query. Γι' αυτό το λόγο, υπάρχει η ανάγκη για ένα πιο σύνθετο score, το οποίο θα λαμβάνει υπόψιν τον αριθμό των κειμένων στα οποία εμφανίζεται ένας όρος τουλάχιστον μία φορά και ορίζεται ως document frequency (df). Ένα από τα πιο συνηθισμένα βάρη είναι το **tf-idf** score, στον υπολογισμό του οποίου χρησιμοποιείται και η παράμετρος idf (inverse document frequency).

Συνοψίζουμε τις παραπάνω παραμέτρους:

- **term frequency ($tf_{t,d}$)**: Η συχνότητα των εμφανίσεων του όρου t στο κείμενο d.
- **document frequency (df_t)**: Το πλήθος των κειμένων στα οποία ο όρος t εμφανίζεται τουλάχιστον μία φορά
- **inverse document frequency (idf_t)**: Το συγκεκριμένο βάρος δίνει ένα μέτρο για το πόσο χαρακτηριστικός είναι ένας όρος t της συλλογής μας. Ορίζεται ως: $\text{idf}_t = \log\left(\frac{N}{\text{df}_t + 1}\right)$

όπου N το συνολικό πλήθος των κειμένων. Όσο πιο σπάνια εμφανίζεται ο όρος στα κείμενα, τόσο πιο υψηλό είναι το idf του όρου. Ο λογάριθμος χρησιμοποιείται για να μετριάσει την επίδραση του idf στο συνολικό score, ενώ η βάση του λογαρίθμου έχει αποδειχτεί ότι δεν επηρεάζει στην ουσία τα τελικά αποτελέσματα[6]. Στον παρονομαστή, προστίθεται η τιμή 1, λαμβάνοντας έτσι υπόψιν την περίπτωση μηδενικής τιμής του df_t.

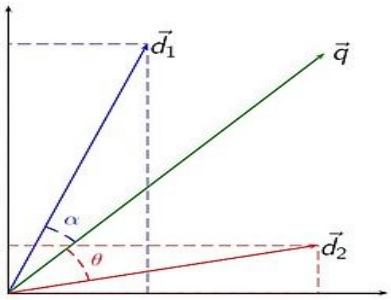
Συνδυάζοντας τα βάρη tf και idf, παράγεται το score tf-idf, το οποίο δίνει ένα μέτρο βάρους για έναν όρο t μέσα σε ένα κείμενο d: $\textbf{tf-idf}_{t,d} = \textbf{tf}_{t,d} * \textbf{idf}_{dt}$

Όπως αναφέραμε και πριν, κάθε κείμενο μπορεί να αναπαρασταθεί ως διάνυσμα, κάθε στοιχείο του οποίου αποτελείται από ένα score σχετικό με έναν όρο για εκείνο το κείμενο. Στο VSM, κάθε query αναπαρίσταται επίσης ως ένα τέτοιο διάνυσμα, γεγονός που μας δίνει τη δυνατότητα να

υπολογίσουμε ένα μέτρο σχετικότητας ανάμεσα σε ένα query και ένα κείμενο. Έτσι, χρησιμοποιείται η **ομοιότητα** του **συνημιτόνου (cosine similarity)** της γωνίας των δύο διανυσμάτων (εικόνα 2.7), η οποία ορίζεται ως εξής:

$$\text{sim}(q,d) = \cos(\vec{q}, \vec{d}) = \frac{\sum_{i=1}^{N} q_i d_i}{\sqrt{\sum_{i=1}^{N} q_i^2}\sqrt{\sum_{i=1}^{N} d_i^2}}$$

, όπου $\mathbf{q_i}$ το βάρος του όρου i στο query,
$\mathbf{d_i}$ το βάρος του όρου i στο document και
N το συνολικό πλήθος των κειμένων



Εικόνα 2.7: Vector Space Model. Τα συνημίτονα των γωνιών α και θ εκφράζουν την ομοιότητα μεταξύ των ζευγών διανυσμάτων (q,d1) και (q,d2) αντίστοιχα.

Αξίζει να σημειωθεί πως δεν υπάρχει κάποιος περιορισμός για το score το οποίο θα χρησιμοποιηθεί σε κάθε διάνυσμα. Στην υλοποίηση μας, όπως θα δούμε και στην ενότητα 3.7, για τα διανύσματα των κειμένων λάβαμε υπόψιν το tf-idf score, ενώ για τα διανύσματα των queries γίνεται χρήση του idf score, καθώς υπάρχει η δυνατότητα να χρησιμοποιηθεί το εκείνο αντί του tf-idf score, όπως παρατηρούμε στο [6] (Κεφ. 6).

Σε μεγάλες συλλογές, εφαρμόζεται συνήθως κάποιου είδους κανονικοποίηση στην term frequency, όπως είναι η μέθοδος κανονικοποίησης με βάση το μήκος του εκάστοτε κειμένου. Έτσι λοιπόν, για κάθε όρο ενός συγκεκριμένου κειμένου προκύπτει:

$$\text{tf}_{i,d} = \frac{tc_{i,d}}{\sqrt{\sum_{k=1}^{N} tc_{k,d}^2}}$$

όπου $\mathbf{tc_{id}}$ το πλήθος των εμφανίσεων του όρου i στο κείμενο d (term counts)

Η συγκεκριμένη μέθοδος χρησιμοποιήθηκε στην εργασίας μας, ωστόσο υπάρχουν και άλλες παραλλαγές κανονικοποίησης των βαρών (π.χ. Pivoted Normalized Document Length, Maximum tf Normalization).

Το μοντέλο VSM λειτουργεί αρκετά καλά σε μεγάλες συλλογές κειμένων και παρέχει βαθμονομημένα αποτελέσματα, σε αντίθεση π.χ. με το boolean model, το οποίο εξετάζει απλώς αν κάποιος όρος περιέχεται μέσα σε κάποιο κείμενο. Ως εκ τούτου, το VSM υπερέχει έναντι του

boolean model όταν οι όροι του επερωτήματος του χρήστη αυξάνονται.

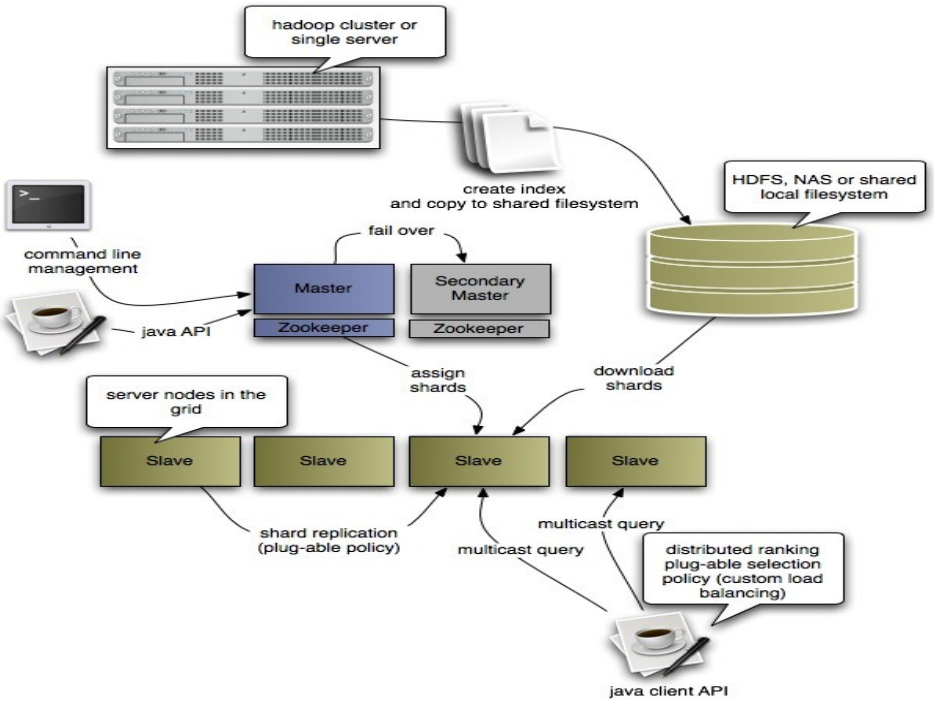
Ένα από τα βασικά μειονεκτήματα του VSM είναι ότι δεν απαιτείται όλοι οι όροι ενός επερωτήματος να βρίσκονται μέσα σε ένα κείμενο. Για παράδειγμα, για ένα επερώτημα "A B", μπορεί να προτιμηθεί ένα κείμενο που περιέχει με μεγάλη συχνότητα το A, αλλά καθόλου το B, από ένα κείμενο το οποίο περιέχει και τα δύο αλλά με μικρότερη συχνότητα. Τέλος, το κύριο μειονέκτημα του VSM είναι η υψηλή πολυπλοκότητα του αλγορίθμου, σε αντίθεση με το boolean model, το οποίο είναι εξαιρετικά απλό και γρήγορο.

2.3.3 Συναφή Συστήματα

Οι μέθοδοι που περιγράφηκαν στις ενότητες 2.3.1 και 2.3.2 σχετικά με την κατασκευή του Inverted Index σε κατανεμημένο περιβάλλον και με την ανάθεση βαρών και εξαγωγή αποτελεσμάτων μέσω του Vector Space Model, έχουν εφαρμογή στο Hadoop και αυτό είναι ένα θέμα που μας ενδιαφέρει να εξεταστεί. Έχουν γίνει προσπάθειες για την ανάπτυξη συστημάτων που υποστηρίζουν λειτουργίες δεικτοδότησης (indexing) και βαθμονόμησης (ranking) των αποτελεσμάτων στο Hadoop και αυτές θα παρουσιάσουμε παρακάτω.

Όπως αναφέραμε στην ενότητα 2.2.1, το **Nutch** υποστηρίζει indexing με βάση το μοτίβο Map-Reduce, χρησιμοποιώντας τη βιβλιοθήκη Apache Lucene. Στο Nutch, αντίθετα με την τεχνική που προτάθηκε στο [1], στη φάση Map το μόνο που γίνεται είναι ο διαχωρισμός σε λέξεις του κειμένου εισόδου και στην έξοδο της συνάρτησης παράγονται ζεύγη <docID,analysedDocument>, όπου το analysedDocument περιέχει το είδος κάθε όρου μέσα στο κείμενο (π.χ. τίτλος σελίδας, anchor text κ.α.), καθώς και την αντίστοιχη συχνότητα εμφάνισης. Στη φάση Reduce, παράγονται στην έξοδο indexes που προκύπτουν από το Lucene και αποθηκεύονται με τη σειρά τους στο HDFS. Το πλεονέκτημα της παρούσας τεχνικής είναι ότι συνολικά τα ενδιάμεσα δεδομένα από τη φάση Map στη φάση Reduce είναι λιγότερα, όμως μεμονωμένα κάθε ζεύγος <docID, analysedDocument> αποτελείται από σαφώς περισσότερα δεδομένα με βάση τα περιεχόμενα της τιμής analysedDocument. Σε κάθε περίπτωση, είναι ένα βήμα για τη μείωση των δεδομένων εξόδου της φάσης Map κατά το indexing. Η αναζήτηση στο Nutch, που πλέον έχει ανατεθεί στο Solr, γίνεται χρησιμοποιώντας το Lucene, δίχως την παρουσία του μοτίβου Map-Reduce.

Μία άλλη εφαρμογή η οποία έχει τη δυνατότητα να λειτουργήσει κατανεμημένα σε ένα Hadoop cluster, αποθηκεύοντας Lucene indexes στο HDFS, είναι το **Katta** (εικόνα 2.8) [26]. Το Katta έχει υλοποιηθεί έτσι ώστε να είναι ανεκτικό σε αστοχίες, ενώ ταυτόχρονα είναι δεκτικό σε πιθανές επεκτάσεις. Ένας κύριος διακομιστής (master server) διαχειρίζεται τους κόμβους του δικτύου και την εκχώρηση των τμημάτων (shards) του εκάστοτε index σε αυτούς. Έτσι, ένας πελάτης (client) έχει τη δυνατότητα να εκτελέσει μία αναζήτηση σε έναν index, αφού συνδεθεί σε όλους τους κόμβους και συγχωνεύσει τα αποτελέσματα από εκείνους σε μία ενοποιημένη λίστα αποτελεσμάτων. Το είδος του κάθε index μπορεί να είναι διαφορετικό, όπως π.χ. Lucene index ή Hadoop mapfile και εξαρτάται από τους διακομιστές περιεχομένου (content servers) που εξυπηρετούνται από τους κόμβους. Ένας index στο Katta δεν είναι τίποτα άλλο από έναν φάκελο που αποτελείται από ένα σύνολο υποφακέλων, οι οποίοι ονομάζονται *index shards*. Εκείνοι αποτελούνται από Lucene indexes ή Hadoop mapfiles (ή οποιοδήποτε άλλο format αποφασίσει να υλοποιήσει ο προγραμματιστής από μόνος του), κάτι αρκετά απλό καθώς στην ουσία ένας Katta index δημιουργείται, για παράδειγμα, από πολλαπλούς Lucene indexes που αντιγράφονται σε ένα φάκελο.



Εικόνα 2.8: Η αρχιτεκτονική του Katta

Επίσης, υπάρχει η δυνατότητα ένας Katta index να δημιουργηθεί από τον προγραμματιστή μέσω του Hadoop Map-Reduce, κάτι που επιτρέπει την δομημένη μορφή του τελικού index με τρόπο που ταιριάζει στην εκάστοτε εφαρμογή (π.χ. κείμενα με πολλούς ίδιους όρους μεταξύ τους να βρίσκονται στο ίδιο shard).

Στα κατανεμημένα συστήματα, η επικοινωνία master-node είναι σημαντική καθώς ο master πρέπει να ενημερώνεται άμεσα για οποιαδήποτε αστοχία σε κάποιο κόμβο. Το Katta, λοιπόν, χρησιμοποιεί για αυτό το σκοπό το σύστημα Zookeeper, το οποίο επιτρέπει την ανάγνωση και την εγγραφή σε ένα είδος ιδεατού κατανεμημένου συστήματος αρχείων. Με αυτόν τον τρόπο, ο master ανταλλάσσει "μηνύματα" με τους κόμβους, ώστε να γίνεται ενημέρωση για πιθανές αλλαγές και φυσικά για να αντιμετωπίζονται οποιαδήποτε προβλήματα προκύψουν στις λειτουργίες του συστήματος. Όπως είδαμε και πιο πάνω, ένας πελάτης επικοινωνεί με τους κόμβους του δικτύου ώστε να εκτελέσει μία αναζήτηση. Αυτό γίνεται εφικτό με τη χρήση του συστήματος Hadoop RPC από το Katta.

Το Katta, τέλος, υποστηρίζει λειτουργία για εξαγωγή των scores με κατανεμημένο τρόπο. Κάθε επερώτημα για το οποίο εκτελείται αναζήτηση, περνάει από δύο στάδια. Αρχικά, λαμβάνονται οι τιμές df για το επερώτημα από όλους τους κόμβους, ενώ στη συνέχεια οι τιμές εκείνες προωθούνται μαζί με το επερώτημα σε όλους τους κόμβους.

Ένα ακόμη ανοικτού κώδικα σύστημα που παρέχει πολλαπλές λειτουργίες indexing και ανάκτησης πληροφορίας είναι το **Terrier** [25]. Η δυνατότητα του Terrier για αποδοτική λειτουργία indexing σε μεγάλου όγκου συλλογές από τον παγκόσμιο ιστό με βάση το μοντέλο Hadoop Map-Reduce είναι αυτό που μας προτρέπει να το εξετάσουμε. Το Terrier κάνει χρήση της τεχνικής "**Single-pass Map-Reduce Indexing Strategy**", η οποία προτάθηκε από τους McCreadie,

Macdonald και Ounis [7] και αποτελεί μία εναλλακτική μέθοδο σχετικά με το Indexing με το Map-Reduce. Το χαρακτηριστικό της τεχνικής αυτής είναι ότι στη φάση Map, η οποία λαμβάνει ως είσοδο ένα υποσύνολο των κειμένων της συλλογής, οι posting lists για κάθε όρο δημιουργούνται και αποθηκεύονται προσωρινά στην κύρια μνήμη. Όταν η μνήμη εξαντληθεί ή η συγκεκριμένη διεργασία map τελειώσει την επεξεργασία, τα ενδιάμεσα δεδομένα με μορφή <term, postingList> αποθηκεύονται στο δίσκο. Στην ουσία λοιπόν, τα δεδομένα που δίνονται ως είσοδος για ένα όρο στη φάση Reduce, αποτελούνται από πολλά διαφορετικά τμήματα του ίδιου index για εκείνο τον όρο. Στη συνάρτηση reduce, οι posting lists για κάθε όρο συγχωνεύονται σε μία τελική λίστα, οπότε και τα δεδομένα αυτά είναι η έξοδος του προγράμματος.

Το πλεονέκτημα της μεθόδου που χρησιμοποιείται στο Terrier έναντι της τεχνικής των Dean και Ghemawat είναι ότι τα ενδιάμεσα δεδομένα από τη φάση Map στη φάση Reduce παράγονται πολύ λιγότερες φορές, όμως είναι μεγαλύτερου μεγέθους. Σε σχέση με την στρατηγική για Indexing που ακολουθείται στο Nutch, η τεχνική Single-pass Map-Reduce παράγει περισσότερες φορές ενδιάμεσα δεδομένα, όμως η κάθε διεργασία reduce επεξεργάζεται ταξινομημένα (κατά όρο) δεδομένα και δεν απαιτεί περαιτέρω ταξινόμηση και λειτουργία αναστροφής για την εξαγωγή του Inverted Index.

Για την κατασκευή του Inverted Index στην εργασία μας, στηριχθήκαμε αρχικά στο [1] όμως στη συνέχεια προέκυψε η ανάγκη να κρατήσουμε περισσότερη πληροφορία όπως είναι το βάρος idf για κάθε όρο, καθώς και τα tf-idf scores για κάθε ζευγάρι όρου-κειμένου. Σύμφωνα με τα παραπάνω, η τεχνική που ακολουθήσαμε παράγει περισσότερες φορές ενδιάμεσα δεδομένα μεταξύ των φάσεων Map και Reduce σε σχέση με το Nutch. Από την άλλη πλευρά όμως, τα δεδομένα για κάθε ενδιάμεση έξοδο της Map στη φάση Indexing του Nutch έχουν σαφώς μεγαλύτερο όγκο, κάτι που είναι πιθανό να εξαντλήσει τα περιθώρια της κύριας μνήμης του συστήματος όταν το μέγεθος των ενδιάμεσων δεδομένων αυξηθεί σημαντικά. Ένα άλλο πλεονέκτημα της υλοποίησής μας είναι ότι δεν περιοριζόμαστε στη χρήση των Lucene indexes, αλλά έχουμε τη δυνατότητα να υλοποιήσουμε τον index με όποιο τρόπο ταιριάζει καλύτερα στην εφαρμογή μας.

Όσον αφορά το Katta, το οποίο χρησιμοποιείται κυρίως ως μία κατανεμημένη "αποθήκη" για indexes και παρέχει υπηρεσίες αναζήτησης στα διάφορα επερωτήματα, υπολογίζει και εκείνο τα tf-idf scores, με τη χρήση του Lucene όμως, με σκοπό να εξάγει αξιόπιστα αποτελέσματα με κατανεμημένο τρόπο. Σε αντίθεση με το Katta, στην υλοποίηση μας τα scores προκύπτουν κάνοντας χρήση του Map-Reduce, κάτι που συμβαίνει και κατά την αναζήτηση δίνοντας ταυτόχρονα στο σύστημα ένα σύνολο από επερωτήματα. Το γεγονός αυτό μας δίνει τη δυνατότητα να εκμεταλλευτούμε τα οφέλη του Map-Reduce εξυπηρετώντας ταυτόχρονα μεγάλο όγκο επερωτημάτων.

Συνοψίζοντας, η υλοποίηση του συστήματος μηχανής αναζήτησης που επιχειρούμε, διαφέρει με τα παραπάνω συστήματα Indexing, στα εξής κύρια σημεία:

- Η διαδικασία Indexing που γίνεται στο Nutch, παράγει μεγάλο όγκο ενδιάμεσων δεδομένων από τη φάση Map στη Reduce, καθώς αποτελείται από ζεύγη της μορφής <docID, analysedDocument>. Αντίθετα, στο σύστημά μας, κατά τη φάση Map παράγονται ζεύγη της μορφής <term, (docId, termFrequency)>, τα οποία, αν και είναι περισσότερα, καταλαμβάνουν συνολικά μικρότερο όγκο.
- Σε σύγκριση με τη δημιουργία του Index από το σύστημα Katta, εμείς επιλέξαμε μία

απλούστερη λειτουργικότητα, δίχως τη χρήση Lucene Indexes, έτσι ώστε να έχουμε μεγαλύτερο έλεγχο των δεδομένων και τη δυνατότητα να διαμορφώσουμε τον Index με βάση τις απαιτήσεις της εργασίας. Κατασκευάζουμε λοιπόν τον Inverted Index από το μηδέν με σκοπό την εξέταση της απόδοσής του στο Hadoop.

- Σε σχέση με τη μέθοδο που χρησιμοποιείται στο Terrier, η δική μας υλοποίηση για τη δημιουργία του Inverted Index, δεν κάνει χρήση της κύριας μνήμης του συστήματος για τα ενδιάμεσα δεδομένα και ως εκ τούτου δεν την επιβαρύνει παραπάνω.

2.4 Personalized PageRank

Εκτός από την ανάλυση του λεκτικού περιεχομένου των σελίδων μίας συλλογής από τον παγκόσμιο ιστό για την εξαγωγή βαθμονομημένων αποτελεσμάτων, η **ανάλυση των συνδέσμων** μεταξύ των σελίδων είναι εξίσου σημαντική για την ταξινόμηση που θα προκύψει από μία μηχανή αναζήτησης. Με βάση αυτή τη λογική, στηριζόμαστε στον κατευθυνόμενο γράφο (directed graph) ο οποίος απεικονίζει τις σελίδες ως κόμβους και τους συνδέσμους μεταξύ τους ως ακμές. Για παράδειγμα, μία σελίδα Α που περιέχει συνδέσμους για τις σελίδες Β και Γ, θα αναπαρίσταται από ένα κόμβο με δύο εξερχόμενες ακμές προς τους κόμβους που αντιστοιχούν στις Β και Γ.

Η τεχνική που θα εξετάσουμε είναι ο αλγόριθμος PageRank, ο οποίος αναπτύχθηκε από τους Lawrence Page και Sergey Brin [11][12] στο Stanford University το 1998 και χρησιμοποιείται από την Google. Ο αλγόριθμος PageRank λαμβάνει υπόψιν τον τρόπο που είναι συνδεδεμένες οι σελίδες στον παγκόσμιο ιστό, δίχως να ασχολείται με το λεκτικό περιεχόμενό τους. Ο αλγόριθμος βασίζεται στη λογική ότι κάθε σελίδα κατέχει ένα μέτρο "κύρους" το οποίο είναι ανάλογο με το άθροισμα των μέτρων "κύρους" των σελίδων που περιέχουν συνδέσμους προς εκείνη. Συνεπώς, όσο πιο σημαντικές είναι εκείνες οι σελίδες, τόσο πιο υψηλή θα είναι η τιμή του PageRank για τη σελίδα στην οποία "δείχνουν". Η τιμή του PageRank για τις διάφορες σελίδες, αποτελεί μία από τις πολλές παραμέτρους (π.χ. Cosine Similarity) που υπολογίζονται ώστε να εξαχθεί ένα σύνθετο τελικό score που θα ανταποκρίνεται στα επερωτήματα των χρηστών του διαδικτύου.

Παρακάτω θα εξηγήσουμε πώς ο αλγόριθμος PageRank μπορεί να τροποποιηθεί ώστε να λαμβάνει υπόψιν τις προτιμήσεις του εκάστοτε χρήστη. Για παράδειγμα, εάν κάποιος ενδιαφέρεται για τα αθλητικά δρώμενα, τότε οι σελίδες με αθλητικό περιεχόμενο θα λαμβάνουν υψηλότερο score στον υπολογισμό του PageRank, οπότε θα βρίσκονται πιο ψηλά στην ιεραρχία των αποτελεσμάτων για ένα επερώτημα του συγκεκριμένου χρήστη. Το παραπάνω σκεπτικό συνιστά την ιδέα του Εξατομικευμένου (Personalized) PageRank, το οποίο θα αναλύσουμε αφού πρώτα εξετάσουμε τις βασικές αρχές του κύριου αλγορίθμου PageRank.

2.4.1 Original PageRank – Θεωρητικό Υπόβαθρο

Ο αλγόριθμος PageRank είναι ίσως ο πιο διαδεδομένος από τους παράγοντες που λαμβάνονται υπόψιν κατά τη διαδικασία αναζήτησης από την Google. Ο αλγόριθμος ορίζεται αναδρομικά και ως εκ τούτου εκτελείται επαναληπτικά. Για να επεξηγήσουμε τον αλγόριθμο, υποθέτουμε ότι ένας τυχαίος χρήστης του διαδικτύου επισκέπτεται μία σελίδα και στη συνέχεια επιλέγει στην τύχη συνδέσμους από εκείνη τη σελίδα και αυτό επαναλαμβάνεται συνεχώς. Το PageRank είναι ένα μέτρο για το πόσο συχνά επισκέπτεται ο χρήστης μας μία σελίδα. Για την ακρίβεια, το PageRank είναι μία κατανομή πιθανότητας για τους κόμβους του γράφου, η οποία αναπαριστά την

πιθανότητα ένα τυχαίο μονοπάτι πάνω στη δομή του γράφου να καταλήξει σε ένα συγκεκριμένο κόμβο. Για να εξηγήσουμε τον πλήρη μαθηματικό τύπο του PageRank, λαμβάνουμε επίσης υπόψιν την πιθανότητα ο χρήστης μας να αγνοήσει τους συνδέσμους της τρέχουσας σελίδας και να επισκεφτεί μία εντελώς διαφορετική σελίδα στην τύχη. Το PageRank της σελίδας p_i ορίζεται ως:

$$R_{k+1}(p_i) = d^* \left(\sum_{p_j \in Bp_i, |p_j| \neq 0} \frac{Rk(p_j)}{|p_j|} + \sum_{p_j, |p_j|=0} \frac{Rk(p_j)}{N} \right) + (1-d)^* \sum_{p_j} \frac{Rk(p_j)}{N} ,$$

όπου: $\mathbf{R_k(p_i)}$ η τιμή PageRank για τη σελίδα p_i στην επανάληψη k

N το πλήθος των σελίδων της συλλογής

d ο παράγοντας "damping factor", που συμβολίζει την πιθανότητα ο τυχαίος χρήστης να επιλέξει ένα σύνδεσμο από τη σελίδα που βρίσκεται. Ο παράγοντας (1-d) ονομάζεται "jump factor" και είναι η πιθανότητα ο χρήστης να μεταπηδήσει σε μία τυχαία διαφορετική σελίδα

$\mathbf{Bp_i}$ το σύνολο που απεικονίζει τους συνδέσμους προς τη σελίδα p_i

$|\mathbf{p_j}|$ το πλήθος των εξω-συνδέσμων από τη σελίδα p_j

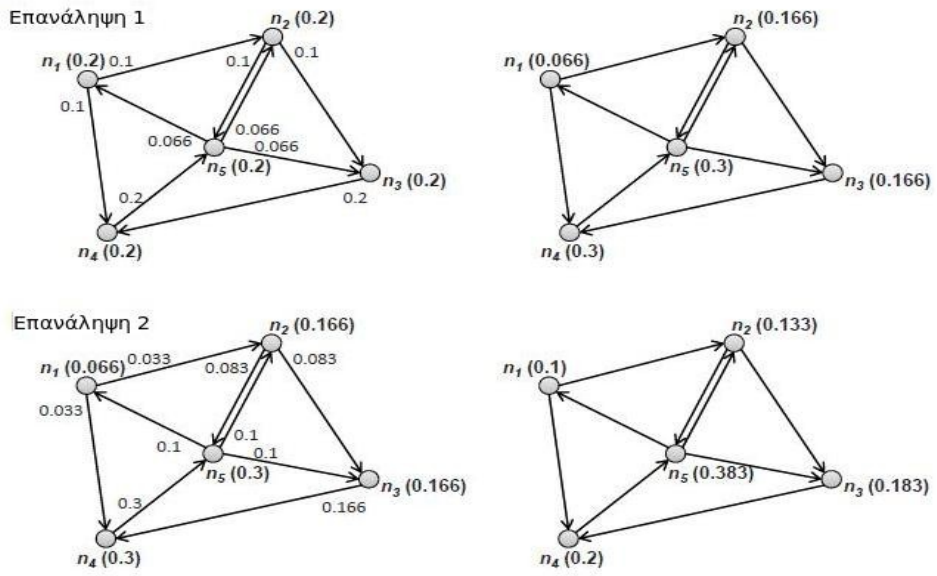
$\mathbf{R_0(p_i)=1/N}$, πριν την πρώτη επανάληψη υποθέτουμε ότι όλες οι σελίδες ξεκινούν έχοντας την ίδια τιμή, ίση με 1/N

Ας αναλύσουμε τους όρους του παραπάνω τύπου. Το πρώτο τμήμα απεικονίζει την πιθανότητα ο χρήστης να επιλέξει ένα σύνδεσμο από την τρέχουσα σελίδα. Η σελίδα p_i δέχεται "συνεισφορές" PageRank από όλες τις σελίδες Bp_i, οι οποίες περιέχουν σύνδεσμο προς αυτή. Έστω μία σελίδα p_j από το σύνολο Bp_i. Ένας τυχαίος χρήστης που βρίσκεται στην p_j θα φτάσει στην p_i με πιθανότητα $1/|p_j|$, καθώς επιλέχθηκε τυχαία ένας σύνδεσμος από όλους τους εξω-συνδέσμους της p_j. Εφόσον όμως η τιμή PageRank της p_j είναι στην ουσία η πιθανότητα ο χρήστης να βρίσκεται στην p_j, τότε η πιθανότητα να φτάσει στην p_i από την p_j είναι $R(p_j)/|p_j|$. Για να υπολογιστεί η τιμή PageRank για την p_i, χρειάζεται να αθροιστούν οι συνεισφορές από όλες τις σελίδες οι οποίες "δείχνουν" σε αυτήν, κάτι που απεικονίζεται στο πρώτο άθροισμα.

Τι γίνεται όμως με τους κόμβους που έχουν μηδενικό πλήθος εξερχόμενων ακμών; Το άθροισμα $\sum_{p_j} \frac{Rk(p_j)}{N}$ χρησιμοποιείται ώστε η τιμή PageRank που έχουν αποκτήσει εκείνοι οι κόμβοι (dangling nodes) να διαμοιραστεί ισότιμα σε όλους τους κόμβους του γράφου.

Όσον αφορά στο δεύτερο τμήμα του τύπου, που αναπαριστά την πιθανότητα ο τυχαίος χρήστης να μεταπηδήσει σε μία οποιαδήποτε σελίδα της συλλογής, περιλαμβάνει το άθροισμα των τιμών του PageRank για όλες τις σελίδες διαιρεμένων με το συνολικό πλήθος των σελίδων. Όμως, ισχύει ότι **το άθροισμα των τιμών PageRank όλων των σελίδων ισούται με 1**, δηλαδή $\sum_{p_j} Rk(p_j) = 1$, οπότε τελικά θα είναι (1-d)/N.

Στην εικόνα 2.8 δίνεται ένα παράδειγμα με δύο επαναλήψεις του αλγορίθμου, με ένα γράφο δίχως "dangling" κόμβους και αγνοώντας την τυχαία μεταπήδηση (random jump) που αναφέραμε παραπάνω (d=1).



Εικόνα 2.8: Παράδειγμα με δύο επαναλήψεις του αλγορίθμου PageRank. Οι αριστεροί γράφοι απεικονίζουν τις τιμές στην αρχή κάθε επανάληψης καθώς και τη συνεισφορά για κάθε γειτονικό κόμβο. Οι δεξιοί γράφοι δείχνουν τις ενημερωμένες τιμές στο τέλος κάθε επανάληψης.

Σε κάθε επανάληψη του αλγορίθμου, ενημερώνεται η τιμή κάθε κόμβου του γράφου, με βάση τους παραπάνω υπολογισμούς. Ο αλγόριθμος εκτελείται μέχρις ότου οι τιμές PageRank να συγκλίνουν (λαμβάνοντας υπόψιν κάποια τιμή ανοχής ε) ή μέχρι κάποιο αριθμό επαναλήψεων, ορισμένο από τον προγραμματιστή. Συγκεκριμένα:

- Αριθμός επαναλήψεων: $\cong \frac{-n}{\log_{10} d}$, Κριτήριο σύγκλισης: $\sum \left| r_{k+1}(p_i) - r_k(p_i) \right| < \varepsilon$

με $\varepsilon=10^{-n}$ και n: αριθμός σημαντικών ψηφίων.

Η διαδικασία υπολογισμού των τιμών PageRank για κάθε κόμβο του γράφου με N κόμβους, μπορεί επίσης να εκφραστεί ως ένα πρόβλημα υπολογισμού ιδιοτιμών[9][31]. Έστω **P** ο στοχαστικός πίνακας μεταβάσεων NxN από τον κόμβο i στον κόμβο j με $P_{ij}=1/\deg(i)$. Έστω **E** ο πίνακας NxN, για τον οποίο ισχύει $\mathbf{E=ev^T}$, όπου **e** είναι το N-διάνυσμα του οποίου τα στοιχεία είναι $e_i=1$ και **v** είναι το N-διάνυσμα του οποίου τα στοιχεία είναι όλα μη αρνητικά και έχουν άθροισμα 1. Αν **d** ο "damping" παράγοντας που είδαμε παραπάνω, τότε ορίζουμε μία αμείωτη αλυσίδα Markov A^T ως εξής:

$$\mathbf{A^T = [d*P + (1-d)*E]^T}$$

Σε κάθε χρονική στιγμή, ένας χρήστης θα μεταπηδήσει με πιθανότητα (1-d) σε μία τυχαία σελίδα και όχι σε κάποιον εξω-σύνδεσμο. Ο προορισμός της μεταπήδησης αυτής επιλέγεται με βάση την κατανομή πιθανότητας που καθορίζεται από το διάνυσμα **v**, για το οποίο ισχύει vi=1/N (ομοιόμορφη κατανομή).

2.4.2 Personalized PageRank – Θεωρητικό Υπόβαθρο

Προχωρώντας στην εξατομίκευση (personalization) του PageRank, είναι προφανές πως ο βασικός αλγόριθμος του PageRank χρειάζεται να τροποποιηθεί [12] ώστε να λαμβάνει υπόψιν τις προτιμήσεις των χρηστών για συγκεκριμένες σελίδες. Αυτό μπορεί να καταστεί δυνατό επεμβαίνοντας στο βήμα μεταπήδησης του τυχαίου χρήστη σε μία εντελώς διαφορετική σελίδα, αγνοώντας τους συνδέσμους που περιέχονται στη σελίδα στην οποία βρίσκεται σε μία στιγμή. Συγκεκριμένα, η πιθανότητα της τυχαίας μεταπήδησης στον τύπο του βασικού αλγορίθμου είναι (1-d)*(1/N), που σημαίνει ότι κάθε κόμβος του γράφου λαμβάνει ίση πιθανότητα τυχαίας επίσκεψης σε εκείνον. Σε αυτό το σημείο λοιπόν, ο υπολογισμός τροποποιείται με τέτοιο τρόπο ώστε η συγκεκριμένη πιθανότητα, έστω $\mathbf{w_{pi}}$, να ακολουθεί ανομοιόμορφη κατανομή, με βάση τις προτιμήσεις του εκάστοτε χρήστη. Έτσι, ο τύπος για το Personalized PageRank θα είναι:

$$R_{k+1}(p_i) = d^* \left(\sum_{p_j \in Bp_i, |p_j| \neq 0} \frac{Rk(p_j)}{|p_j|} + \sum_{p_j, |p_j|=0} \frac{Rk(p_j)}{N} \right) + (1-d)^* \mathbf{w_{pi}}$$

Το βάρος w_{pi} είναι κανονικοποιημένο έτσι ώστε να ισχύει $0 \leq w_{pi} < 1$ και $\sum_{p_i} w_{pi} = 1$.

Γενικά, η παράμετρος w_{pi} μπορεί να έχει διαφορετική τιμή για κάθε σελίδα προτίμησης, όμως στην περίπτωσή μας επιλέγουμε να ισχύει w_{pi}=1/numOfPreferredPages, όπου numOfPreferredPages το πλήθος των σελίδων προτίμησης του εκάστοτε χρήστη. Το δεύτερο μέλος του αθροίσματος, [(1-d)* w_{pi}], λαμβάνει την τιμή μηδέν για όλες τις υπόλοιπες σελίδες, εκτός των προτιμώμενων κάθε χρήστη.

Όσον αφορά στην απεικόνιση του προβλήματος χρησιμοποιώντας ιδιοδιανύσματα, είδαμε παραπάνω ότι ισχύει $A = [d^*P + (1-d)^*E]^T$, το οποίο γίνεται $A = d^*P^T + (1-d)^*ve^T$ [31]. Έστω τώρα x(v) το N-διάνυσμα εξατομικευμένου PageRank (personalized pagerank vector), το οποίο αντιστοιχεί στο N-διάνυσμα εξατομίκευσης (personalization vector) v, το οποίο ακολουθεί τώρα ανομοιόμορφη κατανομή ανάλογα με την πιθανότητα μεταπήδησης του χρήστη σε τυχαία σελίδα. Το πρόβλημα ανάγεται στη λύση της παρακάτω εξίσωσης :

$$\mathbf{x} = A\mathbf{x} \text{ ή}$$

$$\mathbf{x} = d^*P^T\mathbf{x} + (1-d)^*\mathbf{v}$$

Αν το διάνυσμα v ακολουθεί ομοιόμορφη κατανομή v = [1/N,.....,1/N], τότε η αντίστοιχη λύση x(v) είναι το καθολικό διάνυσμα PageRank χωρίς να λαμβάνονται υπόψιν προτιμήσεις για τις σελίδες. Σε αντίθετη περίπτωση, το x(v) αντιστοιχεί στο διάνυσμα Personalized PageRank που προκύπτει με βάση τις προτιμήσεις για συγκεκριμένες σελίδες, όπως απεικονίζονται με πιθανότητες στο διάνυσμα v.

2.4.3 Συναφείς Μελέτες

Ο πρώτος αλγόριθμος σχετικός με Personalized PageRank προτάθηκε στο [12] ως μία παραλλαγή του PageRank και θεωρείται ότι τα αντίστοιχα scores υπολογίζονται *online*, δηλαδή αφότου δοθεί το εκάστοτε επερώτημα. Φυσικά, αυτό καθιστά αδύνατη την άμεση εξαγωγή αποτελεσμάτων, δεδομένου ενός επερωτήματος. Σε κάθε περίπτωση, δε δόθηκε ιδιαίτερη έμφαση στον υπολογισμό των εξατομικευμένων (personalized) scores, πέρα από εκείνα του κύριου αλγορίθμου.

Στο [13] προτάθηκε από τον Haveliwala η έννοια του “Topic-Sensitive” PageRank, όπου η εξατομίκευση έγκειται στα βάρη που δίνονται σε συγκεκριμένες κατηγορίες σελίδων. Συγκεκριμένα, σε πρώτη φάση τα scores για τις σελίδες προϋπολογίζονται *offline*, λαμβάνοντας υπόψιν τη κατηγορία στην οποία ανήκει η κάθε σελίδα. Στη συνέχεια, το εκάστοτε επερώτημα ταξινομείται σε μία από 16 συνολικά κατηγορίες σε online χρόνο με σκοπό την ανάθεση βαρών στις κατηγορίες. Το κύριο μειονέκτημα της μεθόδου είναι ο περιορισμός της χρήσης 16 κύριων κατηγοριών καθώς δεν υπάρχει η δυνατότητα ανάθεσης βάρους σε μεμονωμένες σελίδες, κάτι που θα απαιτούσε απαγορευτικά περισσότερους υπολογισμούς. Μία άλλη προσπάθεια που είχε ως στόχο την μείωση της ποσότητας των υπολογισμών και του αποθηκευτικού όγκου που θα απαιτούνταν για την υλοποίηση του [12], γίνεται στο [9], όπου υπολογίζονται offline περιορισμένες τιμές του PageRank. Στην ουσία, υπολογίζονται μερικώς κάποια Personalized PageRank διανύσματα με βάση ένα σύνολο *H* από προτιμώμενες σελίδες. Ο περιορισμός της συγκεκριμένης μεθόδου είναι ότι γίνεται χρήση ενός υποσυνόλου των σελίδων, οι οποίες απαιτείται να έχουν υψηλό score ώστε να επιτυγχάνεται υψηλή απόδοση στους υπολογισμούς. Σε αντίθετη περίπτωση, αν δηλαδή π.χ. οι σελίδες επιλέγονται τυχαία, το μέγεθος των μερικών διανυσμάτων αυξάνεται σημαντικά.

Ο Fogaras στο [14] κινήθηκε προς την κατεύθυνση της υλοποίησης του πλήρους Personalized PageRank, χωρίς δηλαδή να υπάρχει ο περιορισμός της επιλογής συγκεκριμένων σελίδων ως προτιμώμενες από κάποιο χρήστη, χρησιμοποιώντας τη μέθοδο Monte Carlo. Η μέθοδος Monte Carlo είναι η εναλλακτική επιλογή για τον υπολογισμό του Personalized PageRank, αντίθετα με τη μέθοδο που προτάθηκε στο [12]. Η τεχνική που παρουσιάστηκε είναι η ακόλουθη: Ξεκινώντας από κάποιο κόμβο u του γράφου, διασχίζουμε R τυχαία μονοπάτια που ξεκινούν από το u, καθένα από τα οποία έχει μήκος γεωμετρικά κατανεμημένο ως Geom(1-c), όπου c ο “damping” factor που είδαμε παραπάνω. Έτσι, οι συχνότητες των επισκέψεων σε διαφορετικούς κόμβους αυτών των μονοπατιών θα προσεγγίζουν τις τιμές του Personalized PageRank.

Η πιο σημαντική διαφορά των παραπάνω με τη δική μας υλοποίηση είναι φυσικά η χρήση του Hadoop Map-Reduce, που μας επιτρέπει την αποδοτική επεξεργασία μεγάλου όγκου δεδομένων σε πολλά μηχανήματα ταυτόχρονα. Στο [15] παρουσιάζονται τεχνικές υλοποίησης του PageRank σε Map-Reduce, οι οποίες αυξάνουν την απόδοση και το χρόνο εκτέλεσης των ήδη υπαρχόντων υλοποιήσεων. Μία ακόμη εναλλακτική υλοποίηση του PageRank στο Hadoop δόθηκε από το σύστημα Pegasus [16], όπου ο υπολογισμός των τιμών του PageRank στηρίζεται σε έναν επαναλαμβανόμενο πολλαπλασιασμό πίνακα-διανύσματος και ανταποκρίνεται αποτελεσματικά σε γράφους μεγάλου μεγέθους. Στο [17] προτείνεται η υλοποίηση του πλήρους Personalized PageRank σε Map-Reduce, η οποία βασίζεται στο [14] και αποτελεί ότι πιο νέο υπάρχει στην περιοχή που εξετάζουμε. Στην ουσία, προτείνεται ένας αποτελεσματικός αλγόριθμος σε Map-Reduce για τον υπολογισμό μεμονωμένων τυχαίων μονοπατιών που ξεκινούν από κάθε κόμβο του γράφου (μέθοδος Monte Carlo). Στην παρούσα εργασία, γίνεται χρήση της γραμμικής αλγεβρικής μεθόδου για τον υπολογισμό του Personalized PageRank [12], σε συνδυασμό με την υλοποίηση του PageRank σε Map-Reduce στο [15]. Τα PageRank scores είναι στατικά και βασίζονται αποκλειστικά στη δομή του γράφου των διαθέσιμων κόμβων-σελίδων. Προκύπτει λοιπόν ότι είναι ανεξάρτητα από το εκάστοτε query που θα τεθεί στη μηχανή αναζήτησης.

3. Indexing και Vector Space Model στο Hadoop Map-Reduce

Στο παρόν κεφάλαιο γίνεται η περιγραφή της υλοποίησης του πρώτου τμήματος της μηχανής αναζήτησης, το οποίο βασίζεται αποκλειστικά στο λεκτικό περιεχόμενο των ιστοσελίδων. Ξεκινώντας από τη φάση του crawling, παρουσιάζεται ο τρόπος με τον οποίο χρησιμοποιήθηκε το Nutch ώστε να συλλεχθεί ένας μεγάλος όγκος από δεδομένα, καθώς και η απαραίτητη επεξεργασία την οποία υπέστησαν. Θα περιγραφεί, επίσης, βήμα προς βήμα η κατασκευή του Inverted Index σε συνδυασμό με τα scores που προκύπτουν από τη χρήση του μοντέλου VSM, χρησιμοποιώντας το μοτίβο προγραμματισμού Hadoop Map-Reduce.

3.1 Ανάλυση των απαιτήσεων

Όσον αφορά το συγκεκριμένο κομμάτι της εργασίας, ο στόχος είναι η υλοποίηση ενός ολοκληρωμένου συστήματος για την εκμετάλλευση του λεκτικού περιεχομένου των σελίδων ώστε να προκύψουν ουσιαστικά αποτελέσματα για τις διάφορες αναζητήσεις. Αρχικά, χρησιμοποιείται ο Nutch crawler, που εκτελείται στον Hadoop Cluster και καλύπτει με το παραπάνω την ανάγκη για έναν αποδοτικό crawler. Το διαθέσιμο περιεχόμενο των σελίδων που έχει "κατεβάσει" το Nutch, πρέπει να επεξεργαστεί με κατάλληλο τρόπο ώστε να κατασκευαστεί μία δομή ευρετηρίου που θα χρησιμεύσει στη δημιουργία του συστήματος μηχανής αναζήτησης.

Εκείνη η δομή δεν είναι άλλη από τον **Inverted Index**, ο οποίος θα κατασκευαστεί με βάση το μοτίβο Map-Reduce, ώστε να εκμεταλλευτούμε τις δυνατότητες του Hadoop. Ο Inverted Index προσφέρει ταχύτητα στην ανάκτηση των σελίδων στις οποίες περιλαμβάνεται ένας συγκεκριμένος όρος της συλλογής, γι' αυτό και είναι δομικό συστατικό των μηχανών αναζήτησης. Φυσικά, η συγκεκριμένη δομή δεν αρκεί για την εξαγωγή των τελικών αποτελεσμάτων.

Είναι λοιπόν απαραίτητη η παρουσία μέτρων με βάση τα οποία θα προκύπτουν τα πιο σχετικά αποτελέσματα για κάθε αναζήτηση. Γι' αυτό το λόγο, υπολογίστηκαν scores τα οποία βασίζονται στο Vector Space Model. Αρχικά, υπολογίστηκαν τα **tf-idf scores** στο Hadoop και στη συνέχεια ενσωματώθηκαν στον Inverted Index, δημιουργώντας έτσι έναν νέο βελτιωμένο Index, ο οποίος περιλαμβάνει επιπρόσθετη πληροφορία για τη σχέση ενός όρου με τα κείμενα στα οποία τον συναντάμε.

Η αξιολόγηση μίας μηχανής αναζήτησης, όμως, είναι δυνατή μόνο εάν γίνουν δοκιμές με πραγματικά επερωτήματα (queries), με βάση τα οποία θα παραχθεί μία λίστα αποτελεσμάτων με σχετικές σελίδες. Οπότε, ακολουθώντας τις προσταγές του VSM μοντέλου, υπολογίζεται σε λογική Map-Reduce το μέτρο **Cosine Similarity**, το οποίο υποδεικνύει τη σχέση του query με τις

διαθέσιμες σελίδες. Έτσι, οι σελίδες με υψηλή τιμή για το συγκεκριμένο μέτρο, βρίσκονται και στις πρώτες θέσεις στην κατάταξη των αποτελεσμάτων για τις αναζητήσεις που εκτελούνται.

3.2 Φάση Crawling

Στην ενότητα 2.2 περιγράψαμε τη λειτουργία ενός τυπικού web crawler, ενώ συγκεκριμένα στην 2.2.2 παρουσιάστηκε το Nutch, καθώς επίσης και οι εντολές οι οποίες λαμβάνουν χώρα στη διαδικασία του crawling. Στην εργασία μας, θελήσαμε να συλλέξουμε περίπου 3 με 4 εκατομμύρια ιστοσελίδες, συμπεριλαμβανομένου του περιεχομένου τους και του γράφου που απεικονίζει τους συνδέσμους μεταξύ των σελίδων.

Αρχικά, έπρεπε να οριστεί μία λίστα από root urls, από τα οποία ξεκινάει η διαδικασία του crawling. Χρησιμοποιήσαμε περίπου 4200 τυχαία root urls από το DMOZ open directory [30], τα οποία σε πρώτη φάση εισάγονται στη βάση δεδομένων crawldb, ώστε να συλλεχθούν στη συνέχεια τα περιεχόμενα των αντίστοιχων σελίδων. Αυτό επιτυγχάνεται με την εντολή **inject**:

bin/nutch inject user/gfragiadoulakis/crawlLarge/crawldb user/gfragiadoulakis/urls , όπου urls το αρχείο με τα root urls.

Στη συνέχεια, εκτελέστηκαν οι παρακάτω εντολές σε 12 επαναλήψεις, όπως φαίνεται από το script:

```
function echoThenRun () { # echo and then run the command
        echo $1
        $1
        echo
}

for (( i = 0; i < 12; i++ ))

do
        echoThenRun "bin/nutch generate user/gfragiadoulakis/crawlLarge/crawldb
        user/gfragiadoulakis/crawlLarge/segments -topN 620000"

        seg=`/usr/local/hadoop020203/bin/hadoop fs -ls
        user/gfragiadoulakis/crawlLarge/segments | cut -b 104- | tail -1`

        echoThenRun "bin/nutch fetch user/gfragiadoulakis/crawlLarge/segments/$seg -
threads 10"

        echoThenRun "bin/nutch parse user/gfragiadoulakis/crawlLarge/segments/  $seg"

        echoThenRun "bin/nutch updatedb user/gfragiadoulakis/crawlLarge/crawldb
        user/gfragiadoulakis/crawlLarge/segments/$seg -filter"

done
```

Εικόνα 3.1: Script εντολών Nutch

Συγκεκριμένα, σε κάθε επανάληψη επιλέγονται τα 620000 urls με το πιο υψηλό score (με βάση το scoring του Nutch – εντολή **generate**), ώστε να συλλεχθούν οι ιστοσελίδες που ανταποκρίνονται σε αυτά (εντολή **fetch**) και να επιτευχθεί ο στόχος του όγκου των δεδομένων που θεωρήσαμε ότι είναι ικανός για να τονιστούν τα πλεονεκτήματα του Hadoop. Το περιεχόμενο των σελίδων αναλύεται κατάλληλα (εντολή **parse**), ενώ στη συνέχεια ενημερώνεται η βάση δεδομένων crawldb με τα καινούρια urls, τους συνδέσμους τους προς άλλες σελίδες, καθώς και με άλλου τύπου μεταδεδομένα (εντολή **updatedb**).

Τέλος, δημιουργείται μία ανεστραμμένη δομή μέσω της βάσης δεδομένων linkdb, η οποία αποθηκεύει όλους τους εισερχόμενους συνδέσμους για κάθε url (εντολή **invertlinks**):

bin/nutch invertlinks user/gfragiadoulakis/crawlLarge/linkdb -dir user/gfragiadoulakis/crawlLarge/segments

Με την παραπάνω διαδικασία, έγινε εφικτή η συλλογή 7523958 σελίδων, από τις οποίες εξάγαμε (βλ. ενότητα 3.3) μόνο εκείνες που είναι στην αγγλική γλώσσα, δηλαδή 3689305 σελίδες, που ήταν ο αρχικός μας στόχος.

3.3 Επεξεργασία των δεδομένων

Τα αποτελέσματα μετά τη διαδικασία του crawling βρίσκονται αποθηκευμένα στο HDFS ως SequenceFiles και περιλαμβάνουν αρκετή επιπλέον πληροφορία, μαζί με το λεκτικό περιεχόμενο της κάθε σελίδας. Γίνεται λοιπόν εύκολα αντιληπτό ότι χρειάζεται να επεξεργαστούν κατάλληλα ώστε να δοθούν στη συνέχεια ως είσοδος στα Map-Reduce jobs που είναι υπεύθυνα για λειτουργίες όπως η κατασκευή του Inverted Index, ο υπολογισμός των tf-idf scores κ.α.. Αρχικά, θεωρήσαμε πως θα ήταν καλό να έχουμε τα αποτελέσματα του Nutch σε μορφή απλού κειμένου ώστε να κάνουμε μία επισκόπηση του περιεχομένου που συλλέξαμε. Το Nutch μας δίνει τη δυνατότητα να μετατρέψουμε το περιεχόμενο των σελίδων σε μορφή κειμένου, χρησιμοποιώντας την παρακάτω εντολή:

- Εντολή readseg: *bin/nutch readseg -dump user/gfragiadoulakis/crawlLarge/segments/* user/gfragiadoulakis/dumpSegs -nocontent -nofetch -nogenerate -noparse -noparsedata*

Το περιεχόμενο των σελίδων που βρίσκεται στο directory *segments* μετατρέπεται σε κείμενο και αποθηκεύεται στο directory *dumpSegs2* στο HDFS. Οι παράμετροι *-nocontent -nofetch -nogenerate -noparse -noparsedata* χρησιμοποιούνται ώστε να εξαχθεί μόνο το λεκτικό περιεχόμενο των σελίδων (parse_text) και καμία άλλη πληροφορία για το συγκεκριμένο segment (βλ. Ενότητα 2.2.2).

Η παραπάνω μετατροπή της διαθέσιμης πληροφορίας σε απλό κείμενο επέφερε εγγραφές με συγκεκριμένη μορφή.Ένα παράδειγμα μίας εγγραφής που απεικονίζει το περιεχόμενο μίας σελίδας της συλλογής μας, με την εντολή *readseg,* φαίνεται στην εικόνα 3.2.

Φυσικά, η μορφή εκείνη δεν είναι η καταλληλότερη για να δοθεί ως είσοδος σε μία Map-Reduce job. Γι' αυτό το λόγο, μετατρέψαμε την πληροφορία για το περιεχόμενο των σελίδων στο αρχείο *nutchDocs* (κλάση **DocsExtractor.java**), του οποίου οι εγγραφές είναι της μορφής:

<url1> <webpage_text1>

<url2> <webpage_text2> κοκ.

Recno:: 159

URL:: http://afortmadeofbooks.blogspot.com/2011/06/super-8.html

ParseText::

A Fort Made of Books: Super 8 A Fort Made of Books Theology, Art, Entertainment, and Cat Litter in St. Louis, USA Saturday, June 18, 2011 Super 8 I missed my chance to see Thor . But other than that, today offered a "perfect storm" of summer blockbusters. I could have chosen a Jim Carrey vehicle,.....

Εικόνα 3.2: Το περιεχόμενο μίας εγγραφής μετά την εκτέλεση της εντολής "readseg"

Εξαγωγή σελίδων αγγλικού περιεχομένου

Το κείμενο που αφορά την εκάστοτε ιστοσελίδα μπορεί να είναι σε διαφορετική γλώσσα, καθώς το Nutch δεν έχει κάποιο περιορισμό στη γλώσσα του περιεχομένου των σελίδων που συλλέγει. Λόγω του γεγονότος ότι επιλέξαμε να διαχειριστούμε σελίδες μονάχα αγγλικού περιεχομένου, σε αυτό το σημείο γίνεται μία κατηγοριοποίηση του κειμένου κάθε σελίδας. Αυτό έγινε με μία απλή Map-Reduce job (κλάση **RemoveNonEnglishPages.java**), στην οποία χρησιμοποιήθηκε η βοηθητική κλάση **TextCategorizer.java** [23]. Ως είσοδος δίνεται το αρχείο *nutchDocs*. Στη συνέχεια, κατά τη φάση Map, γίνεται έλεγχος για το περιεχόμενο κάθε σελίδας ούτως ώστε στην έξοδο της συνάρτησης map() να έχουμε μόνο σελίδες αγγλικού περιεχομένου. Στη συγκεκριμένη job δε χρειάζεται να λάβει χώρα φάση Reduce. Το αρχείο *nutchDocsEnglish* που προκύπτει, θα είναι της ίδιας μορφής με το *nutchDocs*. Τελικά, το σύνολο των ιστοσελίδων που θα διαχειριστούμε και προέκυψε από την παραπάνω διαδικασία, είναι **3689304** σελίδες.

Αντικατάσταση των urls με ακέραια ids

Φτάσαμε λοιπόν στο σημείο να έχουμε την απαραίτητη πληροφορία στην κατάλληλη μορφή για επεξεργασία μέσω του Map-Reduce. Υπάρχουν όμως περιθώρια για τη βελτίωση των δεδομένων εισόδου, από πλευράς μεγέθους της πληροφορίας η οποία υφίσταται επεξεργασία. Παρόλο που το HDFS παρέχει τεράστιο αποθηκευτικό χώρο, θεωρήσαμε σημαντικό να περιορίσουμε τον όγκο των δεδομένων, εφαρμόζοντας την εξής απλή σύμβαση: Κάθε url της συλλογής μας θα αντιπροσωπεύεται από έναν και μοναδικό θετικό ακέραιο αριθμό (id). Για να κατανοήσουμε τη σημασία της παραπάνω σύμβασης, αρκεί μόνο να σκεφτούμε ότι το εκάστοτε url, που είναι τύπου Text, έχει μέγεθος τόσα bytes όσα και το πλήθος των χαρακτήρων του, ενώ αντίθετα κάθε id θα είναι τύπου IntWritable και θα έχει μέγεθος 4 bytes. Έτσι, δίνοντας ως είσοδο το αρχείο *nutchDocsEnglish* σε μία Map-Reduce job, κατασκευάσαμε δύο Hadoop Mapfiles που περιέχουν εγγραφές της μορφής <key> <value> (κλάση **CreateUrlId_File.java**). Το πρώτο Mapfile έχει ως κλειδιά τα urls της συλλογής μας και ως τιμές τα ids που τους αντιστοιχούν, ενώ το δεύτερο

Mapfile έχει ως κλειδιά τα ids και ως τιμές τα αντίστοιχα urls και κατασκευάστηκε ώστε να υπάρχει η δυνατότητα ανά πάσα στιγμή να λάβουμε το url που αντιστοιχεί σε κάθε id.

Για την επεξεργασία τώρα του αρχείου *nutchDocsEnglish* και τη μετατροπή των urls σε ids, εκτελέσαμε μία Map-Reduce job (κλάση **ConvertUrlsContentToIds.java**), η οποία λαμβάνει ως είσοδο το αντίστοιχο αρχείο, χρησιμοποιεί το πρώτο Hadoop Mapfile (της μορφής <url> <id>) και εξάγει ένα αρχείο που πλέον αντί για urls, περιλαμβάνει τα ακέραια ids που τους αντιστοιχούν. Το εξαγόμενο αρχείο είναι το *nutchContent_ids* και περιλαμβάνει εγγραφές της μορφής:

<url_ID1> <webpage_text1>

<url_ID2> <webpage_text2> κοκ.

Έχοντας κάνει την απαραίτητη επεξεργασία στα αποτελέσματα του crawling και τις κατάλληλες μετατροπές όσον αφορά το λεκτικό περιεχόμενο των σελίδων, μπορούμε να προχωρήσουμε στη διαδικασία κατασκευής του Inverted Index στο Hadoop. Όσο για την επεξεργασία των αποτελεσμάτων του Nutch σχετικά με την κατασκευή του γράφου που αναπαριστά τον τρόπο με τον οποίο συνδέονται οι σελίδες της συλλογής, θα παρουσιαστεί αναλυτικά στην ενότητα 4.2.

3.4 Κατασκευή του Inverted Index σε Map-Reduce

Ήδη από την ενότητα 2.3.1 έχουμε δείξει σε γενικές γραμμές τον τρόπο με τον οποίο κατασκευάζεται ένας τυπικός Inverted Index σε λογική Map-Reduce. Η συγκεκριμένη δομή μας επιτρέπει την απευθείας πρόσβαση στα κείμενα που εμφανίζεται κάποιος όρος, ενώ με το μοτίβο Map-Reduce γίνεται εφικτή η παραλληλοποίηση των διαδικασιών που επιτρέπουν τη δημιουργία ενός κατακερματισμένου Index στο HDFS. Τώρα, δίνοντας περισσότερες λεπτομέρειες, θα αναλύσουμε την τεχνική που ακολουθήσαμε ώστε να επιτύχουμε τη δημιουργία ενός κατανεμημένου Index στο Hadoop.

Δεδομένα Εισόδου

Τα δεδομένα εισόδου για τη Map-Reduce job (κλάση **BuildInvertedIndex.java**) που είναι υπεύθυνη για τη δημιουργία του Inverted Index, είναι στην ουσία το αρχείο *nutchContent_ids*, που προέκυψε από τη μετατροπή των urls σε ids, όπως εξηγήσαμε στην προηγούμενη ενότητα και έχει την εξής δομή:

<url_id> <webpage_text>

Το αρχείο εισόδου δίνεται σε μορφή text (TextInputFormat).

Φάση Map

Η κάθε διεργασία map λαμβάνει ως είσοδο μία γραμμή του αρχείου εισόδου *nutchContent_ids*. Αρχικά, λαμβάνει χώρα η διαδικασία *tokenization*, όπου το κείμενο κατατμείται στους όρους που το αποτελούν. Στη συνέχεια, αφαιρούνται τα *stopwords* (χρησιμοποιείται η κλάση **TextAnalyser.java**), γίνεται μετατροπή των κεφαλαίων γραμμάτων σε πεζά, ενώ σε κάθε όρο του κειμένου επενεργεί ένας *stemmer* (**IteratedLovinsStemmer.java**), που μετασχηματίζει τον όρο σύμφωνα με τους κανόνες γραμματικής της αγγλικής γλώσσας. Για κάθε όρο του εκάστοτε κειμένου, η έξοδος της map θα είναι ο όρος ως κλειδί, ενώ η αντίστοιχη τιμή θα περιέχει το id του κειμένου, καθώς και το πλήθος των εμφανίσεων του όρου μέσα στο κείμενο (term frequency).

Συνοπτικά η έξοδος της map:

key	value
<term>	<docNumber,termCounts>

Φάση Partitioning

Στη φάση του Partitioning υπάρχει η δυνατότητα τα ενδιάμεσα δεδομένα να κατανεμηθούν σε διαφορετικό reducer με βάση μία συνάρτηση δωσμένη από το προγραμματιστή. Για παράδειγμα, τα δεδομένα θα μπορούσαν να ομαδοποιηθούν με βάση το κείμενο (δηλαδή μία πλειάδα από κείμενα θα επεξεργάζονταν από έναν reducer, ανεξάρτητα με τα κείμενα στα οποία εμφανίζεται ο εκάστοτε όρος). Παρ' όλα αυτά, στη συγκεκριμένη φάση δεν ορίσαμε κάποια καινούρια συνάρτηση, οπότε χρησιμοποιείται ο default partitioner του Hadoop, δηλαδή ο Hash Partitioner (βλ. ενότητα 2.1.1). Έτσι, τα δεδομένα μεταβαίνουν στον εκάστοτε reducer με βάση μία συνάρτηση κατακερματισμού ώς εξής: *key%numOfReducers*. Οπότε, κάθε όρος μαζί με τις αντίστοιχες τιμές θα επεξεργαστεί από έναν reducer.

Φάση Reduce

Η συνάρτηση reduce() λαμβάνει ως είσοδο τα δεδομένα εξόδου της map(), δηλαδή για κάθε όρο λαμβάνει ένα πλήθος από ζευγάρια της μορφής <docNumber,termCounts> και στην ουσία υπολογίζει το πλήθος των κειμένων στα οποία ο εξεταζόμενος όρος εμφανίζεται τουλάχιστον μία φορά (document frequency). Για κάθε όρο, δημιουργείται μία posting list (βλ. Ενότητα 2.3.1), η οποία περιέχει ζεύγη <docNumber,termCounts>, ένα για κάθε κείμενο στο οποίο περιέχεται ο όρος. Τα δεδομένα εξόδου αποθηκεύονται σε SequenceFiles (SequenceFileOutputFormat). Τα SequenceFiles είναι αρχεία που αποτελούνται από δυαδικά ζευγάρια της μορφής <key,value> και τα χρησιμοποιούμε διότι παρέχουν εύκολη διαχείριση των δεδομένων που βρίσκονται στο HDFS.

Η έξοδος της reduce() θα είναι:

key	value
<term>	<docFrequency, postingList(docNumber,termCounts)>

Έχουμε κατασκευάσει λοιπόν μία πρώτη εκδοχή του Inverted Index στο Hadoop. Στη συνέχεια, θα δούμε πώς μπορούμε να κατασκευάσουμε τον Inverted Index σε μία πιο ολοκληρωμένη μορφή, συμπεριλαμβάνοντας στις posting lists τα scores tf-idf και idf, όπως θα δούμε στην ενότητα 3.6.

3.5 Υπολογισμός των tf-idf scores σε Map-Reduce

Ο Inverted Index, του οποίου η κατασκευή εξηγήθηκε προηγουμένως, είναι μία δομή που προσφέρει πολλά πλεονεκτήματα όσον αφορά στη γρήγορη ανάκτηση πληροφορίας και αποτελεί ένα σημαντικό κομμάτι σε μία μηχανή αναζήτησης. Έχοντας δημιουργήσει τον Inverted Index για μία συλλογή κειμένων, γίνεται εφικτή η ταχεία αναζήτηση όρων σε όλο το εύρος της συλλογής.

Ωστόσο, είναι εξίσου σημαντικό η αναζήτηση να έχει όσο το δυνατό καλύτερη απόδοση σχετικά

με τα αποτελέσματα που επιστρέφονται στον εκάστοτε χρήστη της μηχανής αναζήτησης. Είναι λοιπόν απαραίτητο να γίνεται σωστή αξιολόγηση των αποτελεσμάτων από το σύστημα, με σκοπό την εξαγωγή των πιο σχετικών από αυτά με βάση τα επερωτήματα που δέχεται από τους χρήστες. Ένα από τα πιο διαδεδομένα και αξιόπιστα scores που χρησιμοποιούνται στις μηχανές αναζήτησης, είναι το tf-idf score.

Στην ενότητα 2.3.2 παρουσιάστηκε ο τρόπος με τον οποίο υπολογίζεται το tf-idf score για ένα ζεύγος <term,document>. Στην παρούσα ενότητα θα αναλυθεί ο υπολογισμός των tf-idf scores σε λογική Map-Reduce, με σκοπό την εξαγωγή του συγκεκριμένου score για κάθε ζεύγος <term,document> της συλλογής μας. Η υλοποίηση που έγινε για το συγκεκριμένο υπολογισμό απαιτεί 3 Map-Reduce jobs, τα οποία παρουσιάζονται εκτενώς παρακάτω:

3.5.1 Φάση "WordCount"

Η 1η φάση υπολογισμού των tf-idf scores αποσκοπεί στην εξαγωγή του πλήθους των εμφανίσεων ενός όρου σε ένα κείμενο (wordcount), κάτι που είναι απαραίτητο για τον υπολογισμό του term frequency (βλ. ενότητα 2.3.2.).

Δεδομένα Εισόδου

Η είσοδος σε αυτή τη φάση (κλάση **WordCount1.java**) θα είναι τα δεδομένα που προέκυψαν από το βήμα κατασκευής του Inverted Index. Δηλαδή:

key	**value**
<term>	<docFrequency, postingList(docNumber,termCounts)>

Φάση Map

Η συνάρτηση map() είναι εξαιρετικά απλή καθώς εξάγει απλά τα δεδομένα που εισήχθησαν.

Φάση Reduce

Ο σκοπός της συνάρτησης reduce() είναι να εξάγει για κάθε όρο και κάθε κείμενο, το πλήθος των εμφανίσεων του όρου στο εκάστοτε κείμενο. Οπότε, το κλειδί που θα προκύψει θα πρέπει να περιέχει τον όρο και το id του κειμένου, ενώ την αντίστοιχη τιμή θα αποτελεί το πλήθος των εμφανίσεων ως εξής:

key	**value**
<term,docId>	<termCount>

Το κλειδί είναι τύπου *PairOfStringInt,* ενώ η τιμή τύπου *IntWritable.*

3.5.2 Φάση "WordCountPerDocument-Normalized"

Στη 2η φάση, ο στόχος είναι να υπολογίσουμε τον παρονομαστή του τύπου $\mathrm{tf}_{i,d} = \frac{tc_{i,d}}{\sqrt{\sum_{k=1}^{N} tc_{k,d}^2}}$,

ώστε να έχουμε και τους 2 όρους του κλάσματος που αφορά την term frequency.

Φάση Map

Ως είσοδος στο συγκεκριμένο job (κλάση **WordCountPerDoc2.java**) δίνονται τα δεδομένα εξόδου της φάσης 1, δηλαδή το ζεύγος (<term,docId>,<termCount>). Στη φάση Reduce χρειαζόμαστε ως κλειδί μονάχα το docId, ώστε να έχουμε τη δυνατότητα για κάθε docId να υπολογίσουμε τον όρο $\sqrt{\sum_{k=1}^{N} tc_{k,d}^2}$, για το εκάστοτε κείμενο. Αυτό γίνεται εφικτό εξάγοντας από τη συνάρτηση map() δεδομένα της μορφής:

key	**value**
<docId>	<term,termCount>

Φάση Reduce

Η συνάρτηση reduce() δέχεται δεδομένα του τύπου <docId>=>listOf(<term,termCount>).

Εκμεταλλευόμαστε αυτο το γεγονός ώστε να υπολογίσουμε για κάθε κείμενο τον όρο $\sqrt{\sum_{k=1}^{N} tc_{k,d}^2}$ αθροίζοντας επαναληπτικά τις τιμές $tc_{k,d}^2$. Οπότε αυτό που μένει είναι να γίνει η εγγραφή των δεδομένων εξόδου στο HDFS. Η μορφή τους θα είναι η εξής:

key	**value**
<term,docId>	<termCount,termsNorm>

Το key είναι τύπου *PairOfStringInt,* ενώ η value τύπου *PairOfInts.*

Ο λόγος που τα ζεύγη του κλειδιού και της αντίστοιχης τιμής έχουν τη παραπάνω μορφή είναι επειδή για τον υπολογισμό του tf-idf score για κάθε ζεύγος <term,docId>, χρειάζονται οι όροι termCount και termsNorm, καθώς ο λόγος τους αποτελεί την **κανονικοποιημένη τιμή** της term frequency.

3.5.3 Φάση "Tf-Idf Calculation"

Στην 3η και τελευταία φάση (κλάση **DocCountperWord3.java**) υπολογίζονται τα τελικά tf-idf scores για κάθε ζεύγος <term,docId>. Παρουσιάζονται αναλυτικά οι φάσεις Map και Reduce και η λειτουργία τους.

Φάση Map

Τα δεδομένα εισόδου της φάσης Map θα είναι τα δεδομένα που προέκυψαν από την έξοδο της συνάρτησης reduce() της 2η φάσης, δηλαδή: <term,docId> => <termCount,termsNorm>. Ο σκοπός της φάσης 3, όπως είπαμε, είναι ο υπολογισμός των tf-idf scores. Όσον αφορά την τιμή term frequency, έχουμε στη διάθεσή μας τις απαραίτητες τιμές termCount και termsNorm, ώστε να

την υπολογίσουμε. Για την τιμή idf, όμως, χρειάζεται να υπολογιστεί και η τιμή document frequency, ώστε να συνδυαστεί με την term frequency και τελικά να εξδαχθούν τα tf-idf scores. Γι' αυτό το λόγο, τα δεδομένα εξόδου της map() θα έχουν ως εξής:

key	**value**
<term>	<docId,(termCount,termsNorm)>

Θα εξηγήσουμε στη φάση Reduce για ποιο λόγο η μορφή των παραπάνω δεδομένων έχει τη συγκεκριμένη μορφή.

Το key είναι τύπου *Text* και η value είναι τύπου *PairOfWritables<IntWritable,PairOfInts>*.

<u>Φάση Reduce</u>

Στη συνάρτηση reduce() λαμβάνουμε τα δεδομένα με μορφή <term> => listOf(<docId, (termCount,termsNorm)>). Αρχικά, γίνεται η καταμέτρηση των κειμένων (*docsPerTerm*) στα οποία ο τρέχων όρος εμφανίζεται τουλάχιστον μία φορά, δηλαδή για κάθε term, μετρώνται τα docIds που βρίσκονται στη λίστα εισόδου της value για το συγκεκριμένο term (document frequency).

Στη συνέχεια, γίνεται ξανά διάσχιση της λίστας των κειμένων που ανταποκρίνονται στον εκάστοτε όρο, ώστε για κάθε κείμενο να συγκεντρωθούν οι απαραίτητες τιμές για τον τελικό υπολογισμό του tf-idf score για κάθε ζεύγος <term,docId>. Έχοντας διαθέσιμες τις τιμές *termCounts, termsNorm*, το συνολικό πλήθος των κειμένων της συλλογής μας (*totalDocs*) και την τιμή *docsPerTerm* που υπολογίσαμε, περνάμε στον τελικό υπολογισμό:

tf-idf (term,docId) = (termCount/termsNorm) * log[totalDocs/(docsPerTerm+1)]

ακολουθώντας έτσι τους τύπους που αναλύσαμε στην ενότητα 2.3.2, σχετικά με το Vector Space Model. Είναι κατανοητό πως τα δεδομένα εξόδου θα περιλαμβάνουν το εκάστοτε ζεύγος του εξεταζόμενου **όρου** και **κάθε κειμένου** στο οποίο εμφανίζεται καθώς επίσης και την αντίστοιχη **τιμή tf-idf**. Σε αυτό το σημείο, επιλέξαμε να εξάγουμε και την **τιμή idf** για κάθε όρο ως ένα από τα στοιχεία του value. Η συγκεκριμένη κίνηση έγινε με σκοπό την ένταξη της συγκεκριμένης τιμής στον βελτιωμένο Inverted Index, του οποίου την κατασκευή θα αναλύσουμε στη συνέχεια. Όποτε, συνοπτικά η έξοδος της φάσης Reduce και κατ' επέκταση του τελευταίου βήματος υπολογισμού των tf-idf scores:

key	**value**
<term,docId>	<idf,tf-idf>

Το key είναι τύπου *PairOfStringInt* και η value είναι τύπου *PairOfWritables<FloatWritable,FloatWritable>*.

3.6 Κατασκευή του βελτιωμένου Inverted Index σε Map-Reduce

Στην ενότητα 3.4 παρακολουθήσαμε πώς υλοποιήθηκε ο Inverted Index σε λογική Map-Reduce, λαμβάνοντας υπόψιν όλες τις σελίδες που συλλέχθηκαν μέσω του Nutch. Σε εκείνη την πρώτη μορφή του Index, είχαμε συμπεριλάβει scores όπως τα *document frequency* και *term counts*. Η μορφή του Index μέχρι αυτό το σημείο, έχει την εξής μορφή:

key	**value**
<term>	<docFrequency, postingList(docNumber,termCounts)>

Στη συνέχεια της υλοποίησης μας, όπως είδαμε, υπολογίσαμε περισσότερο αντιπροσωπευτικά scores που καταδεικνύουν, για παράδειγμα, πόσο σπάνιος είναι ένας όρος μέσα σε μία συλλογή κειμένων (*idf score*) ή πόσο σημαντικός είναι ένας όρος μέσα σε ένα κείμενο της συλλογής (*tf-idf score*). Έτσι, προκύπτει η ανάγκη για την ύπαρξη μίας δομής η οποία θα περιλαμβάνει τα scores που υπολογίστηκαν και αυτή δεν είναι άλλη από μία βελτιωμένη έκδοση του Inverted Index.

Στην ενότητα 3.5.3 παρακολουθήσαμε τον υπολογισμό σε Map-Reduce των tf-idf scores. Η έξοδος της συγκεκριμένης Map-Reduce job, θα αποτελέσει τα δεδομένα εισόδου της Map-Reduce job που είναι υπεύθυνη για την κατασκευή του νέου βελτιωμένου Inverted Index:

Δεδομένα Εισόδου

Η είσοδος της παρούσας διεργασίας (κλάση **BuildInvIndexTfidf.java**) θα είναι:

key	**value**
<term,docId>	<idf,tf-idf>

Φάση Map

Στη συνάρτηση map(), αυτό που γίνεται είναι στην ουσία ο μετασχηματισμός των δεδομένων εισόδου σε διαφορετική μορφή ώστε τα δεδομένα να ομαδοποιηθούν με βάση τον εκάστοτε όρο, κάτι που εξυπηρετεί την κατασκευή του Index στη φάση reduce. Οπότε, τα δεδομένα εξόδου της map(), θα είναι:

key	**value**
<term>	<idf,(docId,tf-idf)>

Φάση Partitioning

Όπως και στην κατασκευή της πρώτης έκδοσης του Inverted Index σε Map-Reduce (βλ. Ενότητα 3.4), έτσι και εδώ επιλέγουμε τα δεδομένα να κατανεμηθούν με βάση τον προς εξέταση όρο. Δηλαδή, η επεξεργασία ενός όρου και όλων των τιμών <idf,(docId,tf-idf)> που ανταποκρίνονται σε εκείνον θα γίνεται από έναν reducer.

Φάση Reduce

Στη φάση Reduce, λαμβάνουμε ως key το <term> και ως value μία λίστα από τιμές listOf(<idf,

(docId,tf-idf)>). Για κάθε όρο, κατασκευάζεται μία posting list η οποία θα περιέχει όλα τα ζεύγη της μορφής (docId,tf-idf) που συμπεριλαμβάνονται στα δεδομένα εισόδου για το συγκεκριμένο όρο. Με αυτό τον τρόπο, γίνεται δυνατή η ταχεία προσπέλαση των κειμένων και των αντίστοιχων scores σε σχέση πάντα με τον όρο που εξετάζουμε. Στην τιμή εξόδου (value) επιλέγουμε να συμπεριλάβουμε και την τιμή idf, όπως είχαμε κάνει και στην 3η φάση υπολογισμού των tf-idf scores, κάτι που γίνεται σκόπιμα και θα φανεί χρήσιμο στον υπολογισμό της ομοιότητας συνημιτόνου (cosine similarity) μεταξύ των στοιχείων ενός ζεύγους <επερώτημα_χρήστη, κείμενο>, που θα εξετάσουμε στην επόμενη ενότητα. Κατ' αυτόν τον τρόπο, τα δεδομένα εξόδου από την συνάρτηση reduce() διαμορφώνονται ως εξής:

key	**value**
<term>	<idf, postingList(docId,tf-idf)>

Αξίζει να σημειωθεί ότι τα στοιχεία της posting list που αντιστοιχεί σε έναν όρο θα μπορούσαν να είναι ταξινομημένα ως προς την τιμή tf-idf. Αυτό, όμως, δεν αποτελεί κάτι που θα εξυπηρετούσε κάποιο σκοπό στη συνέχεια της παρούσας εργασίας, καθώς κινούμαστε στα πλαίσια του Vector Space Model. Έτσι, για ένα επερώτημα (query) θα ανακτηθούν οι τιμές tf-idf για **όλα** τα κείμενα των posting lists που αντιστοιχούν σε κάθε όρο του επερωτήματος, ανεξάρτητα από το αν έχουν ταξινομηθεί ή όχι. Στην επόμενη ενότητα θα εξετάσουμε τον τρόπο με τον οποίο εκμεταλλευόμαστε τον Inverted Index ώστε να εξάγουμε αποτελέσματα για το πόσο σχετικό μπορεί να είναι ένα query με ένα κείμενο, μέσω του μέτρου “Cosine Similarity”.

3.7 Υπολογισμός Ομοιότητας Συνημιτόνου (Cosine Similarity) σε Map-Reduce

Έχοντας υπολογίσει τα tf-idf scores για τα ζεύγη <term, document>, το στοιχείο που απομένει ώστε η μηχανή αναζήτησης να θεωρηθεί αποτελεσματική, είναι να μπορεί να απαντήσει στα επερωτήματα (queries) που θέτουν οι χρήστες και να είναι σε θέση να εξάγει τα περισσότερο σχετικά αποτελέσματα. Δηλαδή, για το εκάστοτε επερώτημα, να λαμβάνεται μία λίστα από σελίδες οι οποίες ανταποκρίνονται όσο το δυνατόν περισσότερο σε αυτό το οποίο αναζητάει ο χρήστης. Γίνεται λοιπόν κατανοητό ότι χρειάζεται να έχουμε στη διάθεσή μας κάποιο μέτρο του οποίου η τιμή θα αντικατοπτρίζει τη σχέση ανάμεσα σε ένα επερώτημα και στο εκάστοτε κείμενο της συλλογής μας (δηλαδή για κάθε ζεύγος <query, document>). Το μέτρο που χρησιμοποιήθηκε στην εργασία μας για αυτόν τον σκοπό, δεν είναι άλλο από την **Ομοιότητα Συνημιτόνου (Cosine Similarity).**

Ήδη από την ενότητα 2.3.2, έχουμε δει τον τύπο για το Cosine Similarity:

$$\frac{\sum_{i=1}^{N} q_i d_i}{\sqrt{\sum_{i=1}^{N} q_i^2}\sqrt{\sum_{i=1}^{N} d_i^2}}$$

με: $\mathbf{q_i}$ το βάρος του όρου i στο query,

$\mathbf{d_i}$ το βάρος του όρου i στο document και

N το συνολικό πλήθος των κειμένων της συλλογής

Στην παρούσα ενότητα θα παρουσιάσουμε τον τρόπο με τον οποίο υπολογίσαμε την Cosine Similarity με βάση το μοτίβο Map-Reduce:

- **Υπολογισμός L2 norm των κειμένων**

Λαμβάνοντας υπόψιν τον παραπάνω τύπο, επιλέξαμε να υπολογίσουμε ξεχωριστά τον όρο $\sqrt{\sum_{i=1}^{N} d_i^2}$

ο οποίος αποτελεί την κανονικοποιημένη κατά μήκος μορφή (length normalization – L_2 norm) του εκάστοτε κειμένου **d**, με βάταση τους όρους **i** που περιέχει. Ο συγκεκριμένος υπολογισμός έγινε σε μία Map-Reduce job (κλάση **Doc_LengthNorms.java**). Στην ουσία, ο σκοπός της διαδικασίας είναι να υπολογιστεί για κάθε κείμενο, η αντίστοιχη τιμή L_2 norm, η οποία παραμένει σταθερή για κάθε κείμενο, ανεξάρτητα με το επερωτήμα το οποίο μπορεί να εξετάζεται κάθε φορά.

Δεδομένα Εισόδου

Ως είσοδος στην παρούσα διεργασία, δίνονται τα δεδομένα εξόδου της 3ης φάσης του υπολογισμού των tf-idf scores. Δηλαδή:

key	**value**
<term,docId>	<idf,tf-idf>

Ο όρος $\mathbf{d_i}$, όπως αναφέραμε και πριν, αποτελεί το βάρος του όρου i στο κείμενο d. Εφόσον επιλέγουμε να χρησιμοποιήσουμε τα βάρη tf-idf, τότε το d_i θα είναι στην ουσία το tf-idf score μεταξύ του όρου i και του κειμένου d.

Φάση Map

Από τη συνάρτηση map(), εξάγονται για κάθε κείμενο d, όλες οι τιμές d_i^2, λαμβάνοντας υπόψιν όλους τους όρους i του κειμένου. Έτσι, η έξοδος της map():

key	**value**
<docId>	<$(\text{tf-idf})_i^2$>

Φάση Reduce

Στη φάση Reduce, τα ενδιάμεσα δεδομένα που εισέρχονται είναι ζευγάρια του τύπου (<docId>, <listOf(tf-idf_i^2)>). Έτσι, σε κάθε κείμενο αντιστοιχεί ένα πλήθος από τιμές tf-idf_i^2 , μία για κάθε διαφορετικό όρο του κειμένου. Οι τιμές αυτές αθροίζονται και στη συνέχεια, υπολογίζεται η ρίζα του αθροίσματος, ώστε να προκύψει η τιμή L_2 norm για το εκάστοτε κείμενο. Άρα, τα δεδομένα εξόδου από τη συνάρτηση reduce() είναι:

key	**value**
<docId>	<doc_Norm>

Στη συγκεκριμένη φάση επιλέγουμε να εισάγουμε τα παραπάνω ζεύγη σε **Hadoop Mapfiles** τύπου <IntWritable, FloatWritable>, έτσι ώστε στη συνέχεια (υπολογισμός Cosine Similarity), να είναι δυνατή η άμεση ανάκτηση της τιμής (L_2 norm) για το ζητούμενο κλειδί (document Id).

- **Υπολογισμός Cosine Similarity**

Έχοντας υπολογίσει τις L_2 norms για κάθε κείμενο της συλλογής, περνάμε στον υπολογισμό της ομοιότητας συνημιτόνου, δεδομένου ενός πλήθος επερωτημάτων.

Δεδομένα Εισόδου

Τα δεδομένα εισόδου σε αυτή τη φάση θα είναι τα δεδομένα που προέκυψαν από το βήμα της κατασκευής του βελτιωμένου Inverted Index με τις προσθήκες των scores idf και tf-idf, που αναλύθηκε στην ενότητα 3.6. Δηλαδή:

key	**value**
<term>	<idf, postingList(docId,tf-idf)>

Η τιμή idf, όπως αναφέραμε στην ενότητα 2.3.2, αποτελεί το βάρος qi του εκάστοτε όρου i σε ένα query. Παρατηρούμε, συνεπώς, ότι τα δεδομένα εισόδου, μαζί με τις L_2 norms από το προηγούμενο βήμα, συνιστούν τα scores που θα χρειαστούν στον υπολογισμό της Cosine Similarity, όπως μπορούμε να αντιληφθούμε από τον αντίστοιχο τύπο.

Φάση Map

Πριν αρχίσει η διαδικασία υπολογισμού των απαραίτητων τιμών, έλαβαν χώρα κάποιες προκαταρκτικές διαδικασίες, που αφορούν στο διάβασμα των επερωτημάτων αλλά και των τιμών L_2 norm των κειμένων. Στη συνάρτηση **setup()**, λοιπόν, που εκτελείται πριν από τη map(), αρχικά αποθηκεύονται στη μνήμη (σε δομή **HashMap<Integer,Float>**) οι τιμές L_2 norm, με κλειδί τα αντίστοιχα document Ids, χρησιμοποιώντας τα ζεύγη <docId, docNorm> τα οποία, όπως αναφέραμε παραπάνω, είναι αποθηκευμένα στο HDFS ως Hadoop Mapfiles. Ο σκοπός που έγινε αυτό είναι η ταχύτερη προσπέλαση των συγκεκριμένων τιμών από το να διαβάζονται από το HDFS. Στη συνέχεια, διαβάζονται τα επερωτήματα των χρηστών ,τα οποία βρίσκονται στο HDFS και αποθηκεύονται σε μία δομή τύπου **LinkedList<String>**, αφού πρώτα οι όροι που τα αποτελούν, υποστούν αφαίρεση stopwords και stemming.

Στη συνάρτηση map(), πρωτίστως διαβάζονται ένα-ένα τα επερωτήματα. Για κάθε όρο του επερωτήματος, εξετάζουμε αν είναι ίδιος με τον τρέχων όρο της εισόδου (δηλ. του Inverted Index), τον οποίο προσπελαύνει σε εκείνη τη φάση το πρόγραμμα Map-Reduce. Αν ναι, τότε διατρέχουμε την αντίστοιχη posting list για εκείνο τον όρο και για κάθε κείμενο που βρίσκεται στη λίστα, ανακτάμε την L_2 norm του καθώς και την τιμή tf-idf, ενώ έχουμε στη διάθεσή μας την τιμή idf (που βρίσκεται προσκολλημένη στον Index) για τον τρέχων όρο. Έπειτα, υπολογίζουμε για κάθε κείμενο που βρίσκεται στην posting list την τιμή **((term_idf*tf_idf)/docNorm)**, έστω *PartOfSum,* και την εισάγουμε σε μία νέα λίστα η οποία περιέχει ζεύγη της μορφής <term, PartOfSum>. **Για κάθε όρο** του επερωτήματος, θα εξάγονται δεδομένα τα οποία θα είναι της μορφής:

key	**value**
<query>	<idf, list(PartOfSum)>

Για παράδειγμα, για το επερώτημα “European Football Club”, τα αντίστοιχα ενδιάμεσα δεδομένα θα είναι:

key	**value**
<“European Football Club”>	<$idf_{”European”}$, list$(PartOfSum)_{”European”}$>
<“European Football Club”>	<$idf_{”Football”}$, list$(PartOfSum)_{”Football”}$>
<“European Football Club”>	<$idf_{”Club”}$, list$(PartOfSum)_{”Club”}$>

Για ποιο λόγο όμως επιλέξαμε να εξαχθούν δεδομένα αυτής της μορφής; Αν κοιτάξουμε ξανά τον τύπο $\frac{\sum_{i=1}^{N} q_i d_i}{\sqrt{\sum_{i=1}^{N} q_i^2}\sqrt{\sum_{i=1}^{N} d_i^2}}$, παρατηρούμε ότι τα δεδομένα που εξάγονται από τη φάση Map είναι οι όροι $\frac{q_i d_i}{\sqrt{\sum_{i=1}^{N} d_i^2}}$, οι οποίοι αν αθροιστούν, παράγουν μερικώς τον παραπάνω τύπο (εκτός από την κανονικοποιημένη μορφή των scores του εκάστοτε επερωτήματος q για τους όρους i που εμπεριέχει). Έτσι, στη συνέχεια θα είμαστε σε θέση να υπολογίσουμε πλήρως την ομοιότητα συνημιτόνου μεταξύ ενός επερωτήματος και των σχετικών κειμένων, κάνοντας χρήση και του score idf, το οποίο έχει επιλεγεί να είναι το score που αντιπροσωπεύει την τιμή q_i. Δηλαδή, για κάθε όρο i του επερωτήματος q, η ποσότητα q_i θα αποτελεί το idf score του όρου i.

<u>Φάση Reduce</u>

Ο αντικειμενικός σκοπός της φάσης Reduce, είναι σαφώς η εξαγωγή των περισσότερο σχετικών αποτελεσμάτων δεδομένου του εκάστοτε επερωτήματος. Τα δεδομένα εισόδου σε αυτή τη φάση θα είναι ζεύγη (<query>, <listOf(idf, list(docId,PartOfSum))>). Για κάθε query, διατρέχεται η λίστα με τις αντίστοιχες τιμές και σε κάθε τέτοια επανάληψη, αθροίζονται οι τιμές idf^2, που προκύπτουν στην ουσία από το idf score κάθε όρου του query. Παράλληλα, διατρέχεται η λίστα με τα ζεύγη (docId,PartOfSum) , όπου για κάθε documentId, αποθηκεύεται η αντίστοιχη τιμή PartOfSum που είχε υπολογιστεί στην φάση Map. Αν κάποιο κείμενο εμφανίζεται στις λίστες περισσότερων του ενός όρου του επερωτήματος, τότε οι τιμές PartOfSum αθροίζονται και ενημερώνεται η τιμή του συγκεκριμένου document Id. Για αυτό το λόγο χρησιμοποιείται ένα HashMap<Integer,Float>, με κλειδί docId και τιμή το αντίστοιχο άθροισμα. Με αυτόν τον τρόπο, καταφέρνουμε να υπολογίσουμε τον όρο $\frac{\sum_{i=1}^{N} q_i d_i}{\sqrt{\sum_{i=1}^{N} d_i^2}}$. Αυτό που μένει για να ολοκληρωθεί ο υπολογισμός της Cosine Similarity είναι να διαιρέσουμε την παραπάνω τιμή με το $\sqrt{\sum_{i=1}^{N} q_i^2}$, το

οποίο προκύπτει λαμβάνοντας τη ρίζα του αθροίσματος των τιμών idf^2, των οποίων τον υπολογισμό εξηγήσαμε παραπάνω. Έτσι, κατασκευάζεται μία νέα λίστα η οποία περιλαμβάνει για κάθε κείμενο σχετικό με το επερώτημα, την αντίστοιχη τιμή Cosine Similarity, δηλαδή ζεύγη του τύπου (<docId,CosineSimilarity>).

Έχοντας στη διάθεσή μας πλέον τα παραπάνω δεδομένα, αυτό που μας μένει είναι να εξάγουμε τα περισσότερο σχετικά αποτελέσματα για το κάθε επερώτημα. Οπότε, η λίστα που περιλαμβάνει τις τιμές Cosine Similarity για κάθε κείμενο, θα πρέπει να **ταξινομηθεί** με φθίνουσα σειρά. Από τις παραπάνω τιμές, θεωρήσαμε αρκετές τις **1000** πιο υψηλές, τις οποίες και κρατήσαμε. Οι τιμές εκείνες, θα συνδυαστούν στη συνέχεια με τις τιμές pagerank των αντίστοιχων κειμένων, όπως θα δείξουμε στο κεφάλαιο 4. Ένα τελευταίο στοιχείο το οποίο πρέπει να αναφέρουμε σε αυτό το σημείο είναι ότι οι τιμές υπέστησαν κανονικοποίηση πριν την αποθήκευσή τους στο HDFS. Η κανονικοποιημένη μορφή NSi προκύπτει ως εξής:

NSi = (Si – Smin) / (Smax – Smin), όπου Si το εξεταζόμενο score, Smin η ελάχιστη τιμή και Smax η μέγιστη τιμή. Ο λόγος που γίνεται η κανονικοποίηση, είναι ώστε να γίνει εφικτός ο συνδυασμός της Cosine Similarity με τις τιμές του PageRank που θα προκύψουν για κάθε κείμενο, όπως θα δούμε στο επόμενο κεφάλαιο.

Τελικώς, τα δεδομένα εξόδου της φάσης Reduce αυτού του βήματος θα είναι:

key	**value**
<query>	<sortedListTop1000(docId,CosSimNormalized)>

Το κλειδί είναι τύπου *Text* και η τιμή τύπου *ArrayListWritable<PairOfIntFloat>*.

4. Personalized PageRank και Συνδυασμός Μεθόδων Scoring στο Hadoop Map-Reduce

Το κεφάλαιο 4 περιλαμβάνει τον τρόπο με τον οποίο υλοποιήθηκε ο αλγόριθμος Personalized PageRank σε Map-Reduce, έχοντας λάβει υπόψιν προτιμήσεις των χρηστών για συγκεκριμένες ιστοσελίδες. Επίσης, παρουσιάζεται ο συνδυασμός της συγκεκριμένης μεθόδου αξιολόγησης ιστοσελίδων με τη μέθοδο Cosine Similarity, που είδαμε στο προηγούμενο κεφάλαιο. Ο σκοπός της ενοποίησης των scores είναι η εξαγωγή των περισσότερο σχετικών αποτελεσμάτων για τον εκάστοτε χρήστη, με βάση τα ενδιαφέροντά του και φυσικά με δεδομένα τα επερωτήματα που εκτελούνται.

4.1 Ανάλυση των απαιτήσεων

Στο κεφάλαιο 3 παρουσιάστηκε ο τρόπος με τον οποίο εκμεταλλευτήκαμε το λεκτικό περιεχόμενο των σελίδων ώστε να κατασκευάσουμε χρήσιμες δομές και να υπολογίσουμε scores που είναι απαραίτητα για την εξαγωγή των τελικών αποτελεσμάτων, ενώ όλη η διαδικασία υλοποιήθηκε στο Hadoop. Από την άλλη πλευρά, το Nutch μας παρέχει, επίσης, τον webgraph, που περιλαμβάνει όλους τους συνδέσμους μεταξύ των σελίδων της συλλογής. Το επόμενο βήμα, λοιπόν, είναι να προβούμε σε μία ανάλυση του τρόπου με τον οποίο οι σελίδες συνδέονται μεταξύ τους με σκοπό να εξαχθούν scores που είναι ανεξάρτητα με το λεκτικό περιεχόμενο.

Ο διάσημος αλγόριθμος PageRank της Google βασίζεται στην ανάλυση του webgraph, οπότε κατά την εκτέλεση του αλγορίθμου, ανατίθενται τιμές στις σελίδες-κόμβους. Οι τιμές εκείνες αποτελούν στην ουσία ένα μέτρο κύρους για την εκάστοτε σελίδα και αντιπροσωπεύουν το πόσο σημαντική είναι ανάμεσα στις υπόλοιπες. Έτσι και στο σύστημα μας, με βάση τη δομή του γράφου και στηριζόμενοι στη βιβλιοθήκη cloud9 [15] όπου υλοποιήσαμε τις κατάλληλες προσθήκες, εκτελούμε τον αλγόριθμο PageRank στο Hadoop.

Λόγω όμως του ότι οι χρήστες μίας μηχανής αναζήτησης έχουν διαφορετικά πεδία προτιμήσεων, είναι πιθανό να ενδιαφέρονται για συγκεκριμένου είδους αποτελέσματα. Επομένως, η υλοποίηση ενός συστήματος που θα λαμβάνει τις προτιμήσεις των χρηστών, κινείται προς αυτή την ιδέα. Γι' αυτό και προχωρήσαμε στον υπολογισμό του Εξατομικευμένου (Personalized) PageRank, διατηρώντας τη λογική του PageRank, δίνοντας όμως ταυτόχρονα παραπάνω βάρη στις σελίδες προτίμησης των χρηστών.

Οι βασικές απαιτήσεις για την προ-επεξεργασία των δεδομένων και τις προσθήκες που

χρειάζονται να εισαχθούν στον αλγόριθμο PageRank ώστε να λειτουργήσει εξατομικευμένα είναι οι εξής:

- Η μετατροπή του γράφου διαδικτύου σε μορφή κατάλληλη για είσοδο σε μία Map-Reduce job. Επίσης, μία σημαντική πρόκληση ήταν η μετατροπή όλων των urls του γράφου σε ids, με βάση την αντιστοίχηση που έχουμε δείξει ήδη στην ενότητα 3.3. Η συγκεκριμένη διαδικασία χρειάστηκε ειδική αντιμετώπιση όπως θα δείξουμε παρακάτω, καθώς ο όγκος των urls και ids που είναι αποθηκευμένα στο αντίστοιχο Mapfile, αρχικά ήταν απαγορευτικά μεγάλος ώστε να γίνει η προσπέλαση και ανάκτηση όλης της απαιτούμενης πληροφορίας.
- Η προσθήκη του κριτηρίου σύγκλισης στον αλγόριθμο PageRank. Η προυπάρχουσα υλοποίηση [15] δεν έχει λάβει υπόψιν το κριτήριο σύγκλισης, κάτι που χρειάζεται να γίνει ώστε οι επαναλήψεις του αλγορίθμου να μην είναι συγκεκριμένου πλήθος, αλλά να διακόπτονται όταν τηρείται η συνθήκη του κριτηρίου.
- Η προσθήκη της εξατομίκευσης στον αλγόριθμο PageRank. Απαιτείται η κατανομή βαρών στις σελίδες προτίμησης του εκάστοτε χρήστη και στη συνέχεια ο υπολογισμός του Personalized PageRank με βάση εκείνες τις προτιμήσεις.

Από μόνες τους οι τιμές του Personalized PageRank δεν μπορούν να προσφέρουν αποτελέσματα στις αναζητήσεις των χρηστών. Γι' αυτό, είναι απαραίτητο να συνδυαστούν με ένα score που λαμβάνει υπόψιν το κείμενο των σελίδων και στην περίπτωση μας δεν είναι άλλο από την Ομοιότητα Συνημιτόνου (Cosine Similarity). Δημιουργείται με αυτόν τον τρόπο ένα **ενοποιημένο score** που επηρεάζεται α)από το πόσο όμοιο είναι ένα επερώτημα με το λεκτικό περιεχόμενο και β) από τις προτιμήσεις του εκάστοτε χρήστη.

Μπορεί να γίνει εύκολα αντιληπτό ότι για τους παραπάνω υπολογισμούς, γίνεται η επεξεργασία και η διαχείριση μεγάλου όγκου δεδομένων που βρίσκονται στο HDFS, η οποία είναι δυνατή λόγω της χρήσης του μοτίβου Map-Reduce, εκμεταλλευόμενοι την αποδοτικότητα του Hadoop.

4.2 Κατασκευή γράφου διαδικτύου (web graph)

Στην ενότητα 3.2 παρουσιάστηκε η επεξεργασία που υπέστησαν τα δεδομένα αμέσως μετά τη φάση του crawling και πώς τα κείμενα της συλλογής μας απέκτησαν την κατάλληλη μορφή ώστε να δοθούν ως είσοδος για τη δημιουργία του Inverted Index. Εκτός όμως από τα κείμενα (documents) και το λεκτικό τους περιεχόμενο, είναι σημαντικό να έχουμε στη διάθεσή μας το γράφο που απεικονίζει τις συνδέσεις μεταξύ των σελίδων. Όπως και με τα κείμενα και το περιεχόμενό τους, αντίστοιχη διαδικασία ακολουθείται για τη μετατροπή του γράφου σε μορφή κειμένου, εκτελώντας την παρακάτω εντολή του Nutch:

- Εντολή readlinkdb: *bin/nutch readlinkdb -dump user/gfragiadoulakis/crawlLarge/linkdb user/gfragiadoulakis/dumpOutlinks*

  Η πληροφορία σχετικά με τη σύνδεση των σελίδων της συλλογής, που βρίσκεται στο directory *linkdb,* μετατρέπεται σε αναγνώσιμη μορφή στο directory *dumpOutlinks.*

Ένα παράδειγμα μίας εγγραφής που απεικονίζει τους εξω-συνδέσμους από μία σελίδα, χρησιμοποιώντας την εντολή *readlinkdb*, απεικονίζεται στην εικόνα 3.1.

Recno:: 103

URL:: http://abcnews.go.com/Health/video/tiny-baby-home-hospital-15404118

ParseData::

Version: 5

Status: success(1,0)

Title: Tiny Baby Going Home From Hospital | Video - ABC News

Outlinks: 33

outlink: toUrl: http://abcnews.go.com/ anchor: ABC News

outlink: toUrl: http://abcnews.go.com/Video/ anchor: News Videos

outlink: toUrl: http://www.facebook.com/abcnews anchor: Facebook

outlink: toUrl: http://www.espn.com anchor: ESPN

outlink: toUrl: http://www.abc.com anchor: ABC

.....

.....

Content Metadata: ETag="8075e59c18d8cc1:1026" Content-Length=73252 Last-Modified=Sat, 21 Jan 2012 08:42:31 GMT _fst_=33 nutch.segment.name=20120121085127 Connection=close Server=Microsoft-IIS/6.0 X-Powered-By=ASP.NET Cache-Control=max-age=300 Cache-Expires=Sat, 21 Jan 2012 08:45:01 GMT nutch.content.digest=963e1847d2f8113efcc595be3f42a2e9 Date=Sat, 21 Jan 2012 08:58:14 GMT P3P=CP="CAO DSP COR CURa ADMa DEVa TAIa PSAa PSDa IVAi IVDi CONi OUR SAMo OTRo BUS PHY ONL UNI PUR COM NAV INT DEM CNT STA PRE" nutch.crawl.score=1.4087407E-5 Accept-Ranges=bytes Content-Type=text/html;charset=UTF-8 From=abc01

Parse Metadata: CharEncodingForConversion=utf-8 caching.forbidden=content OriginalCharEncoding=utf-8

Εικόνα 4.1: Το περιεχόμενο μίας εγγραφής μετά την εκτέλεση της εντολής "readlinkdb"

Λόγω του ότι η παραπάνω μορφή δεν είναι σύμφωνη με τις προσταγές του μοτίβου Map-Reduce, κατασκευάσαμε το γράφο που απεικονίζει τους συνδέσμους μεταξύ των σελίδων, μετατρέποντας την παραπάνω πληροφορία και δημιουργώντας το αρχείο *nutchWebGraph* (κλάση **WebgraphExtractor.java**), που έχει τη μορφή:

<url1> <outlink1> <outlink2> <outlink3> <outlink4> ... <outlinkN>

<url2> <outlink1> <outlink2> <outlink3> <outlink4> ... <outlinkM> κοκ.

Αντικατάσταση των urls με ακέραια ids

Όπως είδαμε και στην ενότητα 3.3, τα urls των σελίδων μετατράπηκαν σε ids, βάσει του Hadoop Mapfile που κατασκευάσαμε και στο οποίο κάθε url αντιστοιχεί σε ένα μοναδικό id. Έτσι και σε αυτό το σημείο, εκτελώντας μία Map-Reduce job (κλάση **ConvertWebGraphUrlsToIds.java**) που λαμβάνει ως είσοδο το αρχείο *nutchWebGraph,* τα urls μετατρέπονται σε ids και δημιουργείται το αρχείο *nutchWebgraph_ids*. Όταν κάποιο root url δεν περιλαμβάνεται στο Mapfile, τότε εκείνο το url και οι εξω-σύνδεσμοι του δεν συμπεριλαμβάνονται στο νέο αρχείο. Επίσης, αν κάποιος εξω-σύνδεσμος δε περιλαμβάνεται στο Mapfile αλλά το αντίστοιχο root url περιλαμβάνεται, τότε ο εξω-σύνδεσμος αντικαθίσταται με κενό. Η συγκεκριμένη διαδικασία απαιτεί πολλαπλές προσβάσεις στο Mapfile, ώστε να ανακτηθούν τα ids, κάτι που είναι ιδιαίτερα χρονοβόρο. Γι' αυτό το λόγο, αρχικά κατακερματίστηκε το αρχείο Mapfile και στη συνέχεια το αρχείο *nutchWebgraph_ids* δημιουργήθηκε **τμηματικά** σε επαναλήψεις, τόσες όσα και τα μέρη του Mapfile. Είναι εμφανές ότι η διαφορά στο μέγεθος των αρχείων πριν και μετά τις μετατροπές (ειδικά του *nutchWebGraph* που περιλαμβάνει μόνο urls, αλλά και του *nutchDocsEnglish* που είδαμε πριν) είναι μεγάλη.

Έχουμε σχεδόν τελειώσει με την κατασκευή του γράφου ώστε να είναι στην κατάλληλη μορφή για να ξεκινήσει η διαδικασία υλοποίησης του Personalized PageRank στο Hadoop. Μετά και τις μετατροπές των urls σε ids, έχουμε καταλήξει σε ένα αρχείο του οποίου κάθε γραμμή είναι της μορφής:

<url_id> <outlink1_id> <outlink2_id> <outlink3_id> <outlink4_id> ... <outlinkN_id>

Παρατηρήσαμε ότι υπάρχει η πιθανότητα κάποια url_ids να βρίσκονται στο γράφο ως outlinks κάποιων άλλων url_ids, όμως τα ίδια να μην έχουν θέση στο γράφο ως root urls. Έτσι, θεωρήσαμε ότι όλα τα url_ids θα πρέπει να αποτελούν ταυτόχρονα και root urls, ακόμα κι αν έχουν μηδενικό πλήθος εξω-συνδέσμων. Άλλωστε, η υλοποίηση που ακολουθεί λαμβάνει υπόψιν τους κόμβους που δεν έχουν εξω-συνδέσμους (dangling nodes), καθώς το *pagerank mass* εκείνων των κόμβων διαμοιράζεται ισόποσα σε όλους τους κόμβους του γράφου, όπως θα δούμε αναλυτικότερα στη συνέχεια. Η παραπάνω διαδικασία έγινε μέσω της κλάσης **BuildCompleteGraph.java**:

Δεδομένα Εισόδου

Ως είσοδος στη συγκεκριμένη Map-Reduce job δίνεται το αρχείο nutchWebgraph_ids.

Φάση Map

Η διαδικασία που ακολουθείται στη συνάρτηση map() είναι να εξάγονται ως κλειδιά των ενδιάμεσων δεδομένων όλα τα url_ids, ανεξάρτητα από το αν είναι root urls ή όχι. Όταν εξάγεται κάποιο outlink, τότε απλά το κλειδί αποτελείται από το συγκεκριμένο url_id, ενώ η αντίστοιχη τιμή είναι κενή. Όταν εξάγεται ένα root url, τότε εκείνο το url_id αποτελεί το κλειδί, ενώ η τιμή που το συνοδεύει είναι η λίστα από τα ids των εξω-συνδέσμων του (λίστα γειτνίασης – adjacency list). Έτσι, στην έξοδο, θα έχουμε δεδομένα της μορφής:

key	**value**
<url_id>	<listOfOutlinks>

**Η τιμή listOfOutlinks θα είναι κενή εάν το κλειδί αποτελεί id κάποιου εξω-συνδέσμου (outlink).*

Φάση Reduce

Τα δεδομένα που εισέρχονται στη φάση Reduce είναι ζεύγη της μορφής <url_id,listOfValues>. Η παράμετρος *listOfValues* συμβολίζει τις τιμές που εξάγονται μαζί με ένα κλειδί. Εάν το url_id που απεικονίζει το εκάστοτε κλειδί έχει εξω-συνδέσμους, τότε μία από τις τιμές της λίστας *listOfValues* θα είναι στην ουσία η λίστα με όλους τους εξω-συνδέσμους του. Οι υπόλοιπες τιμές θα είναι κενές, μία για κάθε φορά που το συγκεκριμένο url_id εξήχθη ως εξω-σύνδεσμος κάποιου άλλου url_id. Εάν κάποιο κλειδί συνοδεύεται από μία λίστα με **μόνο** κενές τιμές, αυτό σημαίνει ότι δεν υπήρχε ως root url μέχρι τώρα στο γράφο. Γι' αυτό, το συγκεκριμένο url_id εισάγεται στο γράφο ως root url που δεν περιέχει εξω-συνδέσμους. Με αυτόν τον τρόπο, εξασφαλίζουμε ότι κάθε url_id του γράφου θα υπάρχει και ως root url, ακόμα και αν εκείνο δεν περιέχει συνδέσμους προς άλλα urls. Έτσι, τα δεδομένα εξόδου από τη φάση Reduce θα είναι:

key	**value**
<url_id>	<listOfOutlinks>

όπου η τιμή *listOfOutlinks* θα αποτελείται από όλους τους εξω-συνδέσμους του *url_id* ή θα είναι κενή εάν το *url_id* δεν έχει εξω-συνδέσμους.

Ύστερα από τις παραπάνω διαδικασίες, οι οποίες μας οδήγησαν στην κατασκευή του γράφου διαδικτύου (web graph – αρχείο *nutchCompleteWebgraph*), είμαστε έτοιμοι να περάσουμε στο επόμενο βήμα, το οποίο δεν είναι άλλο από την υλοποίηση του πρωτότυπου (original) και του Εξατομικευμένου (Personalized) PageRank σε Map-Reduce.

4.3 Προϋπάρχουσες Υλοποιήσεις – Original PageRank

Για τον υπολογισμό του PageRank των σελίδων της συλλογής μας, στηριχθήκαμε στη βιβλιοθήκη **cloud9** [24], η οποία παρέχει διαφορετικές υλοποιήσεις του αλγορίθμου [15]. Στην εργασία μας, χρησιμοποιήθηκε η μία από αυτές. Στις υποενότητες που ακολουθούν θα παρουσιαστούν οι κλάσεις της Java που χρησιμοποιήθηκαν από τη συγκεκριμένη βιβλιοθήκη για τον υπολογισμό του PageRank των κόμβων του γράφου που δείξαμε πώς κατασκευάστηκε στην προηγούμενη ενότητα.

4.3.1 Κατασκευή δομημένης αναπαράστασης του γράφου στο Hadoop

Έχοντας στη διάθεσή μας το γράφο διαδικτύου, του οποίου την κατασκευή παρακολουθήσαμε στην προηγούμενη ενότητα, η βιβλιοθήκη cloud9 μας παρέχει τη δυνατότητα να κατασκευάσουμε μία πιο ολοκληρωμένη μορφή του γράφου, συμπεριλαμβάνοντας περισσότερα στοιχεία για τον κάθε κόμβο, όπως η τρέχουσα τιμή του PageRank για τον κόμβο αυτό. Σκοπός της συγκεκριμένης υλοποίησης είναι η δομημένη αναπαράσταση του γράφου διαδικτύου, καθώς η μορφή που έχουμε δει μέχρι τώρα είναι απλή μορφή κειμένου (αρχείο *nutchCompleteWebgraph*) όπου κάθε γραμμή περιέχει ένα ή περισσότερα url_ids, με το πρώτο από αυτά να αποτελεί το εκάστοτε root url_id και τα υπόλοιπα τους εξω-συνδέσμους του.

Για να καταστεί δυνατή η δομημένη αναπαράσταση του γράφου, χρησιμοποιείται η κλάση **PageRankNode.java**, η οποία αποτελεί την δομή που απεικονίζει ένα κόμβο του διαδικτύου μαζί

με τους εξω-συνδέσμους τους, καθώς και άλλη πρόσθετη πληροφορία, όπως θα δούμε παρακάτω.

Οι μεταβλητές που περιλαμβάνει η συγκεκριμένη κλάση είναι οι εξής:

- **type (Type)** : Η μεταβλητή type καθορίζει τον τύπο του κόμβου, ο οποίος μπορεί να είναι *Complete* (περιλαμβάνει την τιμή του pagerank και τη λίστα γειτνίασης του κόμβου), *Mass* (περιλαμβάνει μόνο την τιμή του pagerank) ή *Structure* (περιλαμβάνει μόνο τη λίστα γειτνίασης). Θα εξηγήσουμε στη συνέχεια (ενότητα 4.3.2) για ποιο λόγο χρησιμοποιείται η παρούσα μεταβλητή.
- **nodeid (int)** : Η μεταβλητή nodeid απεικονίζει το μοναδικό ακέραιο id που αντιστοιχεί σε κάθε url της συλλογής.
- **pagerank (float)** : Η μεταβλητή pagerank αναπαριστά την τιμή του pagerank για τον τρέχον κόμβο.
- **adjacencyList (ArrayListOfIntsWritable)** : Η adjacencyList απεικονίζει τη λίστα γειτνίασης του τρέχοντος κόμβου, που περιλαμβάνει όλα τα ακέραια url_ids των εξω-συνδέσμων του τρέχοντος url.

Με βάση λοιπόν την παραπάνω δομή για έναν κόμβο του γράφου, προχωράμε στην ανάλυση της Map-Reduce job (κλάση **BuildPageRankRecords.java**) που είναι υπεύθυνη για την αναπαράσταση του γράφου στην αντίστοιχη μορφή.

Δεδομένα Εισόδου

Τα δεδομένα εισόδου της παρούσας job θα είναι το αρχείο *nutchCompleteWebgraph*, που περιλαμβάνει σε μορφή κειμένου το γράφου διαδικτύου που έχουμε κατασκευάσει.

Φάση Map

Για κάθε γραμμή του αρχείου εισόδου, δημιουργείται ένα νέο αντικείμενο της κλάσης PageRankNode.java. Η τιμή της μεταβλητής type αρχικοποιείται σε *Complete*, ενώ η τιμή pagerank αρχικοποιείται σε 1/numOfNodes, όπου numOfNodes το πλήθος των κόμβων του γράφου. Έτσι, όλοι οι κόμβοι λαμβάνουν αρχικά την ίδια τιμή pagerank. Επίσης, στην μεταβλητή nodeid ανατίθεται η τιμή του id του πρώτου στοιχείου της γραμμής εισόδου και τέλος, η μεταβλητή adjacencyList αρχικοποιείται με τη λίστα των ακέραιων ids της γραμμής, εκτός του πρώτου που αποτελεί το root url. Η adjacencyList μπορεί να είναι κενή στην περίπτωση που κάποιος κόμβος του γράφου (dangling node) δεν περιέχει εξερχόμενες ακμές προς άλλους κόμβους, δηλαδή δεν έχει εξω-συνδέσμους.

Δεδομένα Εξόδου

Τα δεδομένα εξόδου από τη φάση Map θα είναι και τα δεδομένα εξόδου της Map-Reduce job, καθώς **δεν** γίνεται χρήση της συνάρτησης reduce(). Έτσι, η έξοδος θα έχει τη μορφή:

key	**value**
<node_id>	<node(*Complete*)>

όπου το κλειδί είναι τύπου IntWritable και η τιμή τύπου PageRankNode. Τα δεδομένα αποθηκεύονται σε μορφή SequenceFileOutputFormat στο HDFS.

Κατάτμηση του γράφου σε διαμερίσεις στο HDFS

Το επόμενο βήμα της προϋπάρχουσας υλοποίησης είναι ο ο κατακερματισμός του γράφου σε διαμερίσεις στο κατανεμημένο σύστημα αρχείων του Hadoop. Η συγκεκριμένη λειτουργία επιτελείται από μία Map-Reduce job (κλάση **PartitionGraph.java**), όπου δε γίνεται τίποτα άλλο από την εκτέλεση τόσων διεργασιών reduce (reduce tasks) όσες θέλουμε να είναι και οι διαμερίσεις του γράφου στο HDFS. Τα **δεδομένα εισόδου** της παραπάνω job είναι τα δεδομένα που εξήχθησαν από τη προηγούμενη, δηλαδή η δομημένη αναπαράσταση του γράφου, όπως εξηγήσαμε πιο πάνω. Η **έξοδος** είναι τα ίδια δεδομένα με τη μόνη διαφορά ότι είναι κατακερματισμένα.

Έτσι, έχουμε φτάσει στο σημείο να έχουμε στη διάθεση μας το γράφο σε μία μορφή που διευκολύνει την υλοποίηση του αλγορίθμου PageRank με τρόπο αποδοτικό και δομημένο, κάτι που θα δούμε στην παρακάτω ενότητα.

4.3.2 Υλοποίηση του αλγορίθμου PageRank στο Hadoop

Με βάση τις παραπάνω προκαταρκτικές διαδικασίες, είμαστε έτοιμοι να προχωρήσουμε στην ανάλυση της υλοποίησης του πρωτότυπου αλγορίθμου PageRank [15], όπως μας δίνεται από τη βιβλιοθήκη cloud9.

Η συγκεκριμένη βιβλιοθήκη περιλαμβάνει διάφορες τεχνικές υλοποίησης του PageRank στο Hadoop, κυρίως με σκοπό τη βελτίωση της αποδοτικότητας του αλγορίθμου όσον αφορά το χρόνο εκτέλεσης των επαναλήψεων. Εμείς χρησιμοποιήσαμε την τεχνική που προτάθηκε στο [1] και υλοποιείται στο [15], την οποία και θα περιγράψουμε παρακάτω.

Όπως έχουμε δει σε προηγούμενες ενότητες, ο αλγόριθμος PageRank εκτελείται επαναληπτικά έως ότου ισχύσει ένα κριτήριο σύγκλισης (ενότητα 2.4.1) ή μέχρι έναν συγκεκριμένο αριθμό επαναλήψεων:

- Αριθμός επαναλήψεων: $\cong \frac{-n}{\log_{10} d}$, Κριτήριο σύγκλισης: $\sum \left| r_{k+1}(p_i) - r_k(p_i) \right| < \varepsilon$

με **ε** $=10^{-n}$ και

n: αριθμός σημαντικών ψηφίων.

Η παρούσα προϋπάρχουσα υλοποίηση **δε χρησιμοποιεί** κάποιο τέτοιο κριτήριο, παρά μόνο έναν αριθμό επαναλήψεων ορισμένο από τον προγραμματιστή, με βάση πάντα το πλήθος των σημαντικών ψηφίων. Γι' αυτό το λόγο, προσθέσαμε στην υλοποίηση αυτή το κριτήριο σύγκλισης, όπως θα δούμε στην ενότητα 4.4.1. Έτσι, επιλέξαμε το κριτήριο σύγκλισης να ελέγχεται σε κάθε 10 επαναλήψεις του αλγορίθμου, ενώ επιλέξαμε το πλήθος **n** των σημαντικών ψηφίων να είναι ίσο με 5. Αυτό σημαίνει ότι η τιμή **ε** θα είναι ίση με 0.00001 και ο αριθμός των επαναλήψεων που θα εκτελεστούν, εάν δεν ισχύσει το κριτήριο σύγκλισης, θα είναι 70.

Στην υλοποίηση που εξετάζουμε, κάθε επανάληψη είναι χωρισμένη σε δύο βήματα και ως εκ τούτου, σε δύο Map-Reduce jobs.

Το **πρώτο** βήμα κάθε επανάληψης του αλγορίθμου είναι υπεύθυνο για τον υπολογισμό της τιμής pagerank κάθε κόμβου, χωρίς όμως να λαμβάνεται υπόψιν το άθροισμα των τιμών pagerank των κόμβων με κενή λίστα γειτνίασης (dangling nodes). Επίσης, δεν υπολογίζεται η τιμή που προκύπτει από την τυχαία μεταπήδηση (random jump) σε κάποιο κόμβο του γράφου. Για τα δύο αυτά στοιχεία και τον πλήρη υπολογισμό των τιμών pagerank των κόμβων του γράφου, υπεύθυνο

είναι το **δεύτερο** βήμα κάθε επανάληψης.

Η κλάση **RunPageRankBasic.java** εμπεριέχει και τις δύο Map-Reduce jobs, που αντιστοιχούν στα δύο βήματα κάθε επανάληψης:

ΒΗΜΑ 1

Δεδομένα Εισόδου

key	**value**
<node_id>	<node(*Complete*)>

Φάση Map

Κατά τη φάση Map, εξάγονται παράλληλα η δομή του κόμβου καθώς επίσης και οι κόμβοι της λίστας γειτνίασης μαζί με το μερίδιο pagerank που τους αντιστοιχεί. Εάν η λίστα γειτνίασης του κόμβου είναι κενή, τότε από τη φάση Map εξάγεται μόνο η δομή του τρέχοντος κόμβου. Τα ενδιάμεσα δεδομένα από τη φάση Map στη Reduce θα έχουν την παρακάτω μορφή:

key	**value**
<node_id>	<node(*Structure* or *Mass*)>

όπου η τιμή node είναι τύπου PageRankNode και αποτελεί είτε δεδομένα που αφορούν τη δομή του κόμβου είτε δεδομένα σχετικά με τους κόμβους της λίστας γειτνίασής του και την τιμή pagerank που τους αναλογεί.

Φάση Reduce

Στη φάση Reduce, όλες οι μερικές συνεισφορές του pagerank αθροίζονται ώστε να ενημερώσουν τους κόμβους με τις νέες τιμές [3]. Τα δεδομένα εξόδου είναι:

key	**value**
<node_id>	<node(*Complete*)>

Μετά το πέρας του πρώτου βήματος, υπολογίζεται πόσο μερίδιο pagerank "χάθηκε" στους κόμβους με κενή λίστα γειτνίασης (dangling nodes) και αυτή η τιμή αποθηκεύεται στο HDFS, ώστε να χρησιμοποιηθεί στο 2ο βήμα της επανάληψης.

ΒΗΜΑ 2

Το δεύτερο βήμα είναι υπεύθυνο για τον διαμοιρασμό του μεριδίου που χάθηκε στους "dangling" κόμβους, σε όλους τους κόμβους του γράφου, καθώς επίσης και για τον υπολογισμό του ποσού που αντιστοιχεί στην τυχαία μεταπήδηση σε οποιοδήποτε κόμβο, οπότε και για τον τελικό υπολογισμό της τιμής pagerank κάθε κόμβου του γράφου για την τρέχουσα επανάληψη.

Δεδομένα Εισόδου

Ως είσοδος στην παρούσα Map-Reduce job δίνονται τα δεδομένα εξόδου από το προηγούμενο βήμα.

Φάση Map

Κατά τη φάση Map, η τιμή pagerank του εκάστοτε κόμβου ενημερώνεται και αποθηκεύεται στο HDFS. Δεδομένου ότι στο δεύτερο βήμα δεν υπάρχει φάση Reduce, τα δεδομένα εξόδου της Map θα είναι και τα τελικά δεδομένα εξόδου της επανάληψης:

key	**value**
<node_id>	<node(*Complete*)>

4.4 Πρόσθετες Υλοποιήσεις – Personalized PageRank

Στην παρούσα ενότητα θα αναλυθούν οι πρόσθετες υλοποιήσεις που αφορούν στον υπολογισμό του Personalized PageRank, με βάση τις προτιμήσεις των χρηστών. Αρχικά, γίνεται η ανάλυση του τρόπου με τον οποίο εφαρμόστηκε το κριτήριο σύγκλισης στην παραπάνω υλοποίηση. Στη συνέχεια, ακολουθεί η περιγραφή της υλοποίησης σχετικά με την κατανομή των βαρών σε συγκεκριμένες σελίδες, με βάση τις προτιμήσεις των χρηστών του συστήματός μας καθώς και ο υπολογισμός των τιμών του Personalized PageRank για τις σελίδες τις συλλογής.

4.4.1 Εφαρμογή του κριτηρίου σύγκλισης στον αλγόριθμο PageRank

Πριν προχωρήσουμε στην ανάλυση των υλοποιήσεων που αφορούν αποκλειστικά τον υπολογισμό του Personalized PageRank, θα παρουσιάσουμε τον τρόπο με τον οποίο εισάγαμε τον σχετικό με το κριτήριο σύγκλισης κώδικα στον υπολογισμό του original PageRank. Φυσικά, η συγκεκριμένη προσθήκη χρησιμοποιήθηκε και για τον υπολογισμό του Personalized PageRank.

Όπως έχουμε δει και παραπάνω, το κριτήριο σύγκλισης εξετάζει το άθροισμα των διαφορών των τιμών pagerank κάθε κόμβου του γράφου, μεταξύ δύο διαδοχικών επαναλήψεων:

$$\boxed{\sum \left| r_{k+1}(p_i) - r_k(p_i) \right| < \varepsilon}$$

Εάν αυτή η διαφορά γίνει μικρότερη από μία τιμή **ε** (στην περίπτωσή μας 0.00001), τότε οι επαναλήψεις διακόπτονται. Το κριτήριο σύγκλισης εξετάζεται σε κάθε 10 επαναλήψεις του αλγορίθμου.

Στην ενότητα 4.3.1 περιγράψαμε την κλάση PageRankNode, η οποία μας παρέχει τη δομημένη αναπαράσταση ενός κόμβου και κατ' επέκταση και όλου του γράφου. Για την προσθήκη του κριτηρίου σύγκλισης, προσθέσαμε μία ακόμη μεταβλητή στην παραπάνω κλάση, την οποία ονομάσαμε **previousPagerank** και αναπαριστά την τιμή pagerank του κόμβου, όπως είχε υπολογιστεί στην προηγούμενη επανάληψη από την τρέχουσα. Η συγκεκριμένη τιμή ενημερώνεται κατά τις φάσεις Map και Reduce του πρώτου βήματος κάθε επανάληψης, ώστε να απεικονίζει πάντα την προηγούμενη τιμή pagerank κάθε κόμβου. Η χρησιμότητά της είναι εμφανής κατά τη **φάση Map του δεύτερου βήματος**, όπου και η τιμή previousPagerank του τρέχοντος κόμβου διαβάζεται και εν συνεχεία, υπολογίζεται η απόλυτη τιμή της διαφοράς της από

την τιμή pagerank που λαμβάνει ο κόμβος στην τρέχουσα επανάληψη. Κάθε τέτοια διαφορά, αποθηκεύεται σε ξεχωριστό αρχείο στο HDFS (στη συνάρτηση cleanup() της φάσης Map), οπότε στο τέλος κάθε επανάληψης, εκείνες οι διαφορές αθροίζονται και η τιμή που προκύπτει συγκρίνεται με το **ε,** ώστε να εξεταστεί το κριτήριο σύγκλισης.

4.4.2 Κατανομή βαρών σε σελίδες με βάση τις προτιμήσεις των χρηστών

Όπως εξηγήσαμε στην ενότητα 2.4.2, ο original αλγόριθμος PageRank μπορεί να τροποποιηθεί με τέτοιο τρόπο ώστε να λαμβάνει υπόψιν τις προτιμήσεις χρηστών για συγκεκριμένες σελίδες. Το πρώτο βήμα για την υλοποίηση του εξατομικευμένου (Personalized) PageRank που έγινε στην εργασία μας, είναι η κατανομή βαρών σε συγκεκριμένες σελίδες του γράφου για τον εκάστοτε χρήστη. Υποθέσαμε ότι το σύστημά μας χρησιμοποιείται από 14 χρήστες, οι οποίοι εμφανίζουν διαφορετικές προτιμήσεις σε σχέση με τις σελίδες της συλλογής. Όσον αφορά στο ποιες σελίδες θεωρήσαμε ότι προτιμά ο κάθε χρήστης, αυτό είναι κάτι που θα μας απασχολήσει στο κεφάλαιο 5, όπου γίνεται διεξοδική ανάλυση της πειραματικής διαδικασίας. Προς το παρόν παρουσιάζονται οι σχετικές διεργασίες σε επίπεδο υλοποίησης.

Για να εισάγουμε το βάρος μίας σελίδας ως παράμετρο στον κώδικά μας, χρειάστηκε να προσθέσουμε ακόμα μία μεταβλητή στην κλάση PageRankNode. Έτσι, εισάγαμε τη μεταβλητή **weight**, η οποία συμβολίζει τον όρο w_{pi} στον παρακάτω τύπο του Personalized PageRank:

$$R_{k+1}(p_i) = d^* \left(\sum_{p_j \in Bp_i, |p_j| \neq 0} \frac{Rk(p_j)}{|p_j|} + \sum_{p_j, |p_j| = 0} \frac{Rk(p_j)}{N} \right) + (1-d)^* \mathbf{w_{pi}}$$

Πριν προχωρήσουμε στην παρουσίαση των δύο Map-Reduce jobs που εκτελέστηκαν ώστε να επιτευχθεί η κατανομή των βαρών στους κόμβους-σελίδες του γράφου, πρέπει να αναφέρουμε ότι η Map-Reduce job που αντιστοιχεί στην κλάση BuildPageRankRecords.java, εκτελέστηκε φυσικά και για το Personalized PageRank. Κατά τη δημιουργία των εγγραφών για το Personalized PageRank, η τιμή της μεταβλητής weight για κάθε κόμβο, αρχικοποιείται στην τιμή 0. Έτσι, κάθε κόμβος του γράφου αναπαρίσταται ως ένα αντικείμενο της κλάσης PageRankNode.java, συνοδευόμενο πάντα από το id του κόμβου, node_id.

Για κάθε χρήστη του συστήματος, οι δύο υπεύθυνες Map-Reduce jobs (κλάση **DistributeWeightsToPages.java**) για την κατανομή των βαρών εκτελούνται ξεχωριστά, καθώς οι προτιμήσεις κάθε χρήστη είναι διαφορετικές. Ακολουθεί η περιγραφή τους:

<u>Map-Reduce job 1 (Μέτρηση πλήθους προτιμώμενων σελίδων για κάθε χρήστη)</u>

Ο σκοπός της παρούσας διεργασίας είναι να υπολογιστεί το πλήθος των σελίδων που συμπεριλαμβάνονται στις προτιμήσεις κάθε χρήστη. Αποφασίσαμε ότι οι χρήστες μεταξύ τους θα έχουν διαφορετικές προτιμήσεις, οπότε επιλέξαμε να αναθέσουμε διαφορετικό εύρος προτιμώμενων σελίδων σε κάθε χρήστη. Για παράδειγμα, για τον χρήστη 1, οι προτιμώμενες σελίδες θα είναι εκείνες με node_id από 1 έως 100000, για τον χρήστη 2, θα είναι από 100001 έως 200000 κοκ. Όμως, μέσα στο εύρος των σελίδων προτίμησης κάθε χρήστη, είναι φυσικό να υπάρχουν σελίδες δίχως εξω-συνδέσμους, στις οποίες επιλέγουμε να μη δώσουμε κάποιο βάρος διότι εκ των προτέρων, το pagerank της συγκεκριμένης σελίδας θα διαμοιραστεί ισόποσα σε όλους τους κόμβους του γράφου. Επίσης, το βάρος που θα δοθεί σε κάθε προτιμώμενη σελίδα, θα είναι ίσο με **1/numOfPreferredPages**, οπότε εάν επιλέγονταν και οι κόμβοι με κενή λίστα γειτνίασης,

τότε θα λάμβαναν μερίδιο χωρίς στην ουσία να μπορεί να αξιοποιηθεί, λόγω της έλλειψης εξω-συνδέσμων. Γι' αυτό και είναι σημαντικό να γνωρίζουμε πόσες σελίδες θα επιλεχθούν από το εκάστοτε εύρος, ώστε να τους δοθεί βάρος προτίμησης.

Δεδομένα Εισόδου

Τα δεδομένα εισόδου θα είναι εκείνα που προέκυψαν από την εκτέλεση της job που αντιστοιχεί στην κλάση BuildPageRankRecords.java και κάθε γραμμή θα είναι της μορφής:

key	**value**
<node_id>	<node(*Complete*)>

όπως περιγράψαμε και στην ενότητα 4.3.1.

Φάση Map

Κατά τη φάση Map, αυτό που γίνεται είναι ο έλεγχος για το εάν το node_id του τρέχοντος κόμβου βρίσκεται μέσα στα όρια των προτιμώμενων σελίδων του χρήστη για τον οποίο εκτελείται η job, καθώς επίσης και εάν η λίστα γειτνίασης του κόμβου έχει τουλάχιστον μία εξερχόμενη ακμή. Δηλαδή, όπως εξηγήσαμε παραπάνω, επιλέγουμε να μην προσδώσουμε βάρος προτίμησης σε κάποια σελίδα που δεν περιέχει εξω-συνδέσμους. Εάν ο τρέχων κόμβος έχει τα χαρακτηριστικά ώστε να είναι προτιμώμενος, τότε εξάγεται το id του από τη φάση Map. Δηλαδή, τα ενδιάμεσα δεδομένα από τη Map στη Reduce θα ειναι:

key	**value**
<node_id>	<"">

Η τιμή είναι κενή, καθώς μας χρειάζεται μόνο το node_id.

Φάση Reduce

Στη φάση Reduce, μετρώνται πόσα κλειδιά (δηλαδή node ids) επιλέχθηκαν από τη φάση Map. Οπότε, στη συνάρτηση cleanup() της φάσης Reduce, το πλήθος των επιλεγμένων σελίδων αποθηκεύεται σε αρχείο στο HDFS. Από τη φάση Reduce, δεν έχουμε δεδομένα εξόδου, καθώς ο σκοπός της διεργασίας ήταν να μετρηθούν οι προτιμώμενες σελίδες.

Η παραπάνω Map-Reduce job εκτελείται για όλους τους χρήστες του συστήματος, οπότε το πλήθος των σελίδων που επιλέγονται για κάθε χρήστη, αποθηκεύεται σε ξεχωριστό αρχείο για τον καθένα στο HDFS ("NumOfBiasedPages-user-***i***").

Map-Reduce job 2 (Κατανομή βαρών στις προτιμώμενες σελίδες κάθε χρήστη)

Έχουμε φτάσει στο σημείο όπου τα βάρη πρέπει να κατανεμηθούν στις προτιμώμενες σελίδες κάθε χρήστη. Πριν τη φάση Map της παρούσας job, διαβάζεται η τιμή του πλήθους των επιλεγμένων σελίδων για τον τρέχων χρήστη από το αρχείο που βρίσκεται στο HDFS, όπως περιγράψαμε στην πρώτη job. Η συγκεκριμένη τιμή, εφόσον γίνει διαθέσιμη, χρησιμοποιείται για

τον υπολογισμό του βάρους που θα κατανεμηθεί στις σελίδες που επιλέχθηκαν. Έτσι, το αποτέλεσμα της διαίρεσης 1/numOfPreferredPages εισέρχεται ως παράμετρος στη φάση Map.

Δεδομένα Εισόδου

Τα δεδομένα εισόδου, όπως και στην προηγούμενη job, θα είναι στην ουσία η δομημένη αναπαράσταση του γράφου, σε εγγραφές του τύπου PageRankNode και τα οποία προκύπτουν από την κλάση BuildPageRankRecords.java:

key	**value**
<node_id>	<node(*Complete*)>

Φάση Map

Όπως και πριν, ο κόμβος που εισέρχεται στη συνάρτηση map(), ελέγχεται εάν το id του βρίσκεται εντός του εύρους των επιλεγμένων σελίδων για τον εκάστοτε χρήστη και εάν η λίστα γειτνίασής του περιέχει τουλάχιστον ένα κόμβο. Αν ο κόμβος ανήκει στους προτιμώμενους κόμβους-σελίδες, τότε δημιουργείται ένα νέο αντικείμενο της κλάσης PageRankNode, το οποίο θα έχει ακριβώς τα ίδια στοιχεία με την τρέχουσα value, εκτός από το **weight**, το οποίο λαμβάνει την τιμή 1/numOfPreferredPages. Σ' αυτήν την περίπτωση, εξάγεται το νέο αυτό αντικείμενο. Αν όμως ο κόμβος δεν ανήκει στους προτιμώμενους, τότε το κλειδί και η τιμή εξόδου θα είναι ακριβώς τα ίδια με εκείνα που εισήχθησαν στην συγκεκριμένη map διεργασία. Η μορφή των δεδομένων εξόδου θα είναι και πάλι:

key	**value**
<node_id>	<node(*Complete*)>

με κάποιους κόμβους να έχουν μη μηδενική τιμή βάρους προτίμησης, ενώ για όλους τους υπόλοιπους η τιμή του βάρους θα παραμείνει ως είχε, δηλαδή μηδέν. Τα δεδομένα που εξάγονται από τη φάση Map θα είναι και τα τελικά δεδομένα εξόδου της Map-Reduce job, η οποία εκτελείται μία φορά για κάθε χρήστη. Τέλος, είναι φυσικό τα δεδομένα εξόδου να είναι διαφορετικά για τον καθένα όσον αφορά τα βάρη προτίμησης.

4.4.3 Προσθήκη εξατομίκευσης (personalization) στον αλγόριθμο PageRank

Στην ενότητα 4.3.2 παρουσιάσαμε την υλοποίηση του original αλγορίθμου PageRank στο Hadoop με βάση την βιβλιοθήκη cloud9. Για την προσθήκη του στοιχείου της εξατομίκευσης (personalization) στον αλγόριθμο, εφαρμόστηκαν αλλαγές σε συγκεκριμένα σημεία του κώδικα, ώστε να επιτευχθεί το επιθυμητό αποτέλεσμα. Όπως εξηγήσαμε και παραπάνω, για το Personalized PageRank, εισάγαμε μία ακόμη μεταβλητή (weight) στην κλάση PageRankNode, η οποία απεικονίζει το βάρος προτίμησης κάθε κόμβου με βάση τον εκάστοτε χρήστη. Η τιμή της συγκεκριμένης μεταβλητής ενημερώνεται κατά τη διάρκεια των φάσεων Map και Reduce του **βήματος 1** του αλγορίθμου, το οποίο έχουμε ήδη αναλύσει.

Όπως έχουμε αναφέρει και στο κεφάλαιο 2, όπου έγινε η θεωρητική ανάλυση του αλγορίθμου PageRank, η εξατομίκευση του αλγορίθμου μπορεί να επιτευχθεί επεμβαίνοντας στο βήμα της τυχαίας μεταπήδησης σε οποιαδήποτε σελίδα της συλλογής, μη συμπεριλαμβανομένων των εξω-

συνδέσμων της τρέχουσας σελίδας. Έτσι, κοιτάζοντας ξανά τους μαθηματικούς τύπους για το Original και Personalized PageRank, η διαφορά βρίσκεται στον όρο που αναπαριστά την τυχαία αυτή μεταπήδηση:

- **Original PageRank:** (1-d)* = (1-d)*(1/N)
- **Personalized PageRank:** (1-d)* w_{pi}

Στην περίπτωση του Original PageRank, η $\sum_{p_j} \frac{Rk(p_j)}{N}$ ι να γίνει η μεταπήδηση σε μία τυχαία σελίδα είναι ίδια για όλες τις σελίδες του γράφου, ενώ υσον αφορά το Personalized PageRank, η μεταπήδηση επηρεάζεται από το βάρος προτίμησης του εκάστοτε χρήστη για τη συγκεκριμένη σελίδα. Έτσι, με βάση την κατανομή των βαρών (ενότητα 4.4.2), **μόνο** οι προτιμώμενες σελίδες του χρήστη θα λαμβάνουν βάρος προτίμησης ίσο με 1/numOfPreferredPages , ενώ για τις υπόλοιπες, η τιμή w_{pi} θα είναι μηδενική. Δηλαδή, το βήμα της "τυχαίας" μεταπήδησης δε θα γίνεται πλέον για οποιαδήποτε σελίδα, αλλά μόνο για εκείνες που συμπεριλαμβάνονται στις προτιμήσεις του χρήστη.

Από την ενότητα 4.3.2, έχουμε δει ότι το **βήμα 2** του αλγορίθμου και η Map-Reduce job που αντιστοιχεί σε αυτό, ολοκληρώνει τον υπολογισμό του pagerank κάθε κόμβου, λαμβάνοντας υπόψιν τους "dangling" κόμβους, καθώς επίσης και την τυχαία μεταπήδηση (random jump). Ως εκ τούτου, τα δύο αυτά στοιχεία δε περιλαμβάνονται στις τιμές των δεδομένων εξόδου του βήματος 1, τα οποία αποτελούν παράλληλα την είσοδο του βήματος 2. Η παραλλαγή στο 2ο βήμα του αλγορίθμου (ενότητα 4.3.2), με σκοπό την προσαρμογή για εξατομίκευση, γίνεται στη **φάση Map** του βήματος:

Τα δεδομένα που εισέρχονται προς επεξεργασία είναι ζεύγη της μορφής (<node_id>,<node(*Complete*)>). Αρχικά, λοιπον, ελέγχεται εάν ο τρέχων κόμβος συνοδεύεται από κάποιο βάρος προτίμησης, δηλαδή εάν η τιμή **weight (w_{pi})** είναι διάφορη του μηδενός. Εάν όντως υπάρχει κάποιο βάρος προτίμησης, τότε ο όρος (1-d)*w_{pi} είναι επίσης μη μηδενικός, οπότε λαμβάνει μέρος και στο συνολικό υπολογισμό του Personalized PageRank. Σε διαφορετική περίπτωση, αν δηλαδή ο κόμβος δεν έγκειται στις προτιμήσεις του χρήστη για τον οποίο εκτελείται ο υπολογισμός, τότε ο παραπάνω όρος είναι ίσος με το μηδέν. Στη συνέχεια, υπολογίζεται συνολικά η τιμή του Personalized PageRank, η οποία ενημερώνει την τιμή της μεταβλητής **pagerank** του κόμβου, ενώ εκτελούνται και παράπλευροι υπολογισμοί σχετικοί με το κριτήριο σύγκλισης, όπως έχουμε εξηγήσει παραπάνω (ενότητα 4.4.1). Τελικά, **για κάθε χρήστη** του συστήματος και μετά το πέρας των επαναλήψεων που εκτελούνται, αποθηκεύονται ζεύγη της μορφής (<node_id>,<node(*Complete*)>) στο HDFS, τα οποία φυσικά συνιστούν ολόκληρο το γράφο. Γίνεται εύκολα αντιληπτό ότι οι τιμές pagerank των κόμβων θα είναι διαφορετικές για κάθε χρήστη, από τη στιγμή που εισήχθησαν βάρη προτίμησης σε διαφορετικούς κόμβους-σελίδες για τον καθένα. Στο επόμενο κεφάλαιο θα παρουσιαστούν παραδείγματα που καταδεικνύουν πώς διαμορφώθηκαν οι τιμές pagerank των σελίδων με βάση τις προτιμήσεις των χρηστών.

4.5 Συνδυασμός των scores: PageRank και Cosine Similarity

Έχοντας υπολογίσει τις τιμές του Personalized PageRank για όλους τους κόμβους του γράφου και για κάθε χρήστη του συστήματος, αυτό που απομένει για την ολοκλήρωση της υλοποίησής μας, είναι ο συνδυασμός των παραπάνω τιμών με τις τιμές που έχουν προκύψει από τον υπολογισμό της Ομοιότητας Συνημιτόνου-Cosine Similarity (ενότητα 3.7). Όπως έχουμε δει, το μέτρο Cosine

Similarity έχει υπολογιστεί για συγκεκριμένα επερωτήματα που δόθηκαν από εμάς και εξαρτάται αποκλειστικά από το λεκτικό περιεχόμενο των σελίδων της συλλογής μας. Από την άλλη πλευρά, το PageRank είναι ένα μέτρο που δεν λαμβάνει υπόψιν το περιεχόμενο των σελίδων, παρά μόνο τους συνδέσμους μεταξύ τους. Γι' αυτό το λόγο, θεωρούμε ότι ο συνδυασμός των δύο μεθόδων-scores έχει αρκετό ενδιαφέρον, αφού θα εξαχθούν αποτελέσματα που θα ανταποκρίνονται στα επερωτήματα των χρηστών με βάση τους όρους που δόθηκαν, καθώς επίσης **και** με βάση το "κύρος" των σελίδων και τις προτιμήσεις (Personalized PageRank) του εκάστοτε χρήστη.

4.5.1 Εξαγωγή των ζευγών <id κόμβου-τιμή pagerank> και εγγραφή στο HDFS

Σκοπός της εργασίας μας είναι να συνδυάσουμε τα scores Cosine Similarity και PageRank (Original και Personalized) ώστε να βελτιστοποιήσουμε όσο το δυνατόν περισσότερο την ποιότητα των αποτελεσμάτων. Για την ώρα, έχουμε στη διάθεσή μας δεδομένα της μορφής (<node_id>,<node(*Complete*)>), όπως εκείνα προέκυψαν από τον επαναληπτικό υπολογισμό του PageRank. Κατά τη φάση του συνδυασμού των scores, προέκυψε η ανάγκη ύπαρξης ζευγών της μορφής (<node_id>,<node_pagerank>) για κάθε χρήστη. Η δημιουργία των ζευγών αυτών παρουσιάζεται πριν από το βήμα του συνδυασμού των scores, καθώς είναι προαπαιτούμενο ώστε να επιτευχθεί η συγχώνευση των δύο μέτρων.

Η κλάση **BuildDocumentPagerankPairs.java** υλοποιεί τα παραπάνω και μέσω αυτής εκτελείται μία Map-Reduce job για κάθε χρήστη, καθώς επίσης και για τα ζεύγη που έχουν προκύψει από τον πρωτότυπο αλγόριθμο PageRank.

<u>Δεδομένα Εισόδου</u>

Τα δεδομένα που δίνονται ως είσοδος είναι εκείνα που έχουν προκύψει από τον υπολογισμό των τιμών του PageRank για κάθε χρήστη. Οπότε, τα δεδομένα εισόδου αποτελούνται από ζεύγη της μορφής (<node_id>,<node(*Complete*)>).

<u>Φάση Map</u>

Κατά τη φάση Map και μέσα στη συνάρτηση map(), αυτό που γίνεται είναι η εξαγωγή της τιμής pagerank του τρέχοντος κόμβου, συνοδευόμενης από το id του κόμβου. Τα ενδιάμεσα δεδομένα είναι της μορφής:

key	**value**
<node_id>	<pagerank>

όπου το *node_id* είναι τύπου IntWritable και το *pagerank* τύπου FloatWritable.

<u>Φάση Reduce</u>

Σκοπός της φάσης Reduce είναι η αποθήκευση των ενδιάμεσων δεδομένων σε Sequence Files, αλλά και σε μορφή κειμένου. Τα αρχεία τύπου **SequenceFile** θα χρησιμεύσουν κατά τη φάση του συνδυασμού των scores (ενότητα 4.5.2), ενώ τα αρχεία τύπου **Text** μας βοήθησαν στην ανάλυση των πειραμάτων και στην εξαγωγή γραφικών παραστάσεων (κεφάλαιο 5). Γι' αυτό το λόγο, χρησιμοποιήσαμε **1** Reducer· εκτελέστηκε δηλαδή μία διεργασία map (map task), ώστε το αρχείο κειμένου να μην είναι κατακερματισμένο, αλλά ενοποιημένο σε ένα αρχείο.

Αρχικά, γίνεται υπερκάλυψη (override) της μεθόδου **setup()** που μας προσφέρει η κλάση Mapper του Hadoop. Η συγκεκριμένη μέθοδος εκτελείται μία φορά στην αρχή κάθε map task. Σκοπός της setup() είναι η αρχικοποίηση των τμημάτων που θα αποτελούν το Sequence File που

χρειαζόμαστε. Έτσι, χρησιμοποιείται η κλάση **SequenceFile.Writer**, η οποία στην ουσία είναι υπεύθυνη για την εγγραφή των δεδομένων στο Sequence File. Στη συγκεκριμένη υλοποίηση, χωρίσαμε το Sequence File σε 2 μέρη, ούτως ώστε η προσπέλαση των δεδομένων του στην κύρια μνήμη να είναι πιο γρήγορη και περισσότερο ανθεκτική σε αστοχίες (π.χ. υπέρβαση ορίου χρήσης κύριας μνήμης που χρησιμοποιείται από το Hadoop).

Μεταβαίνοντας λοιπόν στη συνάρτηση **reduce()**, εκτελείται η πράξη **(node_id%2)**, το αποτέλεσμα της οποίας μας δείχνει σε ποιο από τα 2 μέρη του Sequence File θα αποθηκευτεί η τρέχουσα εγγραφή. Στη συνέχεια, το ζεύγος (<node_id>,<pagerank>) αποθηκεύεται ταυτόχρονα σε μορφή **SequenceFile** και **TextOutputFormat** στο HDFS.

4.5.2 Υπολογισμός ενοποιημένου score Cosine Similarity-Personalized PageRank

Έχουμε φτάσει στη φάση όπου απομένει να γίνει ο τελικός υπολογισμός του score που συνδυάζει τα scores Cosine Similarity και PageRank (Original και Personalized). Λαμβάνονται λοιπόν υπόψιν τα δεδομένα που έχουν προκύψει από τον υπολογισμό του Cosine Similarity (ενότητα 3.7) και του PageRank, για όλους τους χρήστες του συστήματος με βάση τις προτιμήσεις τους.

Η Map-Reduce job (κλάση **CombinePagerankCosineSim.java**) που είναι υπεύθυνη για τον παραπάνω συνδυασμό και εκτελείται επαναληπτικά, μία φορά για κάθε χρήστη, έχει ως εξής:

<u>Δεδομένα Εισόδου</u>

Τα δεδομένα εισόδου αποτελούνται από τα δεδομένα που έχουν προκύψει από τον υπολογισμό της Cosine Similarity (ενότητα 3.7) για τα δοσμένα επερωτήματα και είναι της μορφής:

key	**value**
<query>	<sortedList(docId,CosSimNormalized)>

με το επερώτημα (query) να αποτελεί το κλειδί (key) και τη ταξινομημένη λίστα (ως προς την τιμή του Cosine Similarity σε φθίνουσα σειρά) με τα ζεύγη <docId>,<CosSimNormalized> να αποτελούν την τιμή (value) των δεδομένων εισόδου. Το κλειδί είναι τύπου Text, ενώ η τιμή είναι τύπου ArrayListWritable<PairOfIntFloat>.

Εκτός όμως από τα δεδομένα αυτά, χρειάζεται με κάποιο τρόπο να γίνει χρήση και των τιμών του Personalized PageRank. Η συγκεκριμένη ενέργεια γίνεται με την εισαγωγή των τιμών αυτών ως **παράπλευρα δεδομένα** της job, τα οποία βρίσκονται σε μορφή SequenceFile, όπως εξηγήσαμε στην 4.5.1.

<u>Φάση Map</u>

Αρχικά, δημιουργείται μία ArrayList, η οποία μπορεί να περιέχει ένα πλήθος από αντικείμενα του τύπου ArrayList<PairOfIntFloat>. Δηλαδή, η πρώτη λίστα περιλαμβάνει μία πλειάδα από λίστες. Στην περίπτωσή μας, η πρώτη λίστα περιλαμβάνει 2 λίστες, όσα και τα τμήματα καθενός από τα Sequence Files με τις τιμές pagerank που αντιστοιχούν σε κάθε χρήστη του συστήματος. Κάποιος θα μπορούσε να αναρωτηθεί γιατί επιλέχθηκε η πρώτη λίστα, ενώ θα μπορούσαν απλά να δημιουργηθούν δύο αντικείμενα της δεύτερης. Η απάντηση είναι ότι η υλοποίηση που επιλέξαμε μπορεί να παραμετροποιηθεί για πολύ μεγαλύτερα Sequence Files, τα οποία θα είναι κατακερματισμένα σε περισσότερα από δύο μέρη.

Έστω τώρα ότι το τρέχον ζεύγος τιμής-κλειδιού είναι το query1-value1. Τα περιεχόμενα της λίστας value1 διατρέχονται και το κάθε ζεύγος PairOfIntFloat που περιέχεται σε αυτήν, αποθηκεύεται σε μία από τις λίστες που αναφέραμε προηγουμένως. Η επιλογή της λίστας στην οποία θα αποθηκευτεί το εκάστοτε ζεύγος γίνεται χρησιμοποιώντας την τιμή **(docId%2),** όπως ακριβώς είχαμε κάνει και για τα μέρη του Sequence File που περιέχει ζεύγη <node_id><pagerank>, όπου το node_id αντιστοιχεί στην τιμή του docId. Έτσι, για παράδειγμα, εάν ένα ζεύγος <node_id><pagerank> είναι αποθηκευμένο στο 2ο μέρος του Sequence File, τότε το ζεύγος <docId><CosSimNormalized> που αντιστοιχεί στο ίδιο node_id, θα βρίσκεται στη 2η κατά σειρά λίστα-αντικείμενο της αρχικής ArrayList. Αυτό γίνεται για τον εξής λόγο: τα αρχεία SequenceFile που είναι αποθηκευμένα στο HDFS δεν μπορούν να προσπελαστούν με την ίδια ταχύτητα όπως εάν βρίσκονταν στην κύρια μνήμη. Γι' αυτό το λόγο, χρησιμοποιήθηκε η δομή **HashMap,** η οποία αποθηκεύει ζεύγη <key,value>, όπου η value μπορεί να ανακτηθεί άμεσα με βάση το key. Έτσι, λόγω του ότι θα δημιουργηθούν περισσότερα από 1 HashMap, αυτό που θέλουμε να πετύχουμε είναι να μη βρίσκονται παραπάνω από 1 στην κύρια μνήμη κάθε στιγμή. Οπότε, έχοντας διαμοιράσει τις τιμές pagerank με βάση την τιμή docId%2 στην περίπτωσή μας, διατρέχουμε αρχικά την πρώτη λίστα με τα ζεύγη <docId,pagerank>. Κατά την πρώτη επανάληψη για τη λίστα αυτή, οι τιμές του SequenceFile αποθηκεύονται στο HashMap σε αντίστοιχα ζεύγη <docId,pagerank> του τύπου <Integer,Float>. Έτσι, το συγκεκριμένο HashMap είναι διαθέσιμο για όλες τις εγγραφές που αντιστοιχούν στο τμήμα του SequenceFile που εξετάζεται και με αυτόν τον τρόπο επιτυγχάνεται άμεση πρόσβαση στα ζεύγη <docId,pagerank> εκείνου του τμήματος και κατ' επέκταση και όλου του αρχείου, καθώς για το επόμενο τμήμα ξεκινάει νέα επανάληψη με νέο HashMap κοκ.

Έχοντας αποκτήσει λοιπόν άμεση πρόσβαση στις τιμές pagerank όλων των κόμβων, στη συνέχεια ανακτώνται οι τιμές pagerank που αντιστοιχούν στις 1000 πιο σχετικές σελίδες-κείμενα, οι οποίες έχουν ήδη προκύψει από τον υπολογισμό της Cosine Similarity για το εκάστοτε επερώτημα. Εκείνες οι τιμές pagerank αποθηκεύονται ακολούθως σε μία ArrayList<PairOfIntFloat> συνοδευόμενες από το id της σελίδας-κειμένου-κόμβου. Ο λόγος που αποθηκεύονται στην ArrayList (λ.χ. myList) είναι ότι υπάρχει η δυνατότητα ταξινόμησης των στοιχείων της με βάση την τιμή pagerank. Χρησιμοποιείται η συνάρτηση **sort()** ως εξής:

Collections.sort(myList, new ScoresComparator()) ,

όπου ScoresComparator μία κλάση, της οποίας η μέθοδος *compare()* συγκρίνει δύο αντικείμενα, στην περίπτωση μας του τύπου PairOfIntFloat.

<u>Κανονικοποίηση</u>

Ο λόγος που γίνεται ταξινόμηση της λίστας είναι για να λάβουμε το μέγιστο και το ελάχιστο pagerank score που έχει προκύψει μέχρι στιγμής για το τρέχων επερώτημα, ώστε να εφαρμόσουμε **κανονικοποίηση** των scores με βάση το μέχιστο και το ελάχιστο score, όπως είχαμε κάνει και στην ενότητα 3.7 για τις τιμές της Cosine Similarity για κάθε επερώτημα.

Στο σημείο αυτό, πρέπει να τονιστεί η σημασία της παραπάνω διαδικασίας στην εξαγωγή **ασφαλών** αποτελεσμάτων. Τι θα γινόταν εάν δεν είχε εφαρμοστεί κανονικοποίηση στα παραπάνω scores; Η απάντηση δίνεται με βάση το εξής παράδειγμα: Έστω ότι έχουμε στη διάθεσή μας 10 αποτελέσματα με βάση την Cosine Similarity, οι τιμές της οποίας κυμαίνονται από 0.1 έως 0.9, καθώς όπως έχουμε εξηγήσει, το συγκεκριμένο score είναι στην ουσία ένα μέτρο για τη μέτρηση της ομοιότητας ενός επερωτήματος και του περιεχόμενο μίας σελίδας. Από την άλλη πλευρά, θα μπορούσαμε να έχουμε τις 10 τιμές pagerank οι οποίες αντιστοιχούν στις σελίδες-κόμβους και

εκείνες να κυμαίνονται από 0.000002 έως 0.009, εφόσον οι τιμές pagerank αποτελούν στην ουσία ένα μερίδιο από τη συνολική "μάζα" του συγκεκριμένου score, η οποία είναι ίση με 1. Οπότε, εάν συνδυάζαμε αυτά τα 2 scores, οι τιμές pagerank δε θα είχαν στην ουσία κάποια επίδραση στο ενοποιημένο score, κάτι που φυσικά δεν είναι επιθυμητό.

Αντίθετα, εφαρμόζοντας κανονικοποίηση, κατορθώνουμε να έχουμε scores στην ίδια κλίμακα, κάτι που σαφώς επιφέρει **αξιόπιστα** αποτελέσματα, τα οποία συνδυάζουν και τις δύο μεθόδους.

Συνδυασμός των scores

Έτσι, έχοντας κανονικοποιήσει και τις τιμές pagerank για κάθε επερώτημα, υπάρχει η δυνατότητα να συνδυαστούν τα scores Cosine Similarity και PageRank ώστε να εξαχθεί ένα ενοποιημένο score. Γι' αυτό, διατρέχονται τα στοιχεία της τρέχουσας value, η οποία είναι τύπου ArrayListWritable<PairOfIntFloat> και περιέχει τις κανονικοποιημένες τιμές cosine similarity για το τρέχων επερώτημα. Οπότε, για κάθε σελίδα ανακτάται η τιμή pagerank που της αντιστοιχεί και στη συνέχεια τα 2 scores συνδυάζονται. Ακολουθήσαμε τη μέθοδο **Weighted Linear Combination**, η οποία χρησιμοποιείται στα πειράματα των [18] και [19]. Στο [18] η μέθοδος εφαρμόζεται μετά από κανονικοποίηση των scores που λαμβάνουν μέρος στη διαδικασία του ranking. Με αντίστοιχο τρόπο, προχωράμε στο συνδυασμό των 2 scores:

*combinedScore = weight1*CosSim + weight2*PPR , με weight1+weight2=1*

Παρατηρούμε ότι σε καθεμιά από τις 2 μεθόδους αναθέτουμε ένα βάρος, το οποίο αντικατοπτρίζει την **επίδραση του** αντίστοιχου **score στο τελικό αποτέλεσμα**. Όσον αφορά στις τιμές που επελέγησαν για τα βάρη, εκείνες θα παρουσιαστούν κατά την ανάλυση της πειραματικής διαδικασίας (Κεφάλαιο 5).

Τελικώς, στην έξοδο της φάσης Map θα έχουμε για κάθε query, τα **1000** πιο σχετικά αποτελέσματα με βάση το μέτρο Cosine Similarity, συνδυασμένα με την τιμή pagerank που αντιστοιχεί σε αυτά.

Συνεπώς, τα δεδομένα εξόδου από τη φάση Map θα είναι της μορφής:

key	**value**
<query>	(<docId>,<combinedScore>)

όπου το key είναι τύπου *Text* και η value τύπου *PairOfWritables<IntWritable,FloatWritable>*.

Φάση Reduce

Στη φάση Reduce εισάγονται δεδομένα της μορφής (<query>,<listOf(<docId>,<combinedScore>)>). Για κάθε επερώτημα που εισέρχεται στη συνάρτηση reduce(), διατρέχεται η λίστα με τις τιμές που το συνοδεύουν. Οι τιμές εκείνες μορφοποιούνται κατάλληλα σε αντικείμενα PairOfIntFloat και εισάγονται σε μία ArrayList, η οποία ταξινομείται με τον ίδιο τρόπο, όπως εξηγήσαμε και στη φάση Map. Ο λόγος που γίνεται αυτό είναι ώστε να κρατήσουμε τα 100 (ή λιγότερα εάν δεν επαρκούν) ζεύγη (<docId>,<combinedScore>) με το υψηλότερο ενοποιημένο score, για κάθε επερώτημα, από τα 1000 (ή λιγότερα) που έχουμε στη διάθεσή μας και δεν είναι ταξινομημένα.

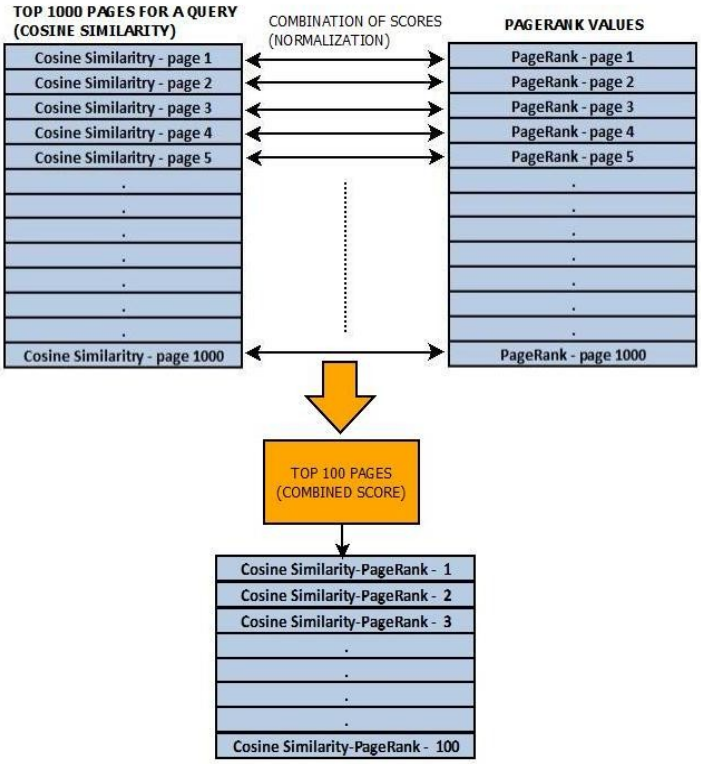
Επομένως, τα δεδομένα εξόδου της job θα αποτελούνται από ζεύγη της μορφής:

key	value
<query>	sortedListTop100(<docId>,<combinedScore>)

όπου το key είναι τύπου *Text* και η value τύπου *ArrayListWritable<PairOfIntFloat>*.

Η παραπάνω διαδικασία απεικονίζεται σχηματικά στο διάγραμμα 4.2.

Με την παρούσα job, στην ουσία ολοκληρώθηκε ο υπολογισμός του ενοποιημένου score που συνδυάζει τις μεθόδους-τιμές Cosine Similarity και Personalized PageRank, δεδομένων κάποιων επερωτημάτων και διαφόρων προτιμήσεων των χρηστών του συστήματός μας. Στην επόμενη ενότητα δείχνουμε πώς τα ids των σελίδων στα παραπάνω αποτελέσματα μετασχηματίζονται στα urls που τους αντιστοιχούν.



Διάγραμμα 4.2: Συνδυασμός των scores Cosine Similarity & PageRank

4.5.3 Μετατροπή των ids στα αντίστοιχα urls

Σε αυτήν την ενότητα θα παρουσιάσουμε αναλυτικά τη διαδικασία κατα την οποία τα ids που αντιστοιχούν στις σελίδες των παραπάνω αποτελεσμάτων, αντικαθίστανται από τα urls τα οποία αντιπροσωπεύουν. Σε αυτό το σημείο είναι καλό να θυμηθούμε το λόγο για τον οποίο χρησιμοποιήσαμε ακέραια ids καθ' όλη τη διάρκεια της εξαγωγής των scores. Όπως εξηγήσαμε και στην ενότητα 3.3, τα ids δεσμεύουν εξαιρετικά μικρότερο χώρο απ' ότι τα urls, τα οποία είναι σε μορφή κειμένου και απαιτούν τεράστιο αποθηκευτικό χώρο. Έτσι, με τη χρήση των ids αποτρέπουμε τη δέσμευση επιπλέον χώρου, ενώ ταυτόχρονα και οι υπολογισμοί στις διάφορες

διεργασίες εκτελούνται πιο γρήγορα. Γι' αυτούς τους λόγους, επιλέξαμε τα ids για όλους τους υπολογισμούς, μέχρι και τον συνδυασμό των scores που εξετάσαμε. Μετά από όλη τη διαδικασία, οι σελίδες θα αντιπροσωπεύονται τελικά από τα urls τους, τα οποία προκύπτουν από την αντιστοίχηση τους με τα ids. Ο τρόπος που γίνεται αυτός ο μετασχηματισμός, παρουσιάζεται παρακάτω.

Η συγκεκριμένη μετατροπή λαμβάνει χώρα κατα τη διάρκεια εκτέλεσης της κλάσης **DumpTopScoreIdsToUrls.java,** η οποία περιλαμβάνει 2 Map-Reduce jobs. Στην ουσία, πριν τη μετατροπή, εκτελείται ένα προπαρασκευαστικό στάδιο, όπου τα ids μετατρέπονται σε μορφή String. Αυτό γίνεται διότι κατα τη διάρκεια της μετατροπής θα προστίθενται συνεχώς τα αντίστοιχα urls στη θέση των ids, οπότε θα πρέπει να χρησιμοποιείται ένας κοινός τύπος δεδομένων για τα 2 στοιχεία, καθώς θα συνυπάρχουν στη μεταβλητή <docId> έως ότου τα δεδομένα πάρουν την τελική τους μορφή. Πιο συγκεκριμένα, στο **πρώτο βήμα**, έχουμε:

Δεδομένα Εισόδου

Τα δεδομένα εισόδου της job είναι τα 100 πρώτα αποτελέσματα που έχουν προκύψει από το συνδυασμό των scores στην προηγούμενη ενότητα:

key	**value**
<query>	sortedListTop100(<docId>,<combinedScore>)

Μέχρι στιγμής η value είναι τύπου *ArrayListWritable<PairOfIntFloat>*. Ο στόχος μας είναι να μετασχηματιστεί σε *ArrayListWritable<PairOfStringFloat>*.

Φάση Map

Κατά τη φάση map, διατρέχεται η παραπάνω λίστα, από την οποία λαμβάνουμε τα ζεύγη (<docId>,<combinedScore>). Το docId, μετατρέπεται σε μορφή String,

String docIdString = String.valueOf(docid);

και πλέον αντικείμενα του τύπου PairOfStringFloat αποθηκεύονται σε μία νέα λίστα, η οποία με τη σειρά της και μαζί με το αντίστοιχο query, αποθηκεύεται στο HDFS.

Πλέον, τα δεδομένα είναι έτοιμα έτσι ώστε να ξεκινήσει η μετατροπή των ids σε urls, στο **δεύτερο βήμα** της διαδικασίας:

Δεδομένα Εισόδου

Τα δεδομένα εισόδου της παρούσας job θα είναι τα ίδια με παραπάνω με τη διαφορά ότι το docId είναι πλέον τύπου String.

Φάση Map

Σε αυτό το σημείο, είναι απαραίτητο να χρησιμοποιηθεί το Hadoop Mapfile το οποίο περιέχει τις αντιστοιχίσεις των ids με τα urls των σελίδων. Δηλαδή είναι της μορφής:

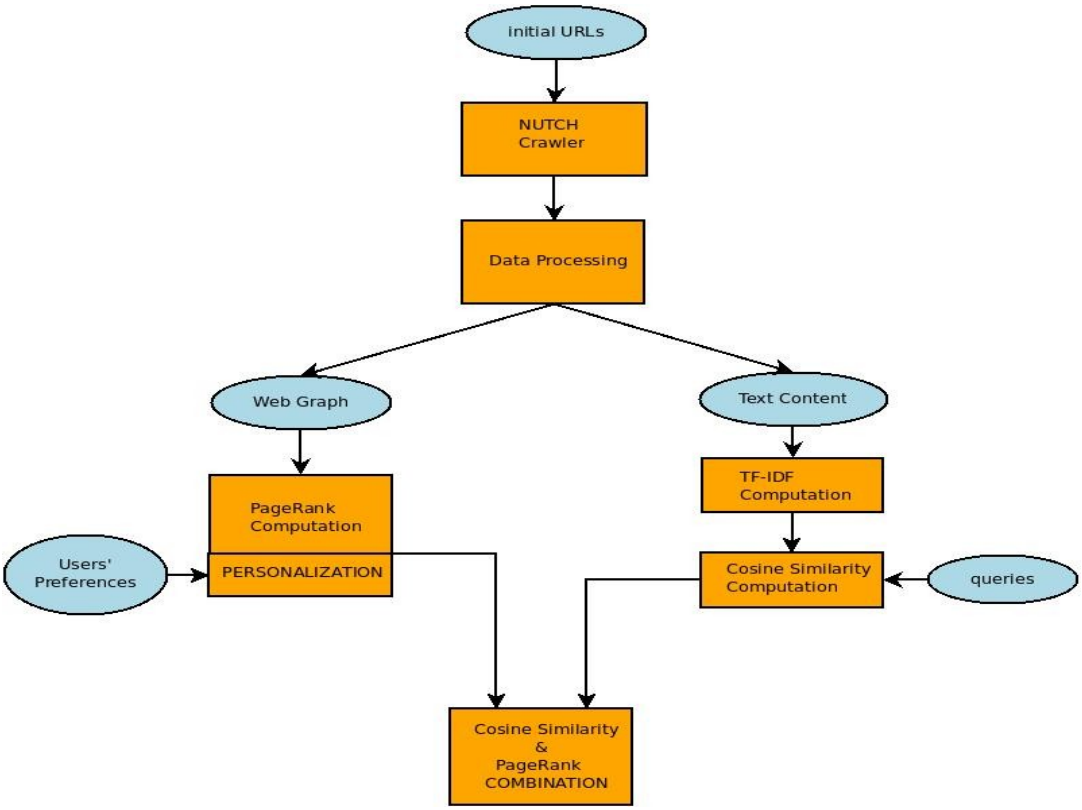
key	**value**
<docId>	<docUrl>

Λόγω του μεγέθους του αρχείου στο HDFS (περίπου 286 mb), το οποίο μας δημιουργούσε

προβλήματα κυρίως σχετικά με το χρόνο ανάκτησης των απαιτούμενων τιμών, θεωρήθηκε αναγκαίος ο κατακερματισμός του αρχείου σε 6 υπο-αρχεία μικρότερου μεγέθους. Η συγκεκριμένη τροποποίηση απαιτεί την εκτέλεση της Map-Reduce job σε 6 επαναλήψεις, 1 για κάθε Mapfile. Σε κάθε επανάληψη θα αντικαθίστανται σταδιακά τα ids με τα αντίστοιχα urls, κάτι που θα εξηγήσουμε στην επόμενη παράγραφο:

Αρχικά, κατά τη φάση **setup()** του Mapper, αποκτούμε πρόσβαση στο μέρος του mapfile που αντιστοιχεί στην τρέχουσα επανάληψη (π.χ. στην 1η επανάληψη, διαχειριζόμαστε το 1ο μέρος). Τα δεδομένα που περιέχονται μέσα σε αυτό, επιλέγουμε να τα αποθηκεύσουμε προσωρινά σε μία δομή τύπου *HashMap<Long, String>* στην κύρια μνήμη. Η συγκεκριμένη δομή μας προσφέρει ταχύτητα στην ανάκτηση των urls για κάθε docId. Στη φάση **map()**, διατρέχεται η λίστα *sortedListTop100(<docId>,<combinedScore>)*. Εάν το docId είναι αριθμός (κατά την πρώτη επανάληψη όλα τα docIds είναι αριθμοί), τότε ανακτάται το url που το αντιπροσωπεύει και αντικαθίσταται από αυτό. Εάν δεν είναι αριθμός, το οποίο σημαίνει ότι έχει γίνει ήδη η αντικατάσταση σε προηγούμενη επανάληψη, τότε το docId παραμένει ως έχει. Με αυτόν τον τρόπο, φτάνουμε στην τελευταία επανάληψη όπου έχουν χρησιμοποιηθεί όλα τα μέρη του Mapfile, οπότε όλα τα ids έχουν μετατραπεί σε urls και τα τελικά αποτελέσματα έχουν αποθηκευτεί στο HDFS.

Στο παρακάτω διάγραμμα απεικονίζονται συνολικά τα κύρια τμήματα και διαδικασίες που έλαβαν χώρα από τη φάση του Crawling μέχρι το συνδυασμό την εξαγωγή του ενοποιημένου τελικού score για τη βαθμονόμηση των αποτελεσμάτων:



Διάγραμμα 4.3: Απεικόνιση των κύριων τμημάτων της εργασίας

5. Πειραματική Διαδικασία

Στο παρόν κεφάλαιο παρουσιάζεται η διαδικασία και τα αποτελέσματα των πειραμάτων που διεξήχθησαν μετά το πέρας της υλοποίησης των τμημάτων της εργασίας που περιγράφηκαν στα προηγούμενα κεφάλαια. Παρακάτω, αναλύεται η κάθε περίπτωση ξεχωριστά, αλλά παράλληλα επιχειρείται και σύγκριση μεταξύ των αποτελεσμάτων που προέρχονται από διαφορετικές μεθόδους ή έχουν προηγουμένως κατηγοριοποιηθεί διαφορετικά, ώστε να εξαχθούν χρήσιμα συμπεράσματα και να αξιολογηθεί η υλοποίηση με το βέλτιστο δυνατό τρόπο.

5.1 Δεδομένα Εισόδου και Μεθοδολογία των Πειραμάτων

Για να αξιολογηθεί η υλοποίηση των επιμέρους τμημάτων της εργασίας, χρειάστηκε να γίνει μεθοδική κατηγοριοποίηση των δεδομένων εισόδου, ώστε να λάβουμε στην έξοδο αποτελέσματα που θα μας οδηγήσουν σε ουσιαστικά συμπεράσματα.

Παραγωγή Επερωτημάτων (Queries)

Όπως είχαμε αναφέρει και στην ενότητα 3.3 όπου έγινε η επεξεργασία των δεδομένων που εξήχθησαν από τη φάση του crawling, το **σύνολο των ιστοσελίδων** της συλλογής μας είναι 3,689,304. Επίσης, το **σύνολο των διαφορετικών όρων (terms)** που εμφανίζονται στις σελίδες είναι 21,677,397.

Θεωρήσαμε καλό να εντάξουμε τους όρους σε κατηγορίες ανάλογα με τη συνχνότητα εμφάνισης τους στη συλλογή. Αυτό θα μας βοηθήσει στην παραγωγή επερωτημάτων με όρους από διαφορετικές κατηγορίες και θα χρησιμεύσει ώστε να παρατηρήσουμε την επίδραση στα τελικά αποτελέσματα και συγκεκριμένα στο score Cosine Similarity (Ομοιότητα Συνημιτόνου). Οι όροι χωρίστηκαν σε 3 κατηγορίες: frequent, medium-frequent και non-frequent. Η κατηγοριοποίηση των όρων έγινε **διαισθητικά** με βάση την κατανομή του πλήθους εμφανίσεων κάθε όρου στη συλλογή (Πίνακας 5.1).

Έγινε λοιπόν, σε offline χρόνο, παραγωγή 100 επερωτημάτων (queries), τα οποία είναι 4 κατηγοριών: Εκείνα που αποτελούνται από frequent terms, εκείνα που αποτελούνται από medium-frequent terms, εκείνα που περιέχουν non-frequent terms και εκείνα των οποίων οι όροι είναι επιλεγμένοι τυχαία και από τις 3 κατηγορίες. Τα queries αποτελούνται από 1-5 όρους έτσι ώστε να υπάρχει ποικιλία στο μήκος τους, όπως συμβαίνει στις αναζητήσεις των χρηστών του πραγματικού κόσμου.

Κατηγορία	Όρια πλήθους εμφανίσεων όρων (Term Counts bounds)	Πλήθος όρων ανά κατηγορία
Frequent	35026-8599392$_{max}$	5188
Medium-Frequent	3000-35026	17507
Non-Frequent	1$_{min}$-3000	21654702

Πίνακας 5.1: Κατηγοριοποίηση των όρων με βάση την κατανομή του πλήθους εμφανίσεων τους

Τα queries δίνονται μαζικά ως είσοδος στο σύστημα, κάτι που εξυπηρετεί τους σκοπούς της πειραματικής διαδικασίας καθώς, όπως είπαμε, χρησιμοποιούνται queries διαφορετικών κατηγοριών. Οπότε, εξετάζουμε τα αποτελέσματα της μηχανής αναζήτησης για τα συγκεκριμένα 100 queries, τα οποία παρουσιάζονται στους πίνακες 5.2.α και 5.2.β, χωρισμένα ανάλογα με τις κατηγορίες στις οποίες ανήκουν:

Terms per Query	Frequent terms queries	Medium-Frequency terms queries
1	gameplay	danville
	powder	righthand
	social	spacer
	border	rosh
	passenger	eminem
2	gossip big	wicker officeholder
	slaves sound	stowage panther
	time fail	eucalyptus mcv
	finest ghost	hydroponics straightforward
	volume bathroom	gogh waterproof
3	look newspaper magnetic	roadmap paranormal kyrgyzstan
	large iran int	caliphate ballpark subcategory
	demonstrate typograph expand	macbeth abduction screensaver
	turner argument forever	rockstar palindrome blockquote
	jquery danger korn	morphology scent scheduler
4	greg citizenship fold eclipse	saddam tzu ivory athlon
	butter nebraska gone influenza	chm nippon filesystem hew
	diver prediction prison particle	engender ferrell megapixel battleground
	volleyball conduct trigger deposit	disastrous ortofon gilmor hvac
	sharp workout lifestyle spent	opr epox mahogan manuel
5	poison symbol cfa speed earliest	ifr photobook subcontractor backgroundcolor lemonade
	float empty smoking ask archbishop	canterbury puritan georgin severne snook
	concept tonight mighty bleed clarification	endocrinology weaponry sterility deuteronomy sprague
	literature receiver halloween minimum departure	ahrq preception succumb siskind pressroom
	ruth sheffield drop chief crawl	misinformation dexter fez napkin antimalware

Πίνακας 5.2.α: Τα επερωτήματα που παράχθηκαν για την πειραματική διαδικασία (frequent και medium-frequency όροι)

Terms per Query	Non-Frequent terms queries	Random-Frequency terms queries
1	ultraorthodox	surrender
	cloudwork	rollback
	westcoerealt	perception
	koumam	trademark
	blockread	linkshel
2	floorhead dindevan	tremble optomec
	autosuggest neurobiology	arrest claim
	aboundsther londontw	veterin signer
	gatekeeper colchicin	nial hashtag
	spiraldesign emilescom	atlas sebastiandelus
3	ciaedu prophyl headwater	significant aneurysm visasfailur
	pluginbud upmark overgrowth	wordcamp periplasmic remibfr
	hominem pvmintr leptospir	biobomb graceland micelleplec
	geocoder hadfield holomorph	whitesand osx chloroplast
	middleearth biisk coregroup	ambition terminal frameshift
4	bhaktisiddhnt aftereffect rainmaker pacifichost	sycamor pregnant quarter spaceflight
	firecracker clarkstown zenov loadmaster	purchaser covariant hirst attorney
	lipoprotein matchbox nobelstiftel psychoanalyst	broadband ethnographer southpoint narrowboat
	monkeesim lifestream toastfir suncreen	firm iverson tire kyoto
	shazam unanticipated framemaker paromomycin	humancomputer beaml martinson admire
5	homefront tuomarin supersport biogeography greenpoint	atomic hammer oldest threat brainwave
	classnoterequiredspan ashfield brainer sappatigharpam undersign	demolition paymentbook hong bluu fullscreen
	velasquez nezaslat beatport tahyesk babycenter	planetside inhabit dealer complaint uncommon
	bookkeeper widower haikouon homesick stargirl	thrust sent bromelain western veterin
	marriedin citriodorus homebound housework guideboat	ronaldinho shipment microcomputer melanion lineheight

Πίνακας 5.2.β: Τα επερωτήματα που παράχθηκαν για την πειραματική διαδικασία (non-frequent και random-frequency όροι)

Για κάθε ένα από τα παραπάνω επερωτήματα, θα εξαχθούν οι **1000** σελίδες με τα πιο υψηλά scores (σε φθίνουσα σειρά) με βάση τη μέθοδο του Cosine Similarity.

Original & Personalized PageRank – Προτιμήσεις Χρηστών

Εκτός όμως από τα παραπάνω queries και το λεκτικό περιεχόμενο των σελίδων, στα αποτελέσματα μας θα λάβουμε υπόψιν και το πόσο σημαντική είναι μία σελίδα της συλλογής σε σχέση με τις υπόλοιπες. Αυτό επιτυγχάνεται με την εφαρμογή του αλγορίθμου PageRank στο σύνολο των σελίδων. Όπως έχουμε δει στο κεφάλαιο 4, στην εργασία μας υλοποιήσαμε και μία παραλλαγή του παραπάνω αλγορίθμου, το Personalized PageRank, του οποίου η σημασία έγκειται στο γεγονός ότι οι χρήστες του συστήματος έχουν διαφορετικές προτιμήσεις όσον αφορά τις σελίδες που αναζητούν και αυτό παίζει ρόλο στα αποτελέσματα της αναζήτησης κάθε χρήστη. Γι αυτό το λόγο, υποθέσαμε ότι έχουμε 14 χρήστες στο σύστημά μας, ο καθένας από τους οποίους προτιμάει σελίδες σε ένα συγκεκριμένο εύρος μέσα στη συλλογή. Στον παρακάτω πίνακα φαίνεται, για κάθε χρήστη, το εύρος των προτιμώμενων σελίδων με βάση το id τους:

Χρήστες	Εύρος Προτιμώμενων Σελίδων (IDs)
1	1-100000
2	100001-200000
3	200001-300000
4	300001-400000
5	400001-500000
6	500001-600000
7	600001-700000
8	700001-800000
9	800001-900000
10	900001-1000000
11	**1000001-1050000**
12	**1100000-1120000**
13	**1500000-2000000**
14	**2000000-3000000**

Πίνακας 5.3: Προτιμήσεις των χρηστών

Για τους χρήστες 11,12,13 και 14 επιλέξαμε διαφορετικό σύνολο προτιμώμενων σελίδων σε σχέση με τους πρώτους 10 (όπου το πλήθος των σελίδων είναι 100000), έτσι ώστε να παρατηρήσουμε πώς θα επηρεάσει αυτό τα τελικά αποτελέσματα.

Συνδυασμός Μεθόδων Scoring

Τα αποτελέσματα της Cosine Similarity, που θα προκύψουν από τις αναζητήσεις για τα queries που παράχθηκαν, θα συνδυαστούν με τα αποτελέσματα του Original PageRank καθώς και με εκείνα του Personalized PageRank, για κάθε έναν από τους 14 χρήστες του συστήματος. Στην ουσία, αυτός ο συνδυασμός γίνεται για τις 1000 σελίδες που έχουν προκύψει από τον υπολογισμό της Cosine Similarity για κάθε επερώτημα. Στην έξοδο, όμως, θα έχουμε τις 100 πιο υψηλές τιμές για το ενοποιημένο score.

Θεωρήσαμε ότι θα έχει ενδιαφέρον να προσδώσουμε διαφορετικά ζεύγη βαρών στα 2 scores που χρησιμοποιούνται στον υπολογισμό του ενοποιημένου score κάθε φορά, γι αυτό και εκτελέσαμε υπολογισμούς με ζεύγη βαρών 25%-75%, 50%-50% και 75%-25%. Για παράδειγμα, όταν υπολογίζουμε το ενοποιημένο score Cosine Similarity-Personalized PageRank για κάποιον χρήστη με βάρη 75%-25%, σημαίνει ότι η Cosine Similarity λαμβάνει βάρος 75%, ενώ η τιμή του pagerank για εκείνο το χρήστη λαμβάνει βάρος 25%. Σε αυτήν την περίπτωση, η τιμή του Cosine Similarity θα επηρεάσει σε μεγαλύτερο βαθμό το τελικό αποτέλεσμα, κάτι που θα παρατηρήσουμε και στα αποτελέσματα στις επόμενες ενότητες του κεφαλαίου.

Συνοπτικά, τα πειράματα που διεξήχθησαν έχουν ως εξής:

- **Cosine Similarity**

  Αρχικά, εξήχθησαν οι 1000 σελίδες με τα πιο υψηλά scores για κάθε επερώτημα με βάση τη μέθοδο του Cosine Similarity.

- **Original PageRank**

  Στη συνέχεια, υπολογίστηκε η τιμή pagerank κάθε σελίδας με βάση τον πρωτότυπο αλγόριθμο PageRank.

- **Personalized PageRank**

  Για κάθε χρήστη, υπολογίστηκε η τιμή (personalized) pagerank για τις σελίδες της συλλογής μας, με βάση τα εύρη προτιμήσεων των 14 χρηστών.

- **Συνδυασμός Cosine Similarity – Original PageRank**

  Οι 1000 σελίδες που προέκυψαν από τη μέθοδο του Cosine Similarity, συνδυάστηκαν με τις τιμές του PageRank (μετά από κανονικοποίηση) και εξήχθησαν οι 100 σελίδες με το υψηλότερο ενοποιημένο score. Τα πειράματα διεξήχθησαν για 3 ζεύγη βαρών των μέτρων: 25%-75%, 50%-50% και 75%-25%. Π.χ. : $0.25*CosineSimilarity_{normalized} + 0.75*PageRank_{normalized}$

- **Συνδυασμός Cosine Similarity – Personalized PageRank**

  Όπως και για τον original αλγόριθμο PageRank, έτσι και για τον εξατομικευμένο (personalized), συνδυάζονται τα scores από το Cosine Similarity με τα scores του Personalized Pagerank **για κάθε χρήστη.** Όπως και πριν, δίνονται τα ίδια ζεύγη βαρών 25%-75%, 50%-50% και 75%-25%.

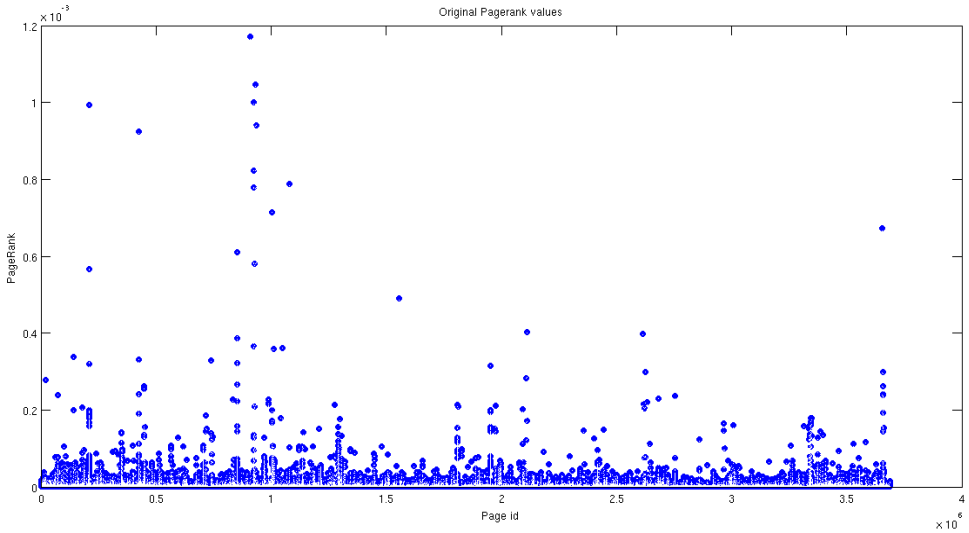
5.2 Αξιολόγηση Πειραματικών Αποτελεσμάτων

Στην παρούσα ενότητα θα παρουσιαστούν τα πειραματικά αποτελέσματα και θα επιχειρηθεί αξιολόγηση αυτών με βάση παραμέτρους όπως το είδος του score που χρησιμοποιείται, η κατηγορία του query για το οποίο εκτελείται αναζήτηση, οι προτιμήσεις των χρηστών κ.α.

5.2.1 Αποτελέσματα Original PageRank

Η υλοποίηση του πρωτότυπου αλγόριθμου PageRank αναλύθηκε στην ενότητα 4.3.2. Μετά το πέρας των επαναλήψεων, όλες οι σελίδες της συλλογής έχουν λάβει από μία τιμή η οποία συμβολίζει το μερίδιο "κύρους" της καθεμιάς. Το άθροισμα αυτών των τιμών, με βάση τη λογική του αλγορίθμου, είναι ίσο με 1. Το διάγραμμα 5.4 απεικονίζει τις τιμές των σελίδων της συλλογής με βάση τον πρωτότυπο αλγόριθμο PageRank.

Όπως παρατηρούμε από τα αποτελέσματα, οι τιμές του Original PageRank κυμαίνονται κατά συντριπτικό ποσοστό στην περιοχή από 10^{-8} έως 10^{-6}. Φυσικά, υπάρχουν τιμές οι οποίες είναι υψηλότερες, που σημαίνει ότι οι σελίδες στις οποίες αντιστοιχούν λαμβάνουν υψηλό μερίδιο "κύρους" και κατ' επέκταση είναι περισσότερο σημαντικές, άρα θα εμφανίζονται σε πιο υψηλές θέσεις στην κατάταξη των αποτελεσμάτων των διαφόρων αναζητήσεων. Ενδεικτικά, οι 20 υψηλότερες τιμές σελίδων για το Original PageRank παρουσιάζονται στον πίνακα 5.5.



Διάγραμμα 5.4: Pageid - Original PageRank

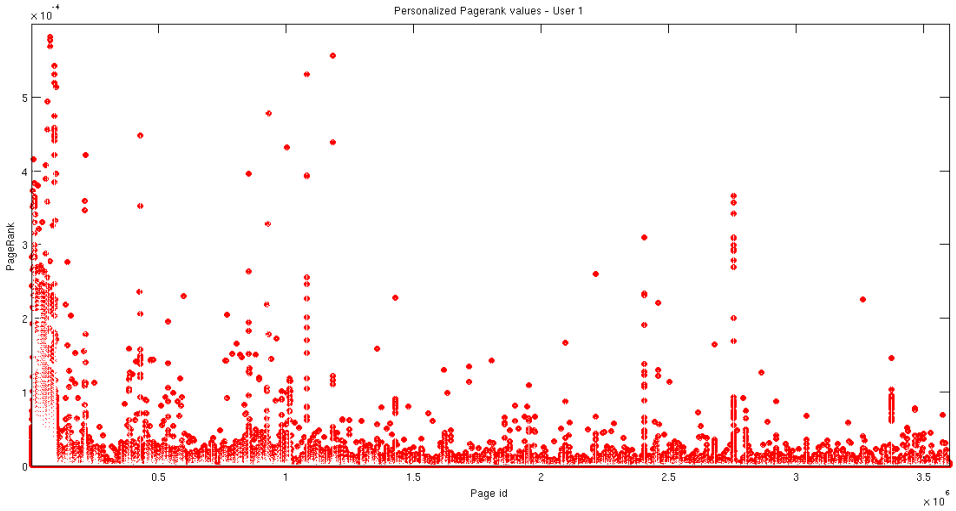
Κατάταξη	**Σελίδα (id)**	**Url**	**Pagerank**
1	908940	http://twitter.com/	0.0011721354
2	932557	http://twitter.com/privacy	0.0010456548
3	922857	http://twitter.com/about	0.0010007541
4	211006	http://creativecommons.org/licenses/by-sa/3.0/	9.923195E-4
5	935241	http://twitter.com/tos	9.39982E-4
6	426790	http://gmpg.org/xfn/11	9.252956E-4
7	922858	http://twitter.com/about/contact	8.219599E-4
8	1079041	http://www.about.com/	7.88102E-4
9	922897	http://twitter.com/account/complete	7.791811E-4
10	1003516	http://wiki.creativecommons.org/Public_domain	7.138857E-4
11	3655504	http://www.youtube.com/	6.7355373E-4
12	851593	http://support.twitter.com/groups/31-twitter-basics/topics/109-tweets-messages/articles/14606-what-is-a-direct-message-dm	6.1038794E-4
13	928937	http://twitter.com/jobs	5.805383E-4
14	210727	http://creativecommons.org/	5.676743E-4
15	1556353	http://www.city-data.com/	4.897024E-4
16	2111246	http://www.grainger.de/index.html	4.0177658E-4
17	2614471	http://www.myspace.com/	3.985013E-4

18	851358	http://support.twitter.com/	3.8740717E-4
19	926018	http://twitter.com/download	3.6541728E-4
20	1049217	http://www.1911encyclopedia.org/Main_Page	3.60E-004

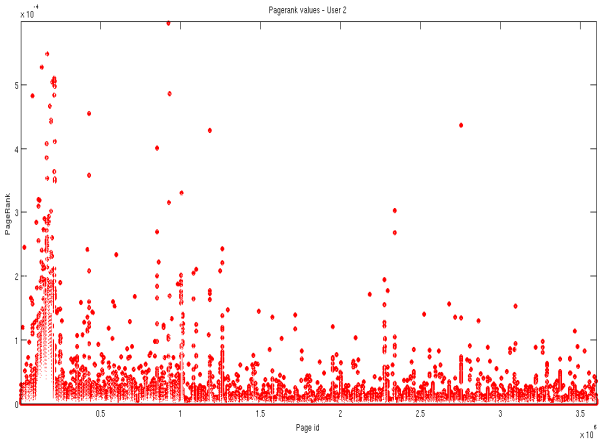
Πίνακας 5.5: Οι 20 σελίδες με τις υψηλότερες τιμές pagerank

5.2.2 Αποτελέσματα Personalized PageRank

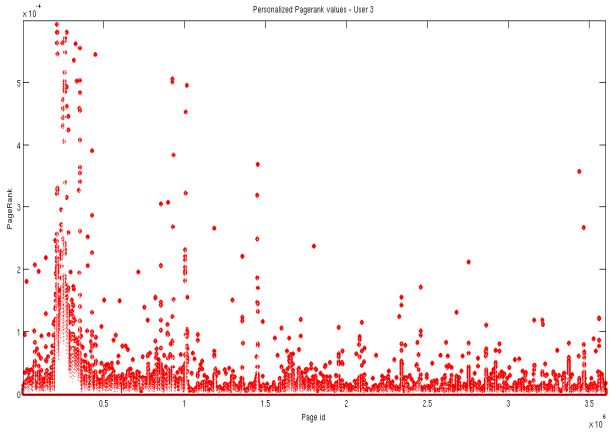
Τώρα, είμαστε έτοιμοι να παρουσιάσουμε τα διαγράμματα που αντιστοιχούν στις τιμές του Personalized PageRank για τους 14 χρήστες του συστήματος, των οποίων οι προτιμήσεις φαίνονται στον Πίνακα 5.3.



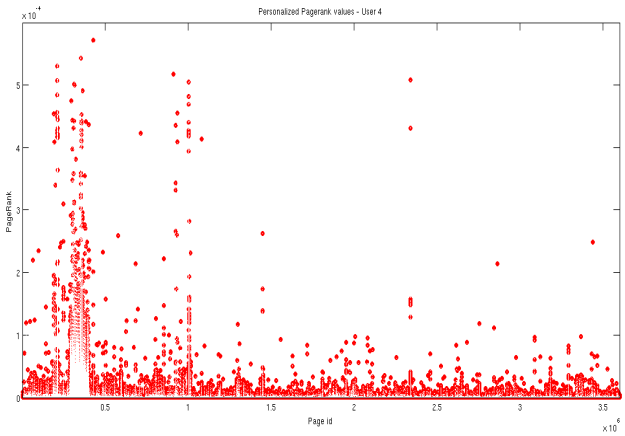
Διάγραμμα 5.6: PageId - Personalized PageRank, ***Χρήστης 1 (1-100000)***



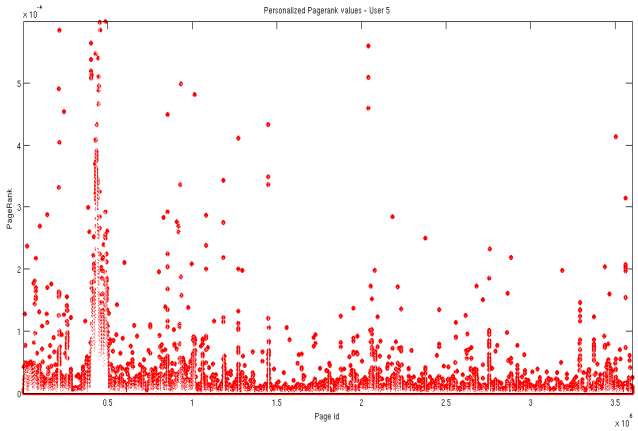
Διάγραμμα 5.7: PageId – Personalized PageRank, ***Χρήστης 2 (100001-200000)***



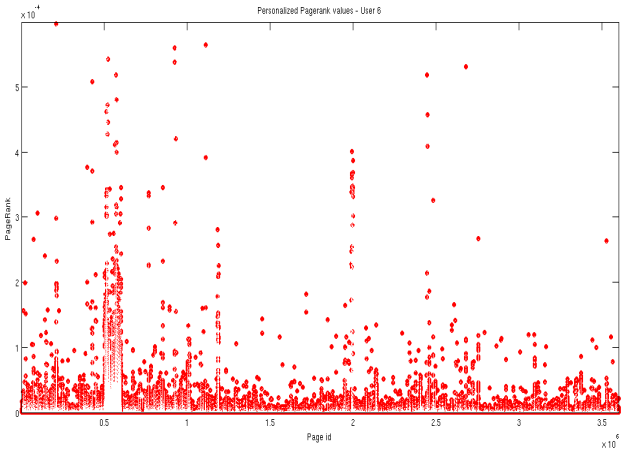
Διάγραμμα 5.8: PageId – Personalized PageRank, ***Χρήστης 3 (200001-300000)***



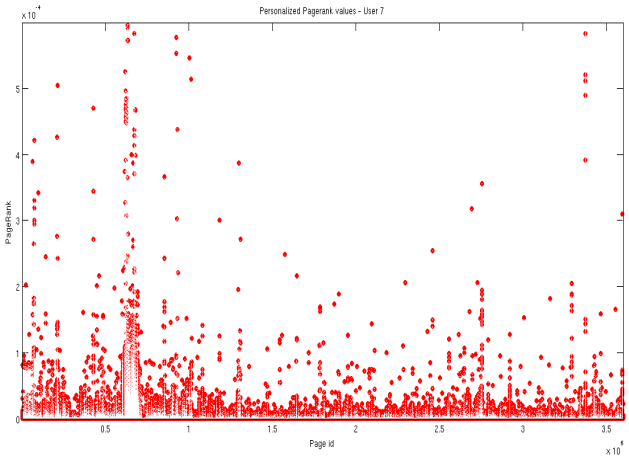
Διάγραμμα 5.9: PageId – Personalized PageRank, ***Χρήστης 4 (300001-400000)***



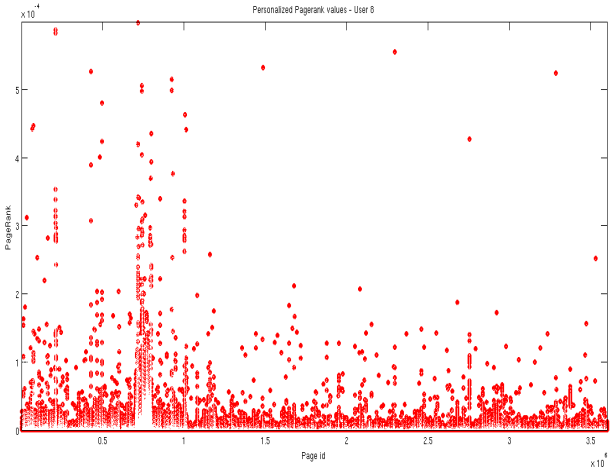
Διάγραμμα 5.10: PageId – Personalized PageRank, ***Χρήστης 5 (400001-500000)***



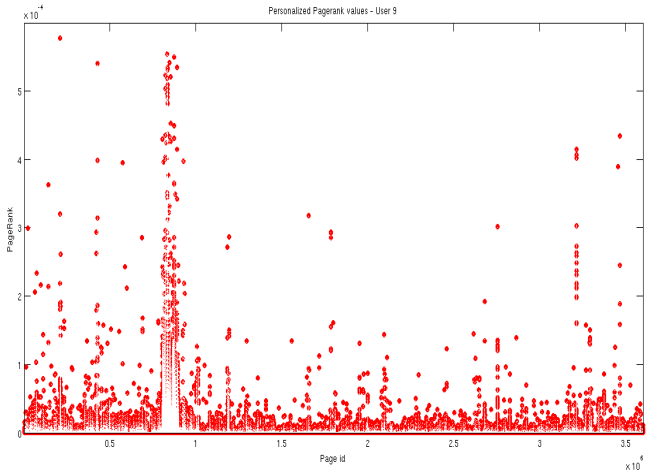
Διάγραμμα 5.11: PageId – Personalized PageRank, ***Χρήστης 6 (500001-600000)***



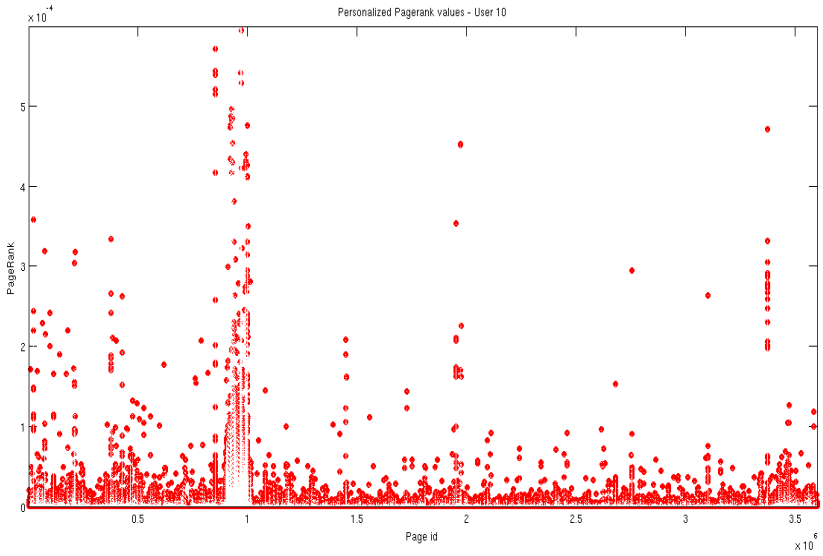
Διάγραμμα 5.12: PageId – Personalized PageRank, ***Χρήστης 7 (600001-700000)***



Διάγραμμα 5.13: PageId – Personalized PageRank, ***Χρήστης 8 (700001-800000)***



Διάγραμμα 5.14: PageId – Personalized PageRank, ***Χρήστης 9 (800001-900000)***



Διάγραμμα 5.15: PageId – Personalized PageRank,
Χρήστης 10 (900001-1000000)

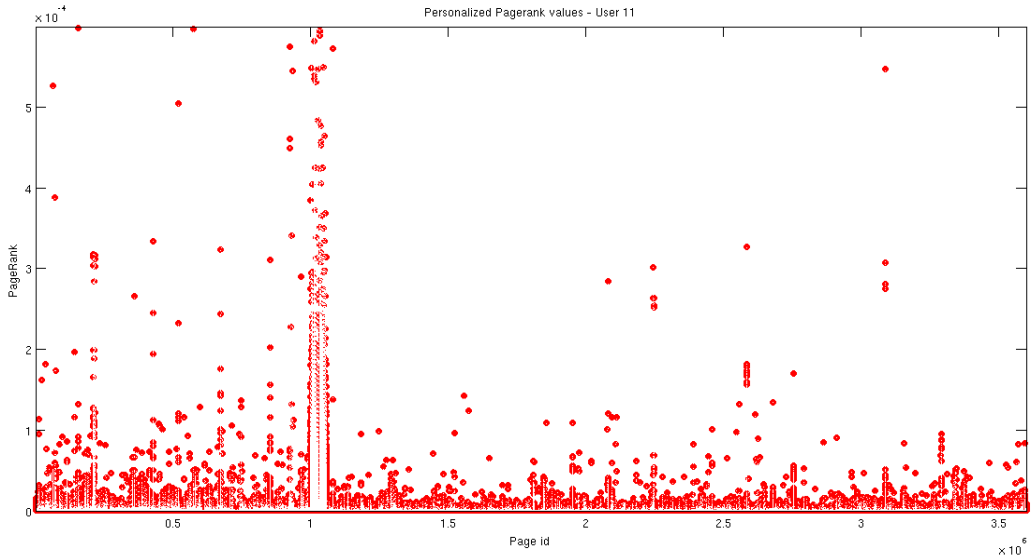
Στα διαγράμματα των πρώτων 10 χρηστών παρατηρούμε ότι οι τιμές του Personalized PageRank για τις σελίδες που βρίσκονται στο εύρος προτιμήσεων κάθε χρήστη, είναι υψηλότερες σε σχέση με τις υπόλοιπες σελίδες, οι οποίες δεν είχαν λάβει βάρος προτίμησης κατά τη διάρκεια των υπολογισμών (ενότητα 4.4.3). Επομένως, είναι φυσιολογικό οι τιμές σε εκείνες τις σελίδες να έχουν κατά βάση μικρότερο pagerank.

Επίσης, παρατηρούμε ότι στα 10 πρώτα διαγράμματα, τα pagerank scores των προτιμώμενων σελίδων κάθε χρήστη βρίσκονται περίπου στο ίδιο εύρος τιμών. Αυτό είναι λογικό να συμβαίνει,

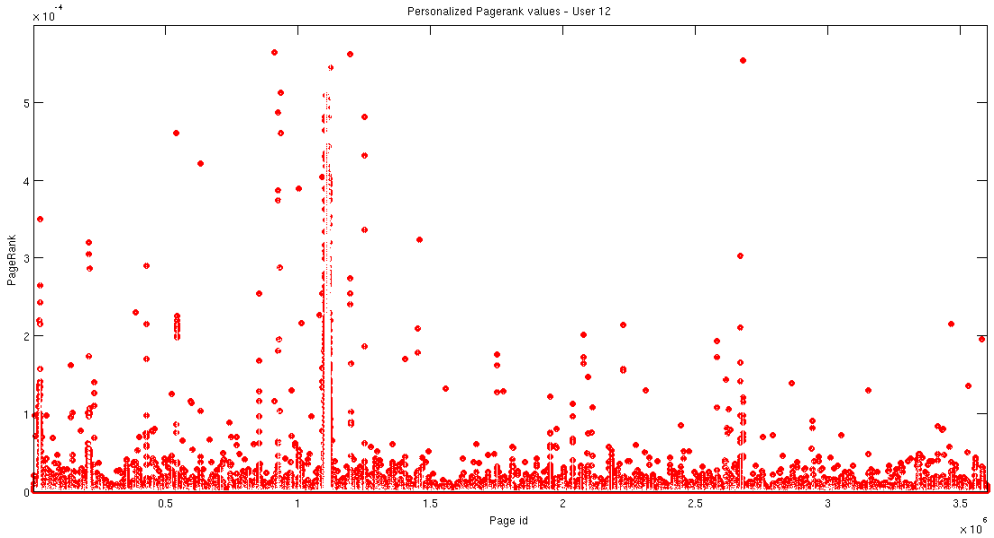
καθώς και για τους 10 χρήστες, το πλήθος των σελίδων προτίμησης είναι 100000, οπότε το βάρος προτίμησης w_{pi} για κάθε προνομιούχα σελίδα θα είναι ίσο με 1/100000. Ας θυμηθούμε σε αυτό το σημείο τον τύπο για το Personalized PageRank:

$$R_{k+1}(p_i) = d^* \left(\sum_{pj \in Bpi, |pj| \neq 0} \frac{Rk(pj)}{|pj|} + \sum_{pj, |pj|=0} \frac{Rk(pj)}{N} \right) + (1-d)^* \mathbf{w_{pi}} \quad ,$$

με w_{pi}=1/numOfPreferredPages



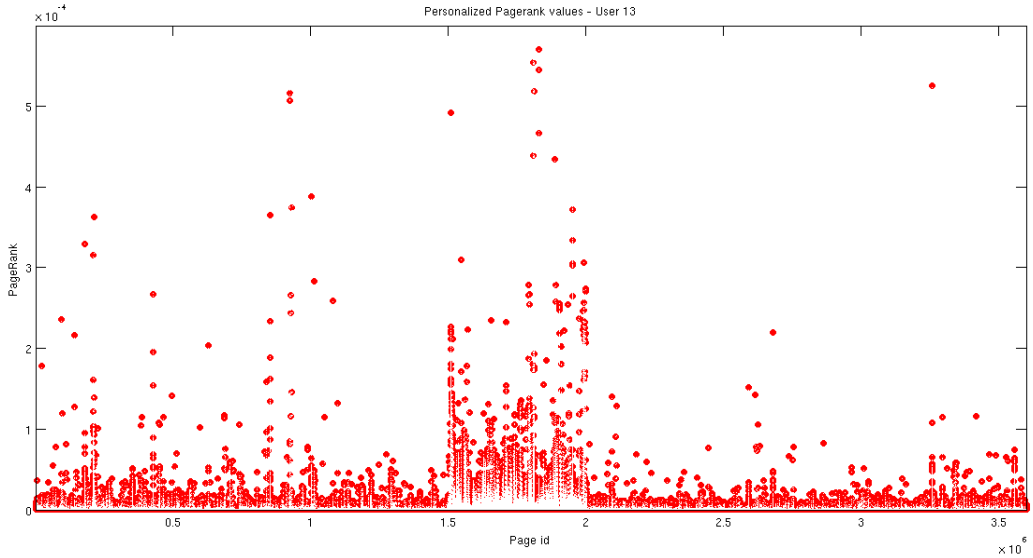
Διάγραμμα 5.16: PageId - Personalized PageRank, ***Χρήστης 11 (1000001-1050000)***



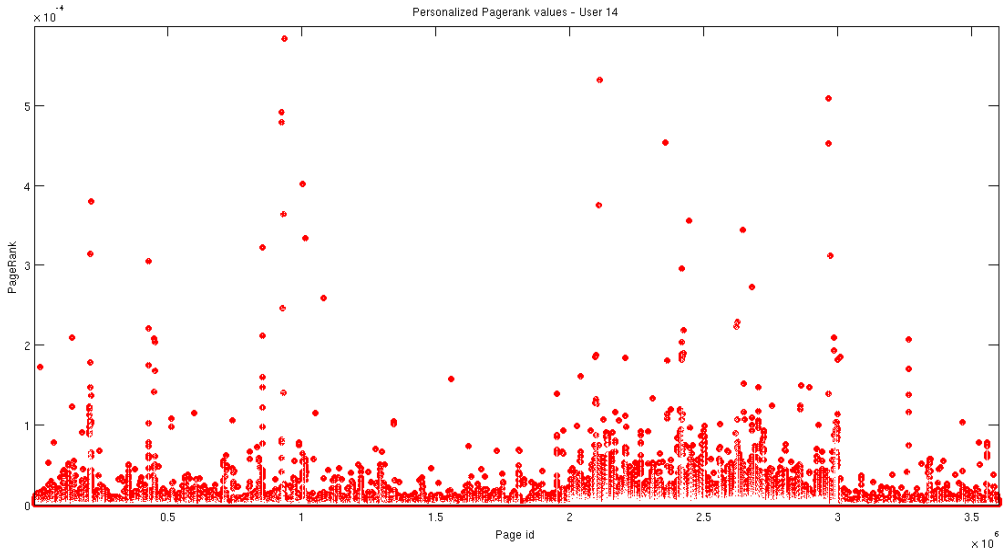
Διάγραμμα 5.17: PageId - Personalized PageRank, ***Χρήστης 12 (1100000-1120000)***

Όσον αφορά τους χρήστες 11 και 12, το πεδίο των προτιμήσεών τους είναι μικρότερο σε σχέση με

εκείνο των 10 πρώτων. Το πλήθος των σελίδων προτίμησης για τον χρήστη 11 είναι 50000, ενώ για το χρήστη 12 είναι 20000. Από τα αντίστοιχα διαγράμματα 5.16-17 παρατηρούμε ότι οι τιμές pagerank για τις προτιμώμενες σελίδες κάθε χρήστη είναι υψηλότερες, όσο μειώνεται το εύρος των προτιμήσεων. Δηλαδή, οι 20000 σελίδες που προτιμάει ο χρήστης 12 λαμβάνουν κατά κύριο λόγο υψηλότερο pagerank score απ' ότι οι 50000 σελίδες που αντιστοιχούν στον χρήστη 11, ενώ οι προτιμώμενες σελίδες και των 2 χρηστών έχουν μεγαλύτερη τιμή pagerank απ' ότι των 10 πρώτων, των οποίων το πλήθος των σελίδων προτίμησης είναι 100000. Το γεγονός αυτό εξηγείται από τον παραπάνω τύπο και την τιμή w_{pi}, η οποία αυξάνεται με τη μείωση των σελίδων προτίμησης.



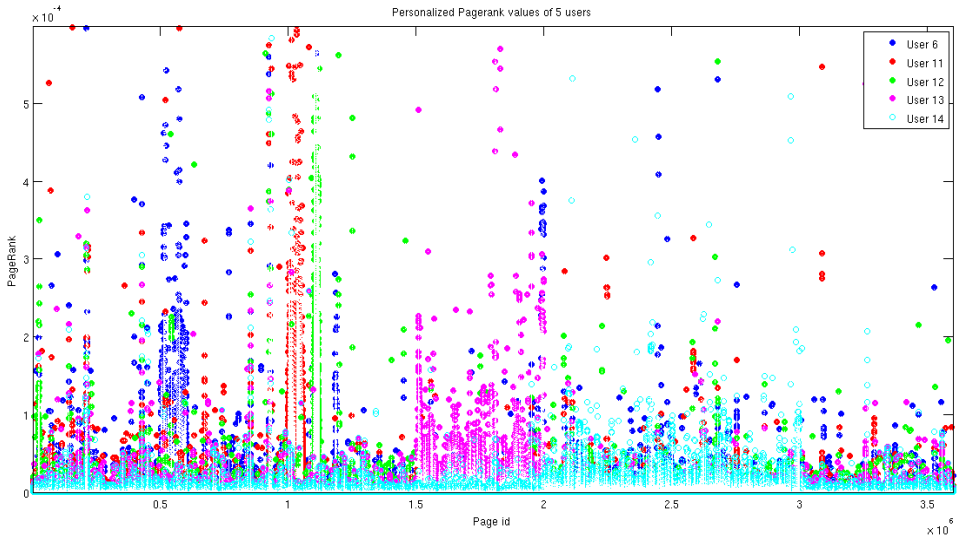
Διάγραμμα 5.18: PageId - Personalized PageRank, ***Χρήστης 13 (1500000-2000000)***



Διάγραμμα 5.19: PageId - Personalized PageRank, ***Χρήστης 14 (2000000-3000000)***

Οι χρήστες 13 και 14 έχουν πεδίο προτιμήσεων 500000 και 1000000 σελίδες αντίστοιχα. Οπότε η τιμή του w_{pi} θα είναι μικρότερη για εκείνους απ' ότι για τους 12 πρώτους χρήστες. Αυτό αντανακλάται και στις τιμές του pagerank για τις σελίδες προτίμησης, οι οποίες έχουν χαμηλότερες τιμές για αυτούς τους 2 χρήστες. Μεταξύ τους, οι προτιμώμενες σελίδες του χρήστη 11 λαμβάνουν υψηλότερο score απ' ότι εκείνες του χρήστη 12 (λόγω μεγαλύτερης τιμής του βάρους προτίμησης), όπως φαίνεται και στα διαγράμματα 5.19-20.

Στο παρακάτω κοινό διάγραμμα απεικονίζονται οι τιμές pagerank για 5 χρήστες με διαφορετικά πεδία προτιμήσεων:



*Διάγραμμα 5.20: PageId-Personalized PageRank, Χρήστης 6**(500001-600000)**, Χρήστης 11**(1000001-1050000)**, Χρήστης 12**(1100000-1120000)**, Χρήστης 13**(1500000-2000000)**, Χρήστης 14*(2000000-3000000)

5.2.3 Αποτελέσματα συνδυασμού Cosine Similarity - PageRank

Όπως έχουμε ήδη αναφέρει από την αρχή του κεφαλαίου, έγινε η παραγωγή 100 επερωτημάτων (queries) διαφορετικών κατηγοριών, όπως είναι η συχνότητα εμφάνισης των όρων που τα αποτελούν καθώς επίσης και το πλήθος των όρων του query (πίνακας 5.2). Για εκείνα τα queries, υπολογίστηκε η τιμή Cosine Similarity σε σχέση με τις σελίδες της συλλογής και εξήχθησαν σε πρώτη φάση τα 1000 καλύτερα αποτελέσματα, καθένα από τα οποία αντιστοιχεί σε μία σελίδα, για κάθε query. Έχουμε δει ότι το μέτρο Cosine Similarity δείχνει πόσο σχετικό είναι ένα query με ένα κείμενο-σελίδα, ενώ οι τιμές που λαμβάνει είναι από 0 έως 1, ανάλογα με την ομοιότητα.

Στη συνέχεια, εκείνα τα αποτελέσματα χρησιμοποιήθηκαν σε συνδυασμό με τις τιμές για το PageRank, ώστε να υπολογιστεί το ενοποιημένο score. Για το συνδυασμό των scores, χρειάστηκε να γίνει κανονικοποίηση των αποτελεσμάτων τόσο της Cosine Similarity, όσο και του PageRank. Οι υπολογισμοί διεξήχθησαν για κάθε query, για κάθε έναν από τους 14 χρήστες του συστήματος και για κάθε ζεύγος βαρών των scores.

Σε αυτή την ενότητα θα παρουσιαστούν ορισμένα αντιπροσωπευτικά παραδείγματα που αφορούν τόσο το Cosine Similarity score, όσο και το ενοποιημένο score για διαφορετικές περιπτώσεις.

- **Queries που αποτελούνται από frequent terms**

Για το query "**powder**":

	Cosine Similarity (Not Normalized)		**Cosine Similarity-Original Pagerank (50%-50%)**		**Cosine Similarity-Original Pagerank (25%-75%)**		**Cosine Similarity-Original Pagerank (75%-25%)**	
Κατά ταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	1458361	0.8676	2131041	0.5287	2131041	0.7644	1458361	0.7500
2	3528680	0.7853	1458361	0.5000	1052259	0.2576	3528680	0.6697
3	1257680	0.7670	3528680	0.4537	1458361	0.2500	1257680	0.6469
4	2534609	0.7397	1257680	0.4335	3528680	0.2376	2534609	0.6222
5	333624	0.6988	2534609	0.4220	2534609	0.2219	333624	0.5756
6	574095	0.6805	333624	0.3856	1257680	0.2200	574095	0.5571
7	3528726	0.6719	574095	0.3740	1811602	0.2184	2331920	0.5474
8	3531873	0.6691	2331920	0.3707	2129393	0.2049	3528726	0.5472
9	2331920	0.6689	746284	0.3705	1045736	0.1960	746284	0.5463
10	746284	0.6675	3528726	0.3662	333624	0.1957	3531873	0.5439
11	333626	0.6675	3531873	0.3636	746284	0.1948	333626	0.5421
12	3180547	0.6611	2129393	0.3624	2331920	0.1940	3180547	0.5356
13	574088	0.6513	333626	0.3621	2887525	0.1934	746277	0.5281
14	746277	0.6501	2887525	0.3585	574095	0.1910	574088	0.5267
15	2887525	0.6434	746277	0.3584	746277	0.1887	2887525	0.5235
16	746281	0.6369	3180547	0.3579	2655989	0.1867	2129393	0.5199
17	2129393	0.6354	574088	0.3538	3528726	0.1852	746281	0.5144
18	746287	0.6201	746281	0.3493	746289	0.1842	746287	0.4969
19	3647855	0.6192	746287	0.3400	746281	0.1841	3647855	0.4944
20	746282	0.6136	1811602	0.3359	3531873	0.1833	746282	0.4901

Πίνακας 5.21: Cosine Similarity-Original PageRank για όλα τα ζεύγη βαρών για το query "powder"

Ο παραπάνω πίνακας απεικονίζει τις τιμές Cosine Similarity και Cosine Similarity-Original

PageRank και για τα 3 ζεύγη βαρών. Για παράδειγμα, για το ζεύγος 50%-50%, ισχύει:

*Final_Score = 0.5*CosSim_norm + 0.5*Pagerank_norm*

Παρατηρούμε ότι υπάρχουν ανακατατάξεις στις σελίδες μετά την προσθήκη του PageRank στον υπολογισμό του score. Για παράδειγμα, η σελίδα με id *2131041* βρίσκεται πρώτη στην κατάταξη του ενοποιημένου score 50%-50% παρόλο που δε βρίσκεται καν στα πρώτα 20 αποτελέσματα με βάση την Cosine Similarity. Αυτό συμβαίνει διότι η συγκεκριμένη σελίδα έχει λάβει πολύ μεγαλύτερο μερίδιο pagerank από τις υπόλοιπες, οπότε και το ποσοστό του score που αντιστοιχεί στο pagerank θα είναι μεγαλύτερο, τόσο ώστε το συνολικό ενοποιημένο score να της δίνει την πρώτη θέση με το συγκεκριμένο ζεύγος βαρών. Το ίδιο φαινόμενο συμβαίνει για το ζεύγος βαρών 25%-75%, όπου η συγκεκριμένη σελίδα είναι ξανά πρώτη στην κατάταξη, κάτι που είναι φυσιολογικό καθώς τώρα το ποσοστό του pagerank είναι 75% έναντι του 25% που αντιστοιχεί στην Cosine Similarity. Επίσης, για το ίδιο ζεύγος βαρών, παρατηρούμε ότι η σελίδα με id *1811602* έχει ανέβει στην 7η θέση, σε αντίθεση με το ζεύγος 50%-50% όπου βρισκόταν στην 20η. Όσον αφορά το ζεύγος βαρών 75%-25%, παρατηρούμε ότι τα αποτελέσματα παρουσιάζουν ελάχιστες διαφορές σε σχέση με εκείνα που έχουν προκύψει μόνο με βάση την Cosine Similarity, κάτι που είναι λογικό καθώς το μέτρο της Cosine Similarity λαμβάνει ποσοστό 75%. Αντίθετα, το 25% που λαμβάνει το pagerank δεν είναι αρκετό ώστε να προκαλέσει σημαντικές ανακατατάξεις στα αποτελέσματα.

Στον πίνακα 5.22, παρουσιάζεται, για το ίδιο query, ο συνδυασμός Cosine Similarity-Personalized PageRank για το **χρήστη 1** (εύρος προτιμήσεων 1-100000). Όπως παρατηρούμε, για το ενοποιημένο score για το ζεύγος βαρών 50%-50%, τρεις σελίδες που εμπίπτουν στις προτιμήσεις του χρήστη 1 με ids 65449, 84293 και 63345 βρίσκονται στην πρώτη δεκάδα των αποτελεσμάτων στις θέσεις 1,5 και 8 αντίστοιχα. Η κατάταξη για τις υπόλοιπες σελίδες παρουσιάζει μικρές διαφορές σε σχέση με τον υπολογισμό μονάχα με βάση την Cosine Similarity, όπως μπορούμε να διακρίνουμε εύκολα από τον πίνακα. Όσο για το ζεύγος βαρών 25%-75%, είναι εμφανές πως οι προτιμήσεις του χρήστη επηρεάζουν τα αποτελέσματα σε τέτοιο σημείο ώστε 15 από τα 20 πρώτα αποτελέσματα για το query "powder" να αντιστοιχούν σε σελίδες με id από 1 έως 100000. Αντίθετα, για το ζεύγος βαρών 75%-25%, καμία από τις προτιμώμενες σελίδες του χρήστη 1 δεν βρίσκεται στα 20 πρώτα αποτελέσματα.

	Cosine Similarity (Not Normalized)		**Cosine Similarity-Personalized Pagerank User 1 (50%-50%)**		**Cosine Similarity-Personalized Pagerank User 1 (25%-75%)**		**Cosine Similarity-Personalized Pagerank User 1 (75%-25%)**	
Κατάταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	1458361	0.8676	65449	0.5730	65449	0.7865	1458361	0.7500
2	3528680	0.7853	1458361	0.5000	84293	0.6290	3528680	0.6653
3	1257680	0.7670	3528680	0.4449	63345	0.4557	1257680	0.6454
4	2534609	0.7397	1257680	0.4304	79590	0.4359	2534609	0.6171
5	333624	0.6988	84293	0.4265	63441	0.4154	333624	0.5748

6	574095	0.6805	2534609	0.4119	18432	0.4129	574095	0.5552
7	3528726	0.6719	3647855	0.4060	91128	0.4101	3528726	0.5464
8	3531873	0.6691	63345	0.4027	63337	0.3898	2331920	0.5433
9	2331920	0.6689	333624	0.3839	63339	0.3873	3531873	0.5433
10	746284	0.6675	574095	0.3703	63342	0.3838	746284	0.5419
11	333626	0.6675	3528726	0.3647	500	0.3629	333626	0.5416
12	3180547	0.6611	2331920	0.3626	3647855	0.2815	3180547	0.5350
13	574088	0.6513	3531873	0.3623	91134	0.2794	3647855	0.5305
14	746277	0.6501	746284	0.3617	81973	0.2761	574088	0.5249
15	2887525	0.6434	333626	0.3611	18431	0.2665	746277	0.5237
16	746281	0.6369	3180547	0.3568	1458361	0.2500	2887525	0.5169
17	2129393	0.6354	574088	0.3501	3528680	0.2245	746281	0.5100
18	746287	0.6201	746277	0.3496	16176	0.2217	2129393	0.5089
19	3647855	0.6192	2887525	0.3453	1257680	0.2154	746287	0.4925
20	746282	0.6136	2129393	0.3404	2534609	0.2067	746282	0.49

Πίνακας 5.22: Cosine Similarity-Personalized PageRank (χρήστης 1-προτιμήσεις σελίδων 1-100000) για όλα τα ζεύγη βαρών για το query "powder"

Ο παρακάτω πίνακας απεικονίζει τα αποτελέσματα για τον **χρήστη 11** (σελίδες 1000001-1050000):

	Cosine Similarity (Not Normalized)		**Cosine Similarity-Personalized Pagerank User 11 (50%-50%)**		**Cosine Similarity-Personalized Pagerank User 11 (25%-75%)**		**Cosine Similarity-Personalized Pagerank User 11 (75%-25%)**	
Κατά ταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	1458361	0.8676	1052259	0.7443	1052259	0.8722	1458361	0.7500
2	3528680	0.7853	1045736	0.5127	1045736	0.4859	3528680	0.6646
3	1257680	0.7670	1458361	0.5000	1458361	0.2500	1257680	0.6453
4	2534609	0.7397	3528680	0.4435	3528680	0.2224	2534609	0.6170
5	333624	0.6988	1257680	0.4303	1257680	0.2153	1052259	0.6165
6	574095	0.6805	2534609	0.4116	2534609	0.2063	333624	0.5743

7	3528726	0.6719	333624	0.3830	333624	0.1916	574095	0.5552
8	3531873	0.6691	574095	0.3702	574095	0.1853	3528726	0.5461
9	2331920	0.6689	3528726	0.3642	3528726	0.1822	3531873	0.5432
10	746284	0.6675	2331920	0.3624	2331920	0.1816	2331920	0.5432
11	333626	0.6675	3531873	0.3622	3531873	0.1812	746284	0.5417
12	3180547	0.6611	746284	0.3614	746284	0.1811	333626	0.5415
13	574088	0.6513	333626	0.3611	333626	0.1806	1045736	0.5396
14	746277	0.6501	3180547	0.3567	3180547	0.1784	3180547	0.5350
15	2887525	0.6434	574088	0.3500	574088	0.1752	574088	0.5248
16	746281	0.6369	746277	0.3494	746277	0.1751	746277	0.5236
17	2129393	0.6354	2887525	0.3449	2887525	0.1731	2887525	0.5167
18	746287	0.6201	746281	0.3402	2129393	0.1709	746281	0.5098
19	3647855	0.6192	2129393	0.3398	746281	0.1705	2129393	0.5086
20	746282	0.6136	746287	0.3285	746287	0.1647	746287	0.49

Πίνακας 5.23: Cosine Similarity-Personalized PageRank (χρήστης 11-προτιμήσεις σελίδων 1000001-1050000) για όλα τα ζεύγη βαρών για το query "powder"

Ο χρήστης 11 έχει ένα εύρος προτιμήσεων 50000 σελίδων, μικρότερο δηλαδή από εκείνο του χρήστη 1. Παρατηρούμε λοιπόν ότι για τα ζεύγη βαρών 50%-50% και 25%-75% οι 2 πρώτες σελίδες της κατάταξης αντιστοιχούν σε προτιμώμενες σελίδες του χρήστη. Με βάση το γεγονός αυτό, από τη μία επιβεβαιώνεται ότι λαμβάνουν μεγαλύτερο μερίδιο pagerank απ' ότι οι προτιμώμενες σελίδες του χρήστη 1 που είναι 50000 περισσότερες. Μία ακόμη απόδειξη αυτού είναι η εμφάνιση των δύο εκείνων σελίδων στα 15 πρώτα αποτελέσματα για το ζεύγος βαρών 75%-25%, για το οποίο στην προηγούμενη περίπτωση του χρήστη 1, δεν εμφανίστηκε κάποια από τις προτιμώμενες σελίδες του. Από την άλλη πλευρά όμως, λόγω του ότι οι προτιμώμενες σελίδες είναι λιγότερες, μειώνεται και η πιθανότητα εκείνες να βρίσκονται στα αποτελέσματα του ενοποιημένου score.

Για το **χρήστη 12** (εύρος προτιμήσεων 1100000-1120000), τα αποτελέσματα έχουν ως εξής:

	Cosine Similarity (Not Normalized)		**Cosine Similarity-Personalized Pagerank User 12 (50%-50%)**		**Cosine Similarity-Personalized Pagerank User 12 (25%-75%)**		**Cosine Similarity-Personalized Pagerank User 12 (75%-25%)**	
Κατάταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	1458361	0.8676	1110184	0.5658	1110184	0.7829	1458361	0.7500
2	3528680	0.7853	1458361	0.5000	1110183	0.5062	3528680	0.6656

3	1257680	0.7670	3528680	0.4454	1458361	0.2500	1257680	0.6455
4	2534609	0.7397	1257680	0.4307	3528680	0.2252	2534609	0.6176
5	333624	0.6988	2534609	0.4129	1257680	0.2159	333624	0.5745
6	574095	0.6805	333624	0.3833	2534609	0.2083	574095	0.5554
7	3528726	0.6719	1110183	0.3800	333624	0.1921	3528726	0.5463
8	3531873	0.6691	574095	0.3707	574095	0.1860	2331920	0.5437
9	2331920	0.6689	3528726	0.3644	2331920	0.1832	3531873	0.5433
10	746284	0.6675	2331920	0.3634	746284	0.1829	746284	0.5423
11	333626	0.6675	746284	0.3626	3528726	0.1826	333626	0.5416
12	3180547	0.6611	3531873	0.3624	3531873	0.1815	3180547	0.5350
13	574088	0.6513	333626	0.3612	333626	0.1808	574088	0.5250
14	746277	0.6501	3180547	0.3568	3180547	0.1786	746277	0.5242
15	2887525	0.6434	746277	0.3505	746277	0.1768	2887525	0.5176
16	746281	0.6369	574088	0.3505	574088	0.1759	746281	0.5104
17	2129393	0.6354	2887525	0.3466	2887525	0.1756	2129393	0.5100
18	746287	0.6201	2129393	0.3426	2129393	0.1752	746287	0.4929
19	3647855	0.6192	746281	0.3413	746281	0.1722	3647855	0.4917
20	746282	0.6136	746287	0.3297	746287	0.1664	746282	0.4861

Πίνακας 5.23: Cosine Similarity-Personalized PageRank (χρήστης 12-προτιμήσεις σελίδων 1100000-1120000) για όλα τα ζεύγη βαρών για το query "powder"

Το εύρος προτιμήσεων του χρήστη 12 είναι 20000 σελίδες, μικρότερο δηλαδή ακόμα και από εκείνο του χρήστη 11. Αυτό που αναμένεται είναι να υπάρχουν λίγες σελίδες στα αποτελέσματα του ενοποιημένου score, αλλά στις πρώτες θέσεις τις κατάταξης, καθώς λαμβάνουν μεγαλύτερο μερίδιο pagerank από το βήμα της τυχαίας αναπήδησης του αλγορίθμου, κάτι που γίνεται κατανοητό εάν θυμηθούμε τον μαθηματικό τύπο του Personalized PageRank:

$$R_{k+1}(p_i) = d^* \left(\sum_{pj \in Bpi, |pj| \neq 0} \frac{Rk(pj)}{|pj|} + \sum_{pj, |pj|=0} \frac{Rk(pj)}{N} \right) + (1-d)^* \mathbf{w_{pi}}$$

όπου w_{pi} = 1/numOfPreferredPages.

Οπότε εφόσον το βάρος w_{pi} επηρεάζεται από το πλήθος των προτιμώμενων σελίδων του χρήστη, για μικρότερο πλήθος έχουμε μεγαλύτερη τιμή του w_{pi}, άρα και η τιμή του pagerank είναι κατά βάση υψηλότερη για εκείνες τις σελίδες.

Παρατηρούμε λοιπόν για το χρήστη 12 ότι δύο από τις σελίδες που βρίσκονται στο εύρος

προτιμήσεων του, έχουν τοποθετηθεί στις δύο πρώτες θέσεις της κατάταξης για το ζεύγος βαρών 25%-75%, ενώ οι ίδιες σελίδες βρίσκονται στις θέσεις 1 και 7 αντίστοιχα για τα βάρη 50%-50%.

Όσο για τον **χρήστη 13** (εύρος προτιμήσεων 1500000-2000000), τα αποτελέσματα είναι:

	Cosine Similarity (Not Normalized)		**Cosine Similarity-Personalized Pagerank User 13 (50%-50%)**		**Cosine Similarity-Personalized Pagerank User 13 (25%-75%)**		**Cosine Similarity-Personalized Pagerank User 13 (75%-25%)**	
Κατάταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	1458361	0.8676	1811602	0.7268	1811602	0.8047	1458361	0.7500
2	3528680	0.7853	1766994	0.5092	1766994	0.7546	3528680	0.6697
3	1257680	0.7670	1458361	0.5000	1983479	0.5373	1811602	0.6489
4	2534609	0.7397	3528680	0.4537	1576308	0.3989	1257680	0.6461
5	333624	0.6988	1983479	0.4532	2227592	0.3952	2534609	0.6192
6	574095	0.6805	1257680	0.4317	1672179	0.3872	333624	0.5749
7	3528726	0.6719	1576308	0.4219	2131041	0.3536	574095	0.5560
8	3531873	0.6691	2534609	0.4161	1935566	0.3442	3528726	0.5471
9	2331920	0.6689	2129393	0.4010	1576257	0.3168	2331920	0.5450
10	746284	0.6675	333624	0.3843	1628637	0.3158	746284	0.5437
11	333626	0.6675	574095	0.3718	1935562	0.3156	3531873	0.5436
12	3180547	0.6611	3528726	0.3660	2129393	0.2628	333626	0.5418
13	574088	0.6513	2331920	0.3659	161609	0.2599	2129393	0.5392
14	746277	0.6501	746284	0.3653	1458361	0.2500	3180547	0.5352
15	2887525	0.6434	3531873	0.3630	391037	0.2420	574088	0.5256
16	746281	0.6369	333626	0.3615	1630634	0.2392	746277	0.5255
17	2129393	0.6354	3180547	0.3573	3528680	0.2377	2887525	0.5209
18	746287	0.6201	746277	0.3532	1576258	0.2353	746281	0.5118
19	3647855	0.6192	2887525	0.3532	161664	0.2348	746287	0.4943
20	746282	0.6136	574088	0.3516	1634950	0.2311	3647855	0.4926

Πίνακας 5.24: Cosine Similarity-Personalized PageRank (χρήστης 13-προτιμήσεις σελίδων 1500000-2000000) για όλα τα ζεύγη βαρών για το query "powder"

Στον παραπάνω πινακα παρατηρούμε ότι 4 σελίδες που έγκεινται στα ενδιαφέροντα του χρήστη 13, βρίσκονται μέσα στα 10 πρώτα αποτελέσματα για τα ζεύγη βαρών 50%-50% και 25%-75% στις θέσεις 1,2,5,7 και 1,2,3,4 αντίστοιχα. Λόγω του ότι το εύρος των προτιμήσεων του χρήστη είναι 500000, μεγαλύτερο δηλαδή από όσων έχουμε δει μέχρι τώρα, η πιθανότητα εμφάνισης εκείνων των σελίδων στα τελικά αποτελέσματα είναι μεγαλύτερη. Γι' αυτό, για το χρήστη 13 εμφανίζονται 4 προτιμώμενες σελίδες, ενώ για το χρήστη 12 εμφανίζονται 2 προτιμώμενες σελίδες στις πρώτες θέσεις για τα συγκεκριμένα ζεύγη βαρών. Για τα βάρη 75%-25% παρατηρούμε μόνο τη μία από τις τέσσερις σελίδες στα πρώτα αποτελέσματα, στη θέση 3, κάτι αρκετά ικανοποιητικό αν λάβουμε υπόψιν το βάρος της Cosine Similarity και το γεγονός ότι για το χρήστη 13 το βάρος $\mathbf{w_{pi}}$ είναι μικρότερο, λόγω του πλήθους των σελίδων που τον ενδιαφέρουν.

Για το query "powder", ο τελευταίος χρήστης του οποίου τα αποτελέσματα θα εξεταστούν είναι ο **χρήστης 14** (περιοχή προτιμήσεων 2000000-3000000):

	Cosine Similarity (Not Normalized)		**Cosine Similarity-Personalized Pagerank User 14 (50%-50%)**		**Cosine Similarity-Personalized Pagerank User 14 (25%-75%)**		**Cosine Similarity-Personalized Pagerank User 14 (75%-25%)**	
Κατά ταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	1458361	0.8676	2131041	0.5287	2131041	0.7644	1458361	0.7500
2	3528680	0.7853	1458361	0.5000	1458361	0.2500	3528680	0.6652
3	1257680	0.7670	3528680	0.4447	2534609	0.2295	1257680	0.6455
4	2534609	0.7397	1257680	0.4306	3528680	0.2242	2534609	0.6247
5	333624	0.6988	2534609	0.4271	1257680	0.2157	333624	0.5744
6	574095	0.6805	333624	0.3832	2655989	0.2124	574095	0.5553
7	3528726	0.6719	2331920	0.3755	2129393	0.2019	2331920	0.5497
8	3531873	0.6691	574095	0.3705	2331920	0.2012	3528726	0.5463
9	2331920	0.6689	3528726	0.3644	2887525	0.1942	3531873	0.5433
10	746284	0.6675	3531873	0.3623	333624	0.1920	746284	0.5421
11	333626	0.6675	746284	0.3621	574095	0.1857	333626	0.5416
12	3180547	0.6611	333626	0.3611	2930243	0.1849	3180547	0.5350
13	574088	0.6513	2129393	0.3604	2333340	0.1830	574088	0.5250
14	746277	0.6501	2887525	0.3590	3528726	0.1825	746277	0.5240
15	2887525	0.6434	3180547	0.3568	746284	0.1822	2887525	0.5238
16	746281	0.6369	574088	0.3503	3531873	0.1814	2129393	0.5189

17	2129393	0.6354	746277	0.3501	333626	0.1807	746281	0.5102
18	746287	0.6201	746281	0.3409	3180547	0.1786	746287	0.4927
19	3647855	0.6192	746287	0.3292	746277	0.1761	3647855	0.4916
20	746282	0.6136	2930243	0.3291	574088	0.1756	746282	0.4859

Πίνακας 5.25: Cosine Similarity-Personalized PageRank (χρήστης 14-προτιμήσεις σελίδων 2000000-3000000) για όλα τα ζεύγη βαρών για το query "powder"

Παρατηρούμε στον παραπάνω πίνακα ότι αρκετές προτιμώμενες σελίδες βρίσκονται στα πρώτα 20 αποτελέσματα για τα ζεύγη βαρών 50%-50%, 25-75%. Για τα βάρη 75%-25%, έχουμε 3 σελίδες προτίμησης στα αποτελέσματα. Το παραπάνω φαινόμενο είναι λογικό δεδομένου του μεγάλου εύρους προτιμήσεων (1000000 σελίδες) αναλογικά πάντα με το συνολικό πλήθος των σελίδων της συλλογής που εξετάζουμε (3689304 σελίδες).
Συνεχίζοντας την παρουσίαση των αποτελεσμάτων για queries που αποτελούνται από "frequent terms", θα εξετάσουμε ορισμένα αποτελέσματα για το query "**sharp workout lifestyle spent**":

	Cosine Similarity (Not Normalized)		**Cosine Similarity-Original Pagerank (50%-50%)**		**Cosine Similarity-Original Pagerank (25%-75%)**		**Cosine Similarity-Original Pagerank (75%-25%)**	
Κατά ταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	1946603	0.5667	551175	0.5763	551175	0.7882	1946603	0.7673
2	375022	0.5565	2032779	0.5759	2032779	0.6014	375022	0.7445
3	1946608	0.5510	1946603	0.5347	2032667	0.5614	1815929	0.7261
4	1815929	0.5505	375022	0.5146	410439	0.4415	1946608	0.7211
5	1814812	0.5414	1946614	0.4959	3074269	0.3498	1946614	0.7157
6	1946614	0.5409	2032667	0.4945	1111089	0.3337	1814812	0.7029
7	2495133	0.5356	1815929	0.4925	3074379	0.3306	2495133	0.6964
8	1809464	0.5299	1946608	0.4813	3074337	0.3255	1946605	0.6901
9	1946612	0.5290	1946605	0.4764	391997	0.3250	1946612	0.6830
10	1946605	0.5282	2495133	0.4707	1946603	0.3020	1809464	0.6812
11	1814385	0.5187	1814812	0.4691	3074377	0.2997	1814385	0.6604
12	3607349	0.5157	1946612	0.4603	375022	0.2847	3607349	0.6584
13	3607347	0.5142	1809464	0.4546	3074375	0.2821	3607347	0.6515

14	1813746	0.5096	3607349	0.4445	1946614	0.2762	1813746	0.6433
15	1946606	0.5053	1814385	0.4410	3074376	0.2751	3607327	0.6405
16	1946609	0.5040	3607327	0.4386	3074374	0.2736	1946606	0.6361
17	3607327	0.5037	410439	0.4379	1227948	0.2706	1946609	0.6328
18	1946617	0.4970	3607347	0.4343	3074452	0.2698	3607345	0.6242
19	3607948	0.4955	1813746	0.4294	3074344	0.2645	2497970	0.6211
20	3607345	0.4952	2497970	0.4290	1946605	0.2628	1946617	0.6197

Πίνακας 5.26: Cosine Similarity-Original PageRank για όλα τα ζεύγη βαρών για το query "sharp workout lifestyle spent"

Στον παραπάνω πίνακα, έχουν επισημανθεί οι σελίδες οι οποίες προέκυψαν από το συνδυασμό Cosine Similarity-Original PageRank και παράλληλα **δεν** βρίσκονται στα 20 πρώτα αποτελέσματα που έχουν εξαχθεί μόνο με βάση την τιμή της Cosine Similarity. Παρατηρούμε ότι για το ζεύγος βαρών 50%-50%, βρίσκονται 2 νέες σελίδες στις 2 πρώτες θέσεις, οι οποίες σαφώς κατατάσσονται εκεί λόγω του μεριδίου του pagerank που απέκτησαν. Όσο για το ζεύγος 25%-75%, στην πρώτη εικοσάδα αποτελεσμάτων βρίσκονται μόνο ελάχιστα από τα αποτελέσματα της πρώτης στήλης (Cosine Similarity), καθώς το 75% που αντιστοιχεί στο PageRank σε συνδυασμό με την υψηλή τιμή pagerank που κατέχουν εκ των προτέρων κάποιες σελίδες, επέφεραν την εμφάνιση στα αποτελέσματα, πολλών νέων σελίδων, για τις οποίες έχει υπολογιστεί υψηλό ενοποιημένο score.

Ακολουθούν πίνακες με τα αποτελέσματα για τους χρήστες **1** (1-100000),**11** (1000000-1050000) και **13** (1500000-2000000):

	Cosine Similarity (Not Normalized)		**Cosine Similarity-Personalized Pagerank User 1 (50%-50%)**		**Cosine Similarity-Personalized Pagerank User 1 (25%-75%)**		**Cosine Similarity-Personalized Pagerank User 1 (75%-25%)**	
Κατά ταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	1946603	0.5667	82778	0.7399	375005	0.7730	375022	0.8008
2	375022	0.5565	375022	0.6272	56796	0.7505	82778	0.7626
3	1946608	0.5510	82781	0.5882	82778	0.7171	1946603	0.7517
4	1815929	0.5505	375005	0.5460	82781	0.6367	1946608	0.7207
5	1814812	0.5414	82780	0.5172	82776	0.5898	1815929	0.7203
6	1946614	0.5409	56796	0.5153	82780	0.5818	1946614	0.7029
7	2495133	0.5356	1946603	0.5033	82782	0.5743	1814812	0.7025
8	1809464	0.5299	1815929	0.4809	62057	0.5391	2495133	0.6920
9	1946612	0.5290	1946608	0.4805	375022	0.4535	1809464	0.6809

10	1946605	0.5282	1946614	0.4704	1946603	0.2550	1946612	0.6796
11	1814385	0.5187	1814812	0.4684	98359	0.2452	1946605	0.6790
12	3607349	0.5157	82782	0.4634	1815929	0.2416	1814385	0.6600
13	3607347	0.5142	2495133	0.4619	1946608	0.2404	3607349	0.6546
14	1813746	0.5096	1946605	0.4542	1946614	0.2378	3607347	0.6515
15	1946606	0.5053	1809464	0.4540	1814812	0.2343	1813746	0.6429
16	1946609	0.5040	1946612	0.4536	2495133	0.2318	1946606	0.6349
17	3607327	0.5037	1814385	0.4400	1946605	0.2294	3607327	0.6326
18	1946617	0.4970	3607349	0.4369	1893560	0.2291	1946609	0.6324
19	3607948	0.4955	3607347	0.4343	1946612	0.2275	1946617	0.6192
20	3607345	0.4952	1813746	0.4286	1809464	0.2270	3607948	0.62

Πίνακας 5.27: Cosine Similarity-Personalized PageRank (χρήστης 1-προτιμήσεις σελίδων 1-100000) για όλα τα ζεύγη βαρών για το query "sharp workout lifestyle spent"

	Cosine Similarity (Not Normalized)		**Cosine Similarity-Personalized Pagerank User 11 (50%-50%)**		**Cosine Similarity-Personalized Pagerank User 11 (25%-75%)**		**Cosine Similarity-Personalized Pagerank User 11 (75%-25%)**	
Κατά ταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	1946603	0.5667	1946603	0.5101	422014	0.7542	1946603	0.7551
2	375022	0.5565	422014	0.5083	1045513	0.4692	375022	0.7365
3	1946608	0.5510	375022	0.4986	1023401	0.3579	1815929	0.7216
4	1815929	0.5505	1815929	0.4835	551175	0.3092	1946608	0.7208
5	1814812	0.5414	1946608	0.4807	2032779	0.2681	1946614	0.7057
6	1946614	0.5409	1946614	0.4759	1946603	0.2652	1814812	0.7026
7	2495133	0.5356	1814812	0.4686	375022	0.2606	2495133	0.6930
8	1809464	0.5299	2495133	0.4639	1946614	0.2462	1946605	0.6814
9	1946612	0.5290	1946605	0.4590	1815929	0.2455	1809464	0.6810
10	1946605	0.5282	1946612	0.4550	1946608	0.2406	1946612	0.6804
11	1814385	0.5187	1809464	0.4541	1946605	0.2367	1814385	0.6601
12	3607349	0.5157	1814385	0.4403	2495133	0.2347	3607349	0.6554

13	3607347	0.5142	3607349	0.4386	1814812	0.2345	3607347	0.6515
14	1813746	0.5096	3607347	0.4343	1946612	0.2297	1813746	0.6430
15	1946606	0.5053	1813746	0.4288	2032667	0.2274	1946606	0.6351
16	1946609	0.5040	3607327	0.4263	1809464	0.2272	3607327	0.6343
17	3607327	0.5037	1946606	0.4240	3607349	0.2217	1946609	0.6325
18	1946617	0.4970	1946609	0.4219	1814385	0.2204	1946617	0.6193
19	3607948	0.4955	3607345	0.4154	3607327	0.2182	3607345	0.6183
20	3607345	0.4952	2497970	0.4131	3607347	0.2172	3607948	0.6170

Πίνακας 5.28: Cosine Similarity-Personalized PageRank (χρήστης 11-προτιμήσεις σελίδων 1000000-1050000) για όλα τα ζεύγη βαρών για το query "sharp workout lifestyle spent"

	Cosine Similarity (Not Normalized)		**Cosine Similarity-Personalized Pagerank User 13 (50%-50%)**		**Cosine Similarity-Personalized Pagerank User 13 (25%-75%)**		**Cosine Similarity-Personalized Pagerank User 13 (75%-25%)**	
Κατά ταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	1946603	0.5667	1946603	0.7666	1946007	0.8743	1946603	0.8833
2	375022	0.5565	1946007	0.7578	1946575	0.8400	1946614	0.8130
3	1946608	0.5510	1946614	0.6905	1946005	0.8347	1815929	0.7803
4	1815929	0.5505	1946575	0.6801	1805724	0.7374	1946605	0.7763
5	1814812	0.5414	1946005	0.6766	1946603	0.6499	1946608	0.7373
6	1946614	0.5409	1946605	0.6489	1946614	0.5681	375022	0.7365
7	2495133	0.5356	1815929	0.6009	1946605	0.5215	1814812	0.7189
8	1809464	0.5299	1946612	0.5318	1604665	0.4353	1946612	0.7188
9	1946612	0.5290	1805724	0.5159	1604662	0.4341	1809464	0.6971
10	1946605	0.5282	1946608	0.5137	1815929	0.4215	2495133	0.6935
11	1814385	0.5187	1814812	0.5012	1604664	0.4156	1814385	0.6775
12	3607349	0.5157	375022	0.4985	1604663	0.4123	1813746	0.6597
13	3607347	0.5142	1893560	0.4866	1807715	0.3947	1946606	0.6578
14	1813746	0.5096	1809464	0.4864	551175	0.3781	3607349	0.6560
15	1946606	0.5053	1814385	0.4751	2032779	0.3549	3607347	0.6515

16	1946609	0.5040	1946606	0.4692	1946584	0.3482	1946007	0.6413
17	3607327	0.5037	2495133	0.4650	1893560	0.3478	1682542	0.6391
18	1946617	0.4970	1813746	0.4623	1946612	0.3448	3607327	0.6367
19	3607948	0.4955	1682542	0.4603	1928517	0.3322	1946617	0.6358
20	3607345	0.4952	1682433	0.4521	1682433	0.3318	1946609	0.6352

Πίνακας 5.29: Cosine Similarity-Personalized PageRank (χρήστης 13-προτιμήσεις σελίδων 1500000-2000000) για όλα τα ζεύγη βαρών για το query "sharp workout lifestyle spent". Με μωβ επισημαίνονται οι σελίδες που τις συναντάμε τόσο στα αποτελέσματα της Cosine Similarity όσο και του ενοποιημένου score και ***ταυτόχρονα*** *βρίσκονται στο πεδίο προτιμήσεων του χρήστη. Με κίτρινο επισημαίνονται όλες οι υπόλοιπες σελίδες προτίμησης του χρήστη, που βρίσκονται στα παραπάνω αποτελέσματα.*

Στον πίνακα 5.27 παρατηρούμε ότι τα αποτελέσματα για τον χρήστη 1 είναι τα αναμενόμενα, καθώς στην κατάταξη των σελίδων για το ενοποιημένο score και για ζεύγος βαρών 25%-75%, οι σελίδες προτίμησης του χρήστη καταλαμβάνουν πολλές από τις πρώτες θέσεις. Για το ζεύγος 50%-50%, οι σελίδες με id από 1 έως 100000 εμφανίζονται επίσης σε αρκετές θέσεις, ενώ για το ζεύγος βαρών 75%-25%, καταλαμβάνεται η 2η θέση από σελίδα προτίμησης του χρήστη 1.

Οι μόνες θέσεις της κατάταξης του πίνακα 5.28 που καταλαμβάνονται από σελίδες προτίμησης του χρήστη 11 είναι η 2η και η 3η θέση για το ζεύγος βαρών 25%-75%. Το μικρό εύρος προτιμήσεων του χρήστη (50000) είναι ο κυριότερος λόγος που μόνο 2 σελίδες βρίσκονται στα τελικά αποτελέσματα για το συγκεκριμένο query. Φυσικά, παίζουν ρόλο και τα αποτελέσματα που έχουν προκύψει από την Cosine Similarity και πιο συγκεκριμένα το αν και πόσες σελίδες προτίμησης του χρήστη βρίσκονται σε αυτά και με τι μέγεθος score.

Όπως μπορούμε εύκολα να κατανοήσουμε από τον πίνακα 5.29, τα αποτελέσματα του χρήστη 13 και για τα 3 ζεύγη βαρών περιλαμβάνουν στο μεγαλύτερο μέρος τους σελίδες που βρίσκονται στο εύρος προτιμήσεων του χρήστη. Ο κυριότερος λόγος που συμβαίνει αυτό είναι διότι στα πρώτα 20 αποτελέσματα με βάση την Cosine Similarity, τα 13 τυχαίνει να αποτελούνται από σελίδες προτίμησης του συγκεκριμένου χρήστη. Οπότε, είναι φυσιολογικό τα αποτελέσματα του ενοποιημένου score να περιλαμβάνουν στο μεγαλύτερο ποσοστό τους εκείνες τις σελίδες, καθώς αυτές λαμβάνουν σχετικά μεγάλες τιμές και από τα 2 μερικά scores, την Cosine Similarity και το Personalized PageRank για τον χρήστη 13. Για το ζεύγος βαρών 75%-25% παρατηρούμε ότι στις πρώτες θέσεις βρίσκονται οι σελίδες που έχουν αποκτήσει μεγάλη τιμή Cosine Similarity. Με μωβ επισημαίνονται εκείνες οι σελίδες που: 1) βρίσκονται ταυτόχρονα στα αποτελέσματα με βάση την Cosine Similarity και σε εκείνα του ενοποιημένου score και 2) έγκεινται στις προτιμήσεις του χρήστη 13. Όσο για το ζεύγος 25%-75%, στις πρώτες θέσεις βρίσκονται σελίδες προτίμησης του χρήστη, οι οποίες δε βρίσκονται στην πρώτη εικοσάδα αποτελεσμάτων της Cosine Similarity (κίτρινο χρώμα). Τα αποτελέσματα για το ζεύγος 50%-50% είναι μοιρασμένα, καθώς παρατηρούμε πως τις πρώτες θέσεις της κατάταξης αποτελούν σελίδες που είτε βρίσκονται στη στήλη της Cosine Similarity, είτε έχουν ανέβει ψηλά λόγω μεγάλης τιμής του pagerank για το συγκεκριμένο χρήστη.

- **Queries που αποτελούνται από medium-frequency terms**

Για το query "**endocrinology weaponry sterility deuteronomy sprague**", θα παρουσιάσουμε και θα συγκρίνουμε τα αποτελέσματα που απεικονίζονται στους παρακάτω πίνακες για τους χρήστες **1** (1-100000),**12** (1100000-1120000) και **14** (2000000-3000000):

	Cosine Similarity (Not Normalized)		**Cosine Similarity-Personalized Pagerank User 1 (50%-50%)**		**Cosine Similarity-Personalized Pagerank User 1 (25%-75%)**		**Cosine Similarity-Personalized Pagerank User 1 (75%-25%)**	
Κατά ταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	482189	0.4007	1277873	0.5109	1277873	0.7554	482189	0.7500
2	719748	0.3955	482189	0.5000	69914	0.3546	719748	0.7388
3	678186	0.3646	719748	0.4925	74450	0.3511	678186	0.6777
4	1848333	0.3586	74450	0.4716	482189	0.2500	1848333	0.6595
5	1031203	0.3425	678186	0.4601	719748	0.2463	1031203	0.6235
6	1811261	0.3291	1848333	0.4411	678186	0.2425	1811261	0.5942
7	2738713	0.3139	1031203	0.4156	1848333	0.2228	74450	0.5922
8	1029217	0.3042	1811261	0.3962	77368	0.2104	2738713	0.5613
9	74450	0.3016	2738713	0.3743	1031203	0.2078	1029217	0.5402
10	678187	0.2986	1029217	0.3602	29603	0.2061	678187	0.5282
11	1023432	0.2978	678187	0.3525	1811261	0.1982	1023432	0.5262
12	3163860	0.2953	1023432	0.3508	24275	0.1925	3163860	0.5208
13	1213317	0.2895	3163860	0.3474	2738713	0.1873	1213317	0.5083
14	1817910	0.2856	1213317	0.3389	24188	0.1858	1817910	0.4997
15	1688426	0.2849	1688426	0.3350	1029217	0.1801	1688426	0.4996
16	1848325	0.2761	1817910	0.3332	678187	0.1768	1848325	0.4801
17	1562419	0.2744	1848325	0.3215	1023432	0.1754	1562419	0.4756
18	3681021	0.2735	1562419	0.3173	3163860	0.1739	3681021	0.4735
19	2216408	0.2730	2216408	0.3168	1688426	0.1704	2216408	0.4733
20	2216428	0.2730	3681021	0.3159	1213317	0.1696	2216428	0.4725

Πίνακας 5.30: Cosine Similarity-Personalized PageRank (χρήστης 1-προτιμήσεις σελίδων 1-100000) για όλα τα ζεύγη βαρών για το query "endocrinology weaponry sterility deuteronomy sprague"

	Cosine Similarity (Not Normalized)		Cosine Similarity-Personalized Pagerank User 12 (50%-50%)		Cosine Similarity-Personalized Pagerank User 12 (25%-75%)		Cosine Similarity-Personalized Pagerank User 12 (75%-25%)	
Κατά ταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	482189	0.4007	1115595	0.5156	1115595	0.7578	482189	0.7500
2	719748	0.3955	482189	0.5000	1115645	0.5378	719748	0.7388
3	678186	0.3646	1115645	0.4971	482189	0.2500	678186	0.6722
4	1848333	0.3586	719748	0.4925	719748	0.2463	1848333	0.6587
5	1031203	0.3425	678186	0.4489	678186	0.2257	1031203	0.6234
6	1811261	0.3291	1848333	0.4394	1848333	0.2202	1811261	0.5942
7	2738713	0.3139	1031203	0.4156	1031203	0.2078	2738713	0.5613
8	1029217	0.3042	1811261	0.3961	1811261	0.1981	1029217	0.5402
9	74450	0.3016	2738713	0.3742	2738713	0.1871	74450	0.5346
10	678187	0.2986	1029217	0.3602	1029217	0.1801	678187	0.5280
11	1023432	0.2978	74450	0.3564	74450	0.1783	1023432	0.5262
12	3163860	0.2953	678187	0.3520	1103571	0.1771	3163860	0.5208
13	1213317	0.2895	1023432	0.3508	678187	0.1761	1213317	0.5082
14	1817910	0.2856	3163860	0.3472	1023432	0.1754	1817910	0.4997
15	1688426	0.2849	1213317	0.3388	3163860	0.1737	1688426	0.4985
16	1848325	0.2761	1817910	0.3331	1213317	0.1694	1848325	0.4792
17	1562419	0.2744	1688426	0.3327	1277873	0.1677	1562419	0.4754
18	3681021	0.2735	1848325	0.3198	1688426	0.1670	3681021	0.4733
19	2216408	0.2730	1562419	0.3170	1817910	0.1666	2216408	0.4725
20	2216428	0.2730	3681021	0.3156	1848325	0.1604	2216428	0.4723

Πίνακας 5.31: Cosine Similarity-Personalized PageRank (χρήστης 12-προτιμήσεις σελίδων 1100000-1120000) για όλα τα ζεύγη βαρών για το query "endocrinology weaponry sterility deuteronomy sprague"

	Cosine Similarity (Not Normalized)		Cosine Similarity-Personalized Pagerank User 14 (50%-50%)		Cosine Similarity-Personalized Pagerank User 14 (25%-75%)		Cosine Similarity-Personalized Pagerank User 14 (75%-25%)	
Κατά ταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	482189	0.4007	1277873	0.5109	1277873	0.7554	482189	0.7500
2	719748	0.3955	482189	0.5000	2671260	0.6858	719748	0.7388
3	678186	0.3646	719748	0.4925	2671267	0.2505	678186	0.6743
4	1848333	0.3586	2671260	0.4667	482189	0.2500	1848333	0.6595
5	1031203	0.3425	678186	0.4532	719748	0.2463	1031203	0.6235
6	1811261	0.3291	1848333	0.4412	678186	0.2322	1811261	0.5942
7	2738713	0.3139	1031203	0.4156	1848333	0.2228	2738713	0.5632
8	1029217	0.3042	1811261	0.3962	1031203	0.2078	1029217	0.5402
9	74450	0.3016	2738713	0.3780	1811261	0.1982	74450	0.5347
10	678187	0.2986	2671267	0.3654	2738713	0.1928	678187	0.5281
11	1023432	0.2978	1029217	0.3602	2216408	0.1911	1023432	0.5262
12	3163860	0.2953	74450	0.3566	2671312	0.1811	3163860	0.5208
13	1213317	0.2895	678187	0.3522	1029217	0.1801	1213317	0.5082
14	1817910	0.2856	1023432	0.3508	74450	0.1786	1817910	0.4997
15	1688426	0.2849	3163860	0.3474	2232076	0.1770	1688426	0.4996
16	1848325	0.2761	1213317	0.3389	678187	0.1764	2216408	0.4835
17	1562419	0.2744	2216408	0.3373	1023432	0.1754	2671267	0.4803
18	3681021	0.2735	1688426	0.3350	3163860	0.1739	1848325	0.4801
19	2216408	0.2730	1817910	0.3332	1688426	0.1704	2671312	0.4763
20	2216428	0.2730	2671312	0.3287	1213317	0.1695	2216428	0.4757

Πίνακας 5.32: Cosine Similarity-Personalized PageRank (χρήστης 14-προτιμήσεις σελίδων 2000000-3000000) για όλα τα ζεύγη βαρών για το query "endocrinology weaponry sterility deuteronomy sprague"

Παρατηρώντας τον πίνακα 5.30 για τον χρήστη 1, γίνεται κατανοητό ότι μόνο στην κατάταξη των πρώτων 20 αποτελεσμάτων για το ζεύγος βαρών 25%-75% βρίσκονται προτιμώμενες σελίδες του χρήστη, εκτός εκείνης με id *74450*, η οποία βρίσκεται ούτως ή άλλως στην πρώτη εικοσάδα

αποτελεσμάτων με βάση την Cosine Similarity, επομένως λαμβάνει μερίδιο και από την Cosine Similarity αλλά και από το PageRank λόγω του ότι είναι στο εύρος 1-100000. Όσο για το χρήστη 12, του οποίου τα αποτελέσματα απεικονίζονται στον πίνακα 5.31, μόνο 2 σελίδες βρίσκονται ψηλά στην κατάταξη για τα ζεύγη βαρών 50%-50% και 25%-75%. Το συγκεκριμένο φαινόμενο είναι λογικό να συμβαίνει, λόγω του μικρού εύρους προτιμήσεων του χρήστη (20000 σελίδες). Όμως, παρά τις λίγες εμφανίσεις σελίδων προτίμησης, εκείνες βρίσκονται στις πρώτες θέσεις των αποτελεσμάτων (1η,3η για 50%-50% και 1η,2η για 25%-75%), κάτι που προκαλείται από το ίδιο το εύρος προτίμησης του χρήστη, καθώς όσο πιο μικρό είναι, τόσο μεγαλύτερο είναι το βάρος προτίμησης που ανατίθεται στις προκείμενες σελίδες. Αντίθετα, από τον πίνακα 5.32 που απεικονίζει τα αποτελέσματα του χρήστη 14, παρατηρούμε ότι αρκετές σελίδες προτίμησης του χρήστη βρίσκονται στην πρώτη εικοσάδα των αποτελεσμάτων του ενοποιημένου score (κυρίως για τα βάρη 50%-50% και 25-75%). Από την άλλη πλευρά, εκείνες οι σελίδες δεν καταλαμβάνουν πολλές από τις πρώτες θέσεις για το ζεύγος βαρών 50%-50% (μόνο την 4η), ενώ για το ζεύγος βαρών καταλαμβάνονται η 2η και η 3η θέση και οι υπόλοιπες θέσεις με προτιμώμενες σελίδες βρίσκονται από την 10η θέση και κάτω.

- **Queries που αποτελούνται από non-frequent terms**

Για το query "**firecracker clarkstown zenov loadmaster**", παρουσιάζονται τα αποτελέσματα για τους χρήστες **8** (700000-800000), **11** (1000000-1050000) και **14** (2000000-3000000):

	Cosine Similarity (Not Normalized)		**Cosine Similarity-Personalized Pagerank User 8 (50%-50%)**		**Cosine Similarity-Personalized Pagerank User 8 (25%-75%)**		**Cosine Similarity-Personalized Pagerank User 8 (75%-25%)**	
Κατάταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	1820591	0.3032	3372442	0.5174	3372442	0.7587	1820591	0.7500
2	646418	0.2112	1820591	0.5000	1820591	0.2500	646418	0.5136
3	667529	0.2050	646418	0.3427	646418	0.1718	667529	0.4976
4	2463364	0.1972	667529	0.3319	667529	0.1662	2463364	0.4775
5	1455530	0.1925	2463364	0.3186	2463364	0.1597	1455530	0.4652
6	3246800	0.1702	1455530	0.3101	1455530	0.1551	3246800	0.4079
7	3372522	0.1374	3246800	0.2720	3246800	0.1362	3372522	0.3237
8	177833	0.1326	3372522	0.2158	706354	0.1249	177833	0.3113
9	992907	0.1250	177833	0.2075	2547748	0.1109	992907	0.2917
10	468810	0.1244	992907	0.1945	3372522	0.1079	468810	0.2904
11	646420	0.1163	468810	0.1937	177833	0.1038	3372442	0.2762

12	2989019	0.1153	646420	0.1800	744817	0.0986	646420	0.2695
13	336866	0.1151	336866	0.1788	992907	0.0973	336866	0.2670
14	667528	0.1142	2989019	0.1779	468810	0.0970	2989019	0.2668
15	873065	0.1140	667528	0.1761	336866	0.0906	667528	0.2640
16	3472154	0.1130	873065	0.1756	646420	0.0904	873065	0.2634
17	868210	0.1110	3472154	0.1739	2989019	0.0889	3472154	0.2609
18	42702	0.1102	868210	0.1706	667528	0.0883	868210	0.2558
19	1550580	0.1100	42702	0.1695	873065	0.0878	42702	0.2538
20	1825172	0.1098	1550580	0.1689	3472154	0.0870	1550580	0.2533

Πίνακας 5.33: Cosine Similarity-Personalized PageRank (χρήστης 8-προτιμήσεις σελίδων 700000-800000) για όλα τα ζεύγη βαρών για το query "firecracker clarkstown zenov loadmaster"

	Cosine Similarity (Not Normalized)		**Cosine Similarity-Personalized Pagerank User 11 (50%-50%)**		**Cosine Similarity-Personalized Pagerank User11 (25%-75%)**		**Cosine Similarity-Personalized Pagerank User 11 (75%-25%)**	
Κατά ταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	1820591	0.3032	1033376	0.5087	1033376	0.7543	1820591	0.7500
2	646418	0.2112	1820591	0.5000	3372442	0.4642	646418	0.5137
3	667529	0.2050	646418	0.3428	1820591	0.2500	667529	0.4977
4	2463364	0.1972	667529	0.3320	646418	0.1719	2463364	0.4776
5	1455530	0.1925	3372442	0.3211	667529	0.1663	1455530	0.4652
6	3246800	0.1702	2463364	0.3187	2463364	0.1598	3246800	0.4079
7	3372522	0.1374	1455530	0.3101	1455530	0.1551	3372522	0.3237
8	177833	0.1326	3246800	0.2721	1036789	0.1527	177833	0.3113
9	992907	0.1250	3372522	0.2158	3246800	0.1362	992907	0.2917
10	468810	0.1244	177833	0.2075	1001307	0.1274	468810	0.2904
11	646420	0.1163	992907	0.1945	1035373	0.1187	646420	0.2696
12	2989019	0.1153	468810	0.1937	1035374	0.1087	2989019	0.2668

13	336866	0.1151	646420	0.1801	3372522	0.1079	336866	0.2665
14	667528	0.1142	2989019	0.1779	177833	0.1038	667528	0.2640
15	873065	0.1140	336866	0.1779	992907	0.0973	873065	0.2634
16	3472154	0.1130	667528	0.1762	468810	0.0970	1033376	0.2630
17	868210	0.1110	873065	0.1756	646420	0.0906	3472154	0.2609
18	42702	0.1102	3472154	0.1739	336866	0.0892	868210	0.2558
19	1550580	0.1100	868210	0.1706	2989019	0.0889	42702	0.2539
20	1825172	0.1098	42702	0.1696	667528	0.0884	1550580	0.2533

Πίνακας 5.34: Cosine Similarity-Personalized PageRank (χρήστης 11-προτιμήσεις σελίδων 1000000-1050000) για όλα τα ζεύγη βαρών για το query "firecracker clarkstown zenov loadmaster"

	Cosine Similarity (Not Normalized)		**Cosine Similarity-Personalized Pagerank User 14 (50%-50%)**		**Cosine Similarity-Personalized Pagerank User 14 (25%-75%)**		**Cosine Similarity-Personalized Pagerank User 14 (75%-25%)**	
Κατά ταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	1820591	0.3032	3372442	0.5174	3372442	0.7587	1820591	0.7500
2	646418	0.2112	1820591	0.5000	2516239	0.6611	646418	0.5143
3	667529	0.2050	2516239	0.4482	2735860	0.6057	667529	0.4980
4	2463364	0.1972	2735860	0.4039	2516248	0.4266	2463364	0.4877
5	1455530	0.1925	646418	0.3440	1820591	0.2500	1455530	0.4652
6	3246800	0.1702	2463364	0.3389	2090519	0.2245	3246800	0.4105
7	3372522	0.1374	667529	0.3327	2463364	0.1902	3372522	0.3237
8	177833	0.1326	1455530	0.3101	2929097	0.1790	177833	0.3113
9	992907	0.1250	2516248	0.2934	646418	0.1738	992907	0.2918
10	468810	0.1244	3246800	0.2773	2385651	0.1735	468810	0.2916
11	646420	0.1163	3372522	0.2158	2675550	0.1717	3372442	0.2762
12	2989019	0.1153	177833	0.2076	667529	0.1674	646420	0.2702
13	336866	0.1151	468810	0.1962	2408364	0.1581	2989019	0.2697

14	667528	0.1142	992907	0.1945	1455530	0.1551	336866	0.2669
15	873065	0.1140	2989019	0.1837	2001683	0.1442	667528	0.2644
16	3472154	0.1130	646420	0.1813	2290949	0.1441	873065	0.2635
17	868210	0.1110	336866	0.1787	3246800	0.1441	3472154	0.2609
18	42702	0.1102	667528	0.1769	2704341	0.1439	868210	0.2559
19	1550580	0.1100	873065	0.1757	2617916	0.1253	42702	0.2545
20	1825172	0.1098	3472154	0.1739	2463405	0.1133	1550580	0.25

Πίνακας 5.35: Cosine Similarity-Personalized PageRank (χρήστης 14-προτιμήσεις σελίδων 2000000-3000000) για όλα τα ζεύγη βαρών για το query "firecracker clarkstown zenov loadmaster"

Στα αποτελέσματα της Cosine Similarity για το παραπάνω query, παρατηρούμε ότι οι τιμές του συγκεκριμένου μέτρου για τις υψηλές στην κατάταξη σελίδες, είναι χαμηλές σε σχέση με εκείνες για τα queries που αποτελούνται από frequent όρους, ή ακόμα και από όρους μέσης συχνότητας (medium frequency terms). Όπως έχουμε εξηγήσει σε προηγούμενα κεφάλαια, η Cosine Similarity λαμβάνει τιμές στο πεδίο μεταξύ 0 και 1, ανάλογα με την ομοιότητα ενός επερωτήματος σε σχέση με ένα κείμενο-σελίδα. Επομένως, οι χαμηλές τιμές για non-frequent όρους είναι φυσιολογικές, καθώς, κατά βάση, οι πιο ασυνήθιστοι όροι δεν έχουν ιδιαίτερες ομοιότητες με το λεκτικό περιεχόμενο των σελίδων. Αντίθετα, οι όροι που εμφανίζονται πιο συχνά σε σελίδες, είναι πιθανό να έχουν μεγάλη ομοιότητα με το περιεχόμενο τους, άρα και μεγαλύτερη τιμή της Cosine Similarity. Το παραπάνω φαινόμενο συμβαίνει γενικά στα αποτελέσματα και σχετίζεται με τη συχνότητα εμφάνισης των όρων που αποτελούν τα queries που χρησιμοποιήσαμε για να εξάγουμε τα πειραματικά αποτελέσματα. Παρακάτω δίνονται ενδεικτικά οι τιμές της Cosine Similarity για τις σελίδες που βρίσκονται στην πρώτη θέση των εκάστοτε αποτελεσμάτων, με queries από διαφορετικές κατηγορίες:

Query	**Category**	**Cosine Similarity**
newspaper magnet	frequent terms	0.7605625
gameplay	frequent terms	0.6172898
finest ghost	frequent terms	0.5552499
engender ferrell megapixel battleground	Medium-frequency terms	0.41949612
opr epox mahogan manuel	Medium-frequency terms	0.5041754
rockstar palindrome blockquote	Medium-frequency terms	0.54922664
ciaedu prophyl headwater	Non-frequent terms	0.2686905
velasquez nezaslat beatport tahyesk babycenter	Non-frequent terms	0.3111712
firecracker clarkstown zenov loadmaster	Non-frequent terms	0.30323973

Για ευνόητους λόγους δεν παρουσιάζονται τα αποτελέσματα από όλα τα queries στο παρόν κείμενο, όμως είναι διαθέσιμα προς εξέταση.

Από τον πίνακα 5.33 παρατηρούμε πως μόνο 2 σελίδες που έγκεινται στο πεδίο προτιμήσεων του

χρήστη 8, βρίσκονται στα πρώτα 20 αποτελέσματα (8η και 12η θέση) του ενοποιημένου score και αυτό μόνο για το ζεύγος 25%-75%. Το γεγονός αυτό μπορεί να οφείλεται σε 2 σημαντικούς παράγοντες: 1) οι σελίδες προτιμήσεων του χρήστη δεν έχουν λάβει υψηλές τιμές Cosine Similarity ή/και 2) οι συγκεκριμένες σελίδες που βρίσκονται στα αποτελέσματα της Cosine Similarity τυχαίνει να μην έχουν τόσο μεγάλο pagerank ώστε να αποκτήσουν εξίσου υψηλό ενοποιημένο score που θα τις φέρει στις πρώτες θέσεις των αποτελεσμάτων.

Όσον αφορά τον χρήστη 11, παρατηρούμε (πίνακας 5.34) ότι παρόλο που μόνο μία σελίδα προτίμησης βρίσκεται στις πρώτες 5 θέσεις για τα ζεύγη βαρών 50%-50% και 25%-75% (κάτι που είναι λογικό λόγω του μικρού εύρους προτίμησης των 50000 σελίδων), εκείνη έχει αναρριχηθεί στην πρώτη θέση της κατάταξης εξαιτίας του υψηλής τιμής pagerank (μικρό εύρος προτιμήσεων-μεγάλο βάρος προτίμησης $\mathbf{w_{pi}}$)!

Αντίθετα από τα αποτελέσματα του χρήστη 11, για τον χρήστη 14 μπορούμε να δούμε από τον πίνακα 5.35 ότι η κατάταξη για τα ζεύγη βαρών 50%-50% και 25%-75% περιλαμβάνει αρκετές σελίδες προτίμησης του χρήστη, κάτι που συμβαίνει λόγω του μεγάλου εύρους προτιμήσεών του (1000000 σελίδες). Όσο για το ζεύγος 75%-25%, η κατάταξη περιέχει 2 σελίδες προτίμησης του χρήστη, οι οποίες όμως βρίσκονται σε εκείνες τις θέσεις λόγω της τιμής Cosine Similarity (μωβ επισήμανση στον πίνακα) και όχι του μεριδίου pagerank που έχουν αποκτήσει από την εφαρμογή του αλγορίθμου για τον συγκεκριμένο χρήστη.

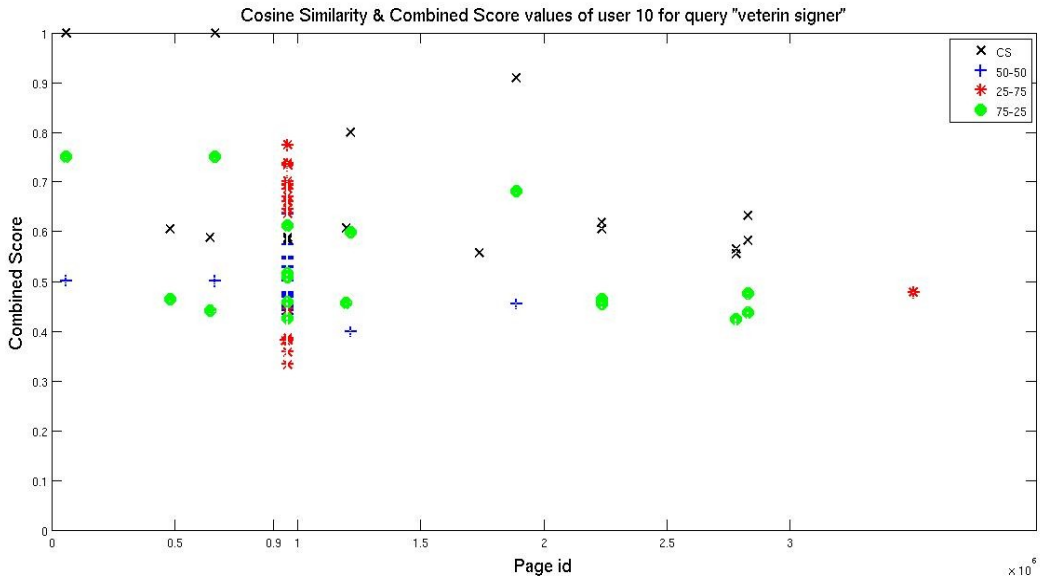
- **Queries που αποτελούνται από random-frequency terms**

Θα παρουσιάσουμε ένα τελευταίο παράδειγμα που αφορά τα πειραματικά αποτελέσματα για το query "**veterin signer**", για τους χρήστες **10** (900000-1000000), **11** (1000000-1050000), **12** (1100000-1120000), **13** (1500000-2000000) και **14** (2000000-3000000) για όλα τα ζεύγη βαρών, καθώς και διαγράμματα που απεικονίζουν τα scores για όλα τα ζεύγη βαρών:

	Cosine Similarity (Not Normalized)		**Cosine Similarity-Personalized Pagerank User 10 (50%-50%)**		**Cosine Similarity-Personalized Pagerank User 10 (25%-75%)**		**Cosine Similarity-Personalized Pagerank User 10 (75%-25%)**	
Κατάταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	59107	0.6960	959073	0.6363	958963	0.7750	59107	0.7505
2	664495	0.6958	958964	0.5739	958964	0.7381	664495	0.7503
3	1886802	0.6485	958963	0.5500	958969	0.7336	1886802	0.6822
4	1214302	0.5912	958947	0.5457	958967	0.7016	959073	0.6123
5	2828588	0.5041	958969	0.5296	958970	0.6951	1214302	0.5996
6	2235131	0.4965	959075	0.5275	958957	0.6933	959076	0.5169
7	1197349	0.4909	958967	0.5012	958947	0.6865	959057	0.5078

8	2235123	0.4899	59107	0.5011	958966	0.6855	2828588	0.4754
9	480466	0.4899	664495	0.5010	958936	0.6710	480466	0.4648
10	959073	0.4809	958936	0.4784	959073	0.6602	2235131	0.4644
11	644304	0.4807	958970	0.4739	958930	0.6602	959070	0.4587
12	959076	0.4789	958957	0.4703	958986	0.6461	1197349	0.4572
13	2828598	0.4781	958966	0.4687	959075	0.6368	2235123	0.4542
14	959057	0.4779	1886802	0.4554	3501599	0.4786	644304	0.4409
15	2780767	0.4690	959070	0.4516	959070	0.4446	2828598	0.4382
16	2780771	0.4690	959076	0.4492	959052	0.3853	959053	0.4252
17	2780770	0.4689	958930	0.4460	959076	0.3815	2780767	0.4242
18	2780768	0.4688	958986	0.4413	950684	0.3813	2780771	0.4242
19	1738715	0.4648	959057	0.4329	959057	0.3581	2780770	0.4240
20	2780764	0.4638	1214302	0.3998	959056	0.3330	2780768	0.4239

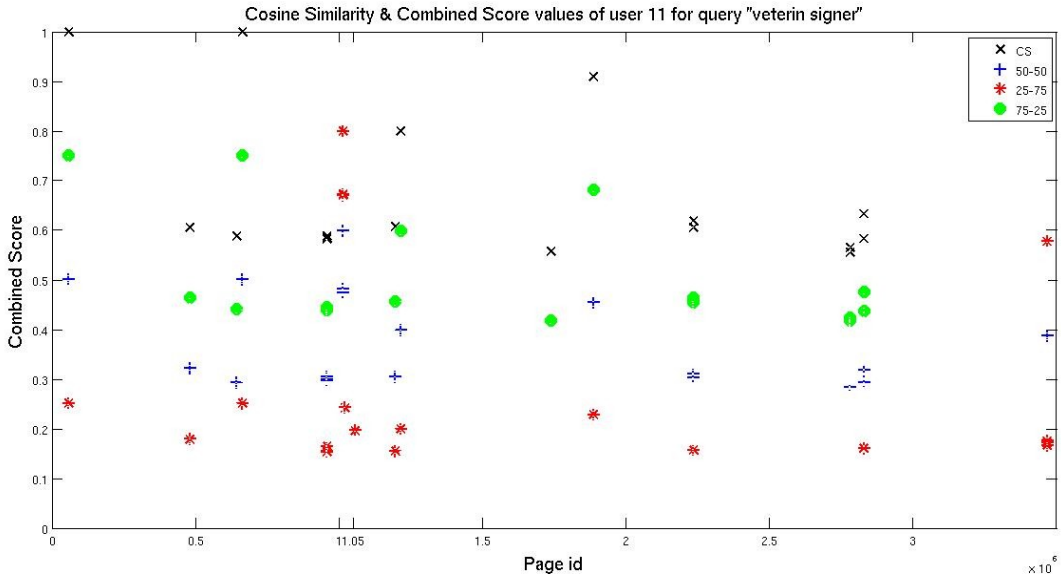
Πίνακας 5.36: Cosine Similarity-Personalized PageRank (χρήστης 10-προτιμήσεις σελίδων 900000-1000000) για όλα τα ζεύγη βαρών για το query "veterin signer"



Διάγραμμα 5.37: Cosine Similarity-Personalized PageRank (χρήστης 10-προτιμήσεις σελίδων 900000-1000000) για όλα τα ζεύγη βαρών για το query "veterin signer"

	Cosine Similarity (Not Normalized)		Cosine Similarity-Personalized Pagerank User 11 (50%-50%)		Cosine Similarity-Personalized Pagerank User 11 (25%-75%)		Cosine Similarity-Personalized Pagerank User 11 (75%-25%)	
Κατά ταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	59107	0.6960	1014842	0.5997	1014842	0.7999	59107	0.7505
2	664495	0.6958	59107	0.5009	1014556	0.6728	664495	0.7502
3	1886802	0.6485	664495	0.5008	1014553	0.6711	1886802	0.6822
4	1214302	0.5912	1014553	0.4810	3466961	0.5769	1214302	0.5996
5	2828588	0.5041	1014556	0.4740	664495	0.2514	2828588	0.4753
6	2235131	0.4965	1886802	0.4552	59107	0.2514	2235131	0.4643
7	1197349	0.4909	1214302	0.3998	1020920	0.2427	480466	0.4635
8	2235123	0.4899	3466961	0.3874	1886802	0.2283	1197349	0.4566
9	480466	0.4899	480466	0.3214	1214302	0.1999	2235123	0.4542
10	959073	0.4809	2828588	0.3178	1056348	0.1962	959076	0.4444
11	644304	0.4807	2235131	0.3104	480466	0.1793	959073	0.4432
12	959076	0.4789	1197349	0.3056	3466981	0.1750	644304	0.4409
13	2828598	0.4781	959076	0.3043	3466966	0.1713	959057	0.4399
14	959057	0.4779	2235123	0.3028	3466967	0.1668	2828598	0.4381
15	2780767	0.4690	959073	0.2982	959076	0.1641	2780767	0.4242
16	2780771	0.4690	959057	0.2971	2828588	0.1604	2780771	0.4242
17	2780770	0.4689	644304	0.2940	2235131	0.1565	2780770	0.4240
18	2780768	0.4688	2828598	0.2930	1197349	0.1547	2780768	0.4239
19	1738715	0.4648	2780767	0.2828	959057	0.1544	1738715	0.4184
20	2780764	0.4638	2780771	0.2828	959073	0.1531	2780764	0.4179

Πίνακας 5.38: Cosine Similarity-Personalized PageRank (χρήστης 11-προτιμήσεις σελίδων 1000000-1050000) για όλα τα ζεύγη βαρών για το query "veterin signer"

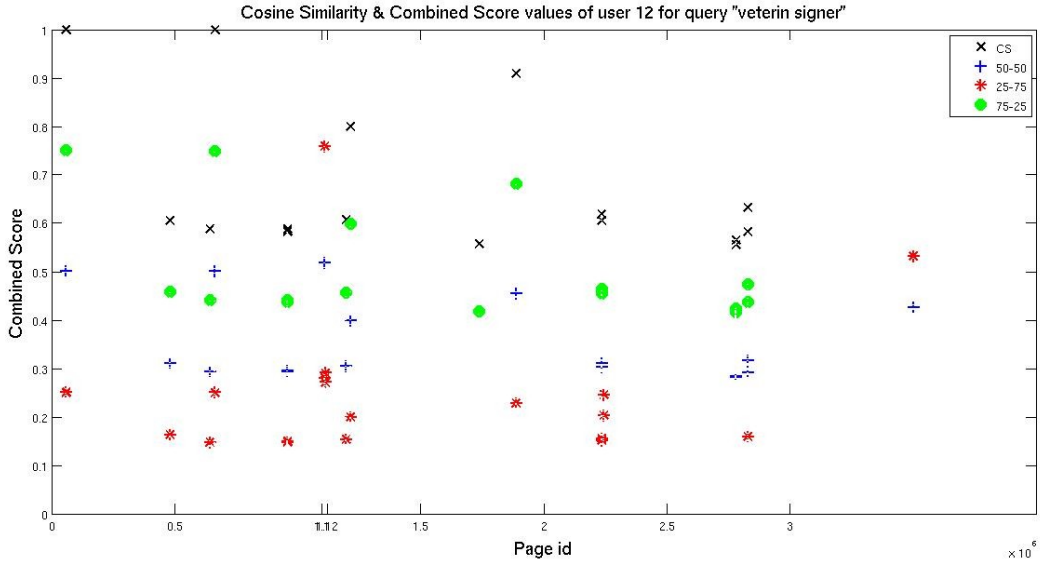


Διάγραμμα 5.39: Cosine Similarity-Personalized PageRank (χρήστης 11-προτιμήσεις σελίδων 1000000-1050000) για όλα τα ζεύγη βαρών για το query "veterin signer"

	Cosine Similarity (Not Normalized)		**Cosine Similarity-Personalized Pagerank User 12 (50%-50%)**		**Cosine Similarity-Personalized Pagerank User 12 (25%-75%)**		**Cosine Similarity-Personalized Pagerank User 12 (75%-25%)**	
Κατά ταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	59107	0.6960	1110435	0.5181	1110435	0.7591	59107	0.7502
2	664495	0.6958	59107	0.5004	3501204	0.5319	664495	0.7499
3	1886802	0.6485	664495	0.5002	1112117	0.2908	1886802	0.6820
4	1214302	0.5912	1886802	0.4548	1110436	0.2802	1214302	0.5996
5	2828588	0.5041	3501204	0.4257	1112112	0.2712	2828588	0.4748
6	2235131	0.4965	1214302	0.3997	59107	0.2506	2235131	0.4640
7	1197349	0.4909	2828588	0.3170	664495	0.2506	480466	0.4581
8	2235123	0.4899	480466	0.3107	2244067	0.2461	1197349	0.4561
9	480466	0.4899	2235131	0.3097	1886802	0.2277	2235123	0.4542
10	959073	0.4809	1197349	0.3047	2244088	0.2037	959073	0.4419
11	644304	0.4807	2235123	0.3028	1214302	0.1999	644304	0.4409

12	959076	0.4789	959073	0.2954	480466	0.1632	959076	0.4397
13	2828598	0.4781	959076	0.2948	2828588	0.1591	2828598	0.4376
14	959057	0.4779	644304	0.2940	2235131	0.1554	959057	0.4376
15	2780767	0.4690	959057	0.2925	1197349	0.1533	2780767	0.4242
16	2780771	0.4690	2828598	0.2922	2235123	0.1514	2780771	0.4242
17	2780770	0.4689	2780767	0.2828	959076	0.1499	2780770	0.4240
18	2780768	0.4688	2780771	0.2828	959073	0.1489	2780768	0.4239
19	1738715	0.4648	2780770	0.2827	959057	0.1475	1738715	0.4183
20	2780764	0.4638	2780768	0.2826	644304	0.1470	2780764	0.4172

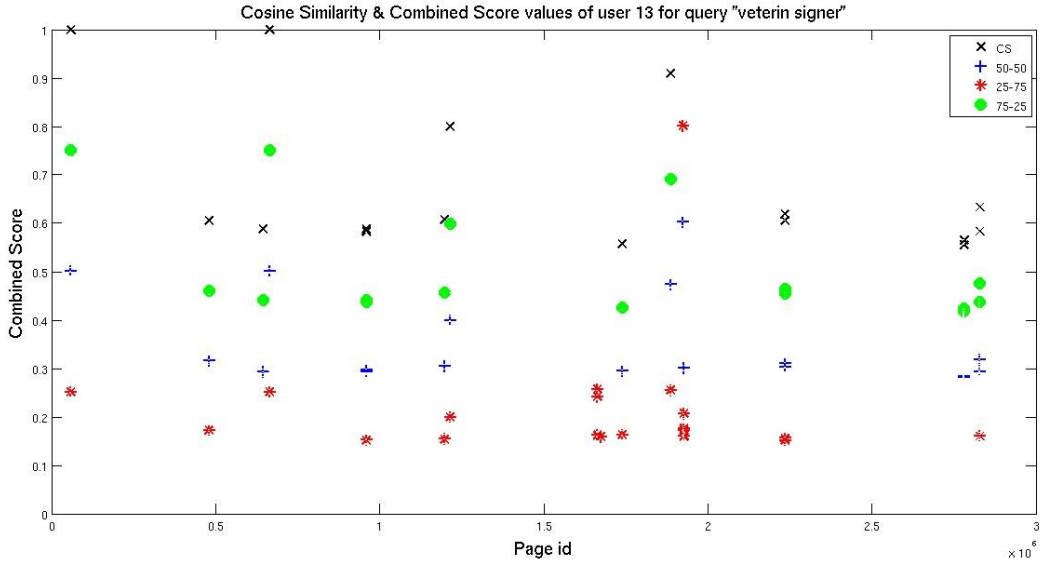
Πίνακας 5.40: Cosine Similarity-Personalized PageRank (χρήστης 12-προτιμήσεις σελίδων 1100000-1120000) για όλα τα ζεύγη βαρών για το query "veterin signer"



Διάγραμμα 5.41: Cosine Similarity-Personalized PageRank (χρήστης 12-προτιμήσεις σελίδων 1100000-1120000) για όλα τα ζεύγη βαρών για το query "veterin signer"

	Cosine Similarity (Not Normalized)		Cosine Similarity-Personalized Pagerank User 13 (50%-50%)		Cosine Similarity-Personalized Pagerank User 13 (25%-75%)		Cosine Similarity-Personalized Pagerank User 13 (75%-25%)	
Κατά ταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	59107	0.6960	1924049	0.6021	1924049	0.8011	59107	0.7503
2	664495	0.6958	59107	0.5007	1663555	0.2571	664495	0.7501
3	1886802	0.6485	664495	0.5005	1886802	0.2552	1886802	0.6911
4	1214302	0.5912	1886802	0.4732	59107	0.2510	1214302	0.5996
5	2828588	0.5041	1214302	0.3997	664495	0.2510	2828588	0.4751
6	2235131	0.4965	2828588	0.3174	1663558	0.2420	2235131	0.4642
7	1197349	0.4909	480466	0.3164	1928042	0.2070	480466	0.4610
8	2235123	0.4899	2235131	0.3101	1214302	0.1999	1197349	0.4564
9	480466	0.4899	1197349	0.3053	1928038	0.1768	2235123	0.4542
10	959073	0.4809	2235123	0.3028	1928032	0.1730	959073	0.4423
11	644304	0.4807	1928042	0.3018	480466	0.1718	644304	0.4409
12	959076	0.4789	959076	0.2965	1738715	0.1632	959076	0.4405
13	2828598	0.4781	959073	0.2963	1663556	0.1622	959057	0.4380
14	959057	0.4779	1738715	0.2947	1928036	0.1602	2828598	0.4379
15	2780767	0.4690	644304	0.2940	2828588	0.1598	1738715	0.4261
16	2780771	0.4690	959057	0.2934	1673681	0.1591	2780767	0.4242
17	2780770	0.4689	2828598	0.2926	2235131	0.1560	2780771	0.4242
18	2780768	0.4688	2780767	0.2828	1197349	0.1542	2780770	0.4240
19	1738715	0.4648	2780771	0.2828	959076	0.1525	2780768	0.4239
20	2780764	0.4638	2780770	0.2827	2235123	0.1514	2780764	0.4176

Πίνακας 5.42: Cosine Similarity-Personalized PageRank (χρήστης 13-προτιμήσεις σελίδων 1500000-2000000) για όλα τα ζεύγη βαρών για το query "veterin signer"

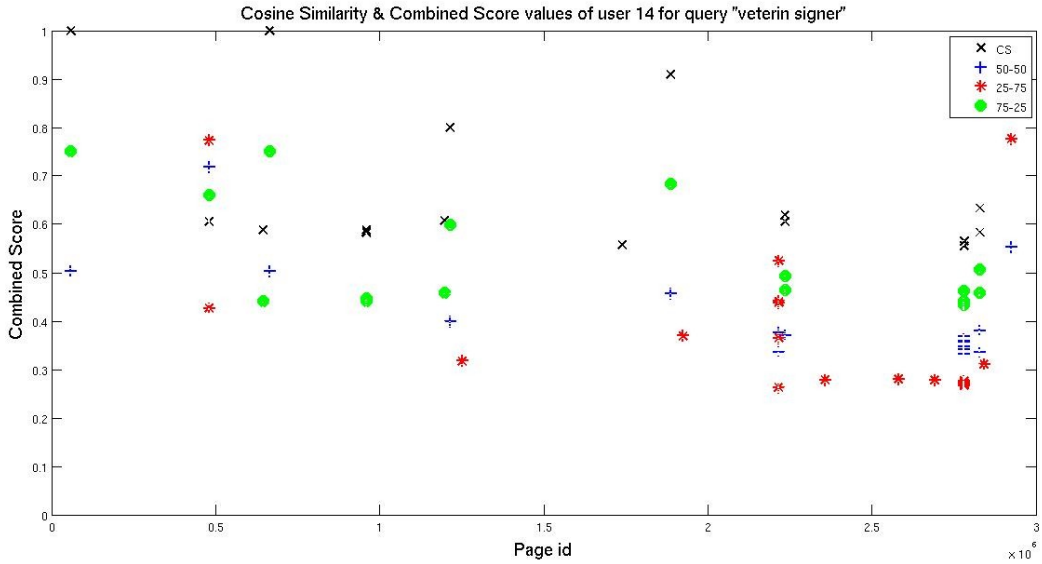


Διάγραμμα 5.43: Cosine Similarity-Personalized PageRank (χρήστης 13-προτιμήσεις σελίδων 1500000-2000000) για όλα τα ζεύγη βαρών για το query "veterin signer"

	Cosine Similarity (Not Normalized)		**Cosine Similarity-Personalized Pagerank User 14 (50%-50%)**		**Cosine Similarity-Personalized Pagerank User 14 (25%-75%)**		**Cosine Similarity-Personalized Pagerank User 14 (75%-25%)**	
Κατά ταξη	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**	**Σελίδα (id)**	**score**
1	59107	0.6960	480466	0.7175	2925126	0.7764	59107	0.7515
2	664495	0.6958	2925126	0.5529	480466	0.7734	664495	0.7514
3	1886802	0.6485	664495	0.5032	2216067	0.5249	1886802	0.6829
4	1214302	0.5912	59107	0.5030	2216069	0.4406	480466	0.6615
5	2828588	0.5041	1886802	0.4567	2216068	0.4376	1214302	0.5996
6	2235131	0.4965	1214302	0.3998	480464	0.4266	2828588	0.5067
7	1197349	0.4909	2828588	0.3807	1924049	0.3695	2235131	0.4941
8	2235123	0.4899	2216067	0.3763	2216065	0.3654	2235123	0.4644
9	480466	0.4899	2216069	0.3762	1250793	0.3176	2780764	0.4624
10	959073	0.4809	2235131	0.3699	2841305	0.3102	2828598	0.4590
11	644304	0.4807	2780764	0.3692	2581034	0.2807	1197349	0.4587

12	959076	0.4789	2780780	0.3592	2357293	0.2785	959076	0.4474
13	2828598	0.4781	2780763	0.3567	2691428	0.2775	959073	0.4457
14	959057	0.4779	2780784	0.3475	2780780	0.2769	2780780	0.4416
15	2780767	0.4690	2780779	0.3471	2780764	0.2760	959057	0.4413
16	2780771	0.4690	2780782	0.3464	2780763	0.2722	2780763	0.4413
17	2780770	0.4689	2780776	0.3412	2780781	0.2709	644304	0.4409
18	2780768	0.4688	2216068	0.3352	2780784	0.2683	2780779	0.4360
19	1738715	0.4648	2828598	0.3349	2780782	0.2669	2780767	0.4345
20	2780764	0.4638	2780781	0.3320	2214766	0.2625	2780771	0.4345

Πίνακας 5.44: Cosine Similarity-Personalized PageRank (χρήστης 14-προτιμήσεις σελίδων 2000000-3000000) για όλα τα ζεύγη βαρών για το query "veterin signer"



Διάγραμμα 5.45: Cosine Similarity-Personalized PageRank (χρήστης 14-προτιμήσεις σελίδων 2000000-3000000) για όλα τα ζεύγη βαρών για το query "veterin signer"

Από τους παραπάνω πίνακες και τα διαγράμματα του τελευταίου παραδείγματος σχετικά με τα αποτελέσματα του Cosine Similarity-Personalized PageRank score, μπορούμε αρχικά να παρατηρήσουμε ότι για τον χρήστη 10 (πίνακας 5.36-διάγραμμα 5.37), το μεγαλύτερο μέρος των αποτελεσμάτων για τα βάρη 50%-50% και 25%-75% αποτελείται από σελίδες προτίμησής του. Ο λόγος που συμβαίνει αυτό είναι ότι στα αποτελέσματα της Cosine Similarity, βρίσκονται σελίδες προτίμησης του χρήστη 10, το πλήθος και το score των οποίων αρκεί ώστε σε συνδυασμό με την

τιμή του Personalized PageRank, να αυξάνει τις πιθανότητες εμφάνισής τους στις πρώτες θέσεις της κατάταξης του ενοποιημένου score.

Προχωρώντας στα αποτελέσματα για τους χρήστες 11 και 12, παρατηρούμε ότι στις πρώτες θέσεις της κατάταξης για τα ζεύγη βαρών 50%-50% και 25%-75%, βρίσκονται προτιμώμενες σελίδες των χρηστών, παρόλο που το εύρος των προτιμήσεων τους είναι σχετικά μικρό. Φυσικά, όπως έχουμε ήδη εξηγήσει, αυτός είναι ο λόγος που αποκτούν υψηλές τιμές pagerank.

Όσο για τους χρήστες 13 και 14, οι οποίοι έχουν εύρη προτιμήσεων 500000 και 1000000 σελίδες αντίστοιχα, στα αποτελέσματα του ενοποιημένου score για τα βάρη 50%-50% παρατηρούμε ότι στις πρώτες 5 θέσεις της κατάταξης εμφανίζεται μόνο μία σελίδα προτίμησης για κάθε χρήστη (εκτός από εκείνες που μπορεί να βρίσκονται σε αυτές τις θέσεις λόγω υψηλής τιμής Cosine Similarity). Βέβαια, στη δεύτερη δεκάδα αποτελεσμάτων για τον χρήστη 14, συναντάμε πολλές σελίδες προτίμησης, λόγω του μεγάλου εύρους προτιμήσεων. Όσον αφορά τα αποτελέσματα για το ζεύγος βαρών 25%-75%, παρατηρείται η εμφάνιση πολλών προτιμώμενων σελίδων των 2 χρηστών, καθώς το μεγάλο εύρος προτιμήσεων σε συνδυασμό με το υψηλό βάρος που δίνεται στο pagerank, επιφέρουν αντίστοιχα υψηλό ενοποιημένο score για εκείνες τις σελίδες.

Όπως είναι φυσικό, τα αποτελέσματα της τελευταίας στήλης με ζεύγος βαρών 75%-25% για όλους τους χρήστες, περιλαμβάνουν κατά κύριο λόγο σελίδες που έχουν προκύψει με βάση το λεκτικό περιεχόμενο των σελίδων (μέτρο Cosine Similarity).

5.2.4 Χρόνος Εκτέλεσης Online Διεργασιών

Αφού παρουσιάσαμε αποτελέσματα σχετικά με τις τιμές που λαμβάνουν τα scores Cosine Similarity και PageRank, καθώς και ο συνδυασμός τους, θα εξετάσουμε το χρόνο εκτέλεσης των online διεργασιών του συστήματός μας. Οι διαδικασίες που εκτελούνται online, αφού δηλαδή δοθούν τα επερωτήματα προς αναζήτηση, είναι ο υπολογισμός της Cosine Similarity με βάση εκείνα τα επερωτήματα και ο συνδυασμός των scores για τον υπολογισμό του ενοποιημένου score. Μετρήσαμε τους χρόνους εκτέλεσης για 3 περιπτώσεις αναζήτησης, για διαφορετικά πλήθη διεργασιών map (Mappers) και reduce (Reducers).

Πιο συγκεκριμένα, οι περιπτώσεις για τις οποίες λάβαμε μετρήσεις είναι:

1. Αναζητήσεις με επερωτήματα που αποτελούνται από **frequent** όρους, με βάση τις προτιμήσεις του **χρήστη 5** και ζεύγος βαρών 50%-50%
2. Αναζητήσεις με επερωτήματα που αποτελούνται από **medium-frequency** όρους, με βάση τις προτιμήσεις του **χρήστη 12** και ζεύγος βαρών 25%-75%
3. Αναζητήσεις με επερωτήματα που αποτελούνται από **non-frequent** όρους, με βάση τις προτιμήσεις του **χρήστη 14** και ζεύγος βαρών 75%-25%

Τα πλήθη των Mappers και Reducers που χρησιμοποιήθηκαν είναι: 2,4,8,12 και 16.

Δεδομένου ότι ο Hadoop cluster του εργαστηρίου αποτελείται από 17 κόμβους, για κάθε ένα από τα παραπάνω πλήθη, θα ισχύει ότι κατά τη διάρκεια κάθε job, χρησιμοποιούνται τόσοι υπολογιστικοί κόμβοι όσοι και Mappers/Reducers.

Οι χρόνοι εκτέλεσης με βάση τις παραπάνω προδιαγραφές, φαίνονται στους πίνακες και στα αντίστοιχα διαγράμματα:

- **Υπολογισμός Cosine Similarity**

Συνθήκες Αναζήτησης	**Πλήθος Mappers/Reducers**				
	2	**4**	**8**	**12**	**16**
Frequent όροι, Χρήστης 5, 50%-50%	12m30s	9m40s	6m25s	5m18s	5m12s
Medium-Frequency όροι, Χρήστης 12, 25%-75%	15m24s	7m58s	6m38s	5m29s	4m55s
Non-Frequent όροι, Χρήστης 14, 75%-25%	15m21s	10m11s	5m56s	5m23s	4m55s

Πίνακας 5.46: Χρόνοι Εκτέλεσης υπολογισμού της Cosine Similarity – Πλήθος Mappers/Reducers: 2,4,8,12,16

- **Υπολογισμός Ενοποιημένου Score**

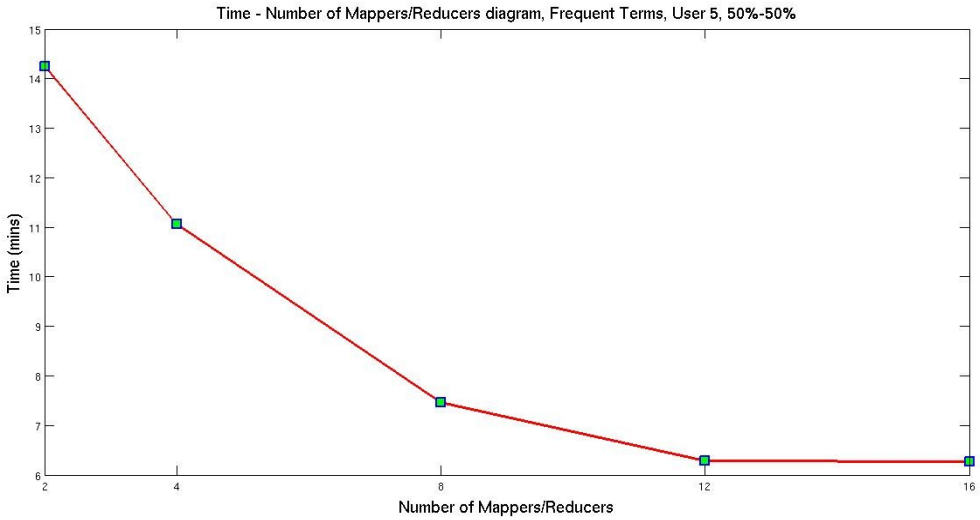
Συνθήκες Αναζήτησης	**Πλήθος Mappers/Reducers**				
	2	**4**	**8**	**12**	**16**
Frequent όροι, Χρήστης 5, 50%-50%	1m45s	1m24s	1m3s	59s	1m3s
Medium-Frequency όροι, Χρήστης 12, 25%-75%	1m37s	1m13s	1m4s	58s	59s
Non-Frequent όροι, Χρήστης 14, 75%-25%	1m54s	1m28s	1m8s	59s	1m

Πίνακας 5.47: Χρόνοι Εκτέλεσης υπολογισμού του ενοποιημένου score – Πλήθος Mappers/Reducers: 2,4,8,12,16

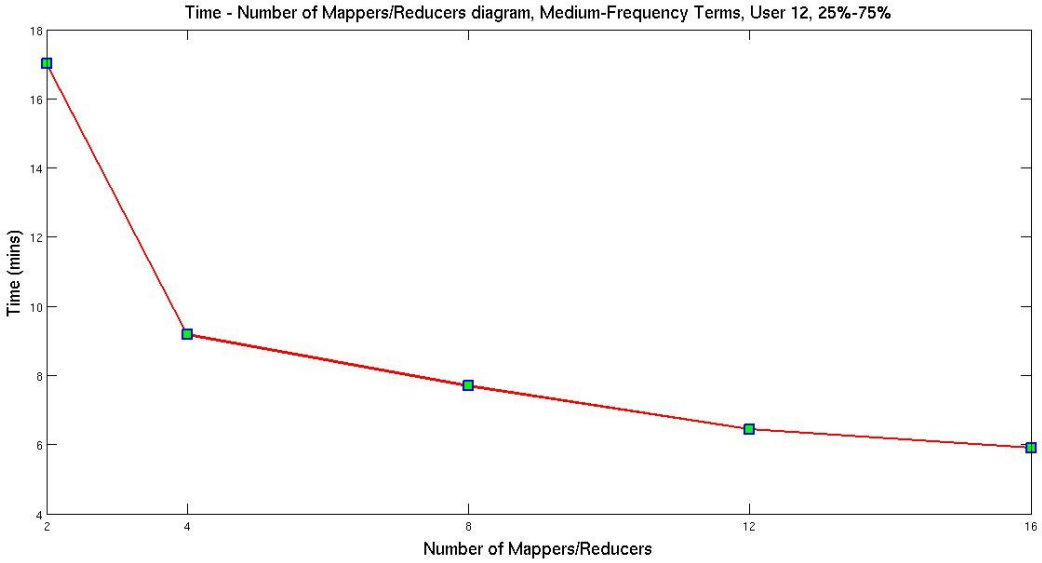
- **Συνολικός Χρόνος Εκτέλεσης online διεργασιών**

Συνθήκες Αναζήτησης	**Πλήθος Mappers/Reducers**				
	2	**4**	**8**	**12**	**16**
Frequent όροι, Χρήστης 5, 50%-50%	14m15s	11m4s	7m28s	6m17s	6m15s
Medium-Frequency όροι, Χρήστης 12, 25%-75%	17m1s	9m11s	7m42s	6m27s	5m54s
Non-Frequent όροι, Χρήστης 14, 75%-25%	17m15s	11m39s	7m4s	6m22s	5m55s

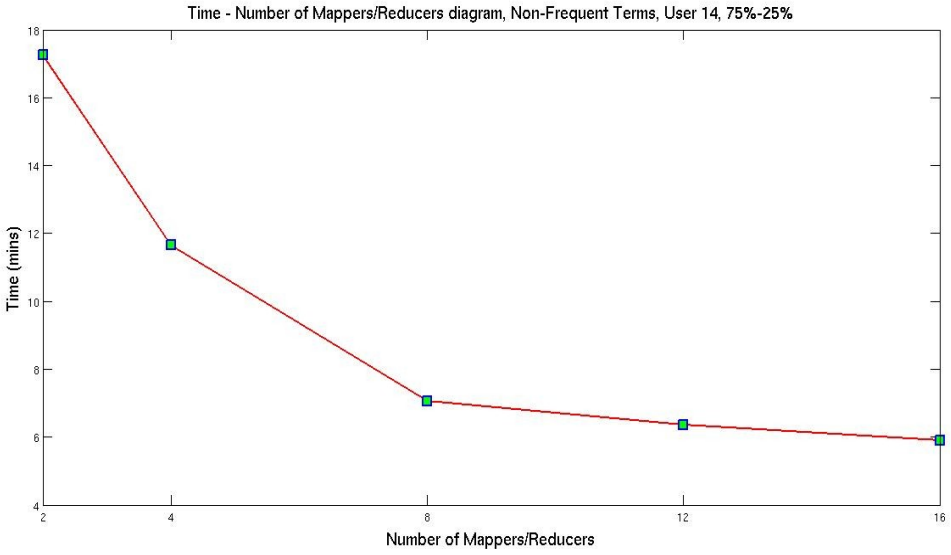
Πίνακας 5.48: Χρόνοι Εκτέλεσης υπολογισμού των online διεργασιών (Υπολογισμοί Cosine Similarity και ενοποιημένου score) – Πλήθος Mappers/Reducers: 2,4,8,12,16



Διάγραμμα 5.49: Χρόνος Εκτέλεσης – Πλήθος Mappers/Reducers (επερωτήματα με frequent όρους, Χρήστης 5, βάρη: 50%-50%)



Διάγραμμα 5.50: Χρόνος Εκτέλεσης – Πλήθος Mappers/Reducers (επερωτήματα με medium-frequency όρους, Χρήστης 12, βάρη: 25%-75%)



Διάγραμμα 5.51: Χρόνος Εκτέλεσης – Πλήθος Mappers/Reducers (επερωτήματα με non-frequent όρους, Χρήστης 14, βάρη: 75%-25%)

Από τα παραπάνω διαγράμματα, παρατηρούμε ότι ο χρόνος εκτέλεσης παρουσιάζει ιδιαίτερη μείωση ειδικά όταν μεταβαίνουμε από 2 Mappers/Reducers σε 4 και από 4 σε 8. Η χρονική διαφορά στις εκτελέσεις για 8, 12 και 16, είναι αρκετά πιο μικρή, όσο πλησιάζουμε την τιμή 17, δηλαδή το πλήθος των υπολογιστικών κόμβων του cluster.

Όπως αναφέραμε και προηγουμένως, για τα παραπάνω πλήθη Mappers/Reducers, ισχύει ότι αντιστοιχούν στο πλήθος των κόμβων που λαμβάνουν μέρος στις διεργασίες, καθώς είναι τιμές μικρότερες από 17 και το Hadoop αναλαμβάνει να καταμερίσει τις διεργασίες αποδοτικά σε κάθε κόμβο του cluster. Έτσι, κάθε κόμβος εκτελεί έναν μοναδικό Mapper και Reducer. Ο καταμερισμός των εργασιών είναι το μεγάλο πλεονέκτημα του Hadoop και της παράλληλης επεξεργασίας μεγάλου όγκου δεδομένων. Όσο λοιπόν αυξάνονται οι κόμβοι εργασίας, κάθε κόμβος αναλαμβάνει να διαχειριστεί μικρότερο όγκο πληροφορίας, άρα έχει τη δυνατότητα να εκτελέσει με μεγαλύτερη ταχύτητα τις απαιτούμενες διεργασίες. Είναι φυσιολογικό λοιπόν να μειώνεται ο χρόνος εκτέλεσης των διαφόρων διαδικασιών, όταν οι σταθμοί εργασίας αυξάνονται.. Είναι προφανές πως εάν είχαμε στη διάθεσή μας περισσότερους υπολογιστικούς κόμβους, οι τιμές των χρόνων εκτέλεσης θα ήταν ακόμη πιο χαμηλές.

Με το παραπάνω παράδειγμα ολοκληρώθηκε η επίδειξη και αξιολόγηση ενός υποσυνόλου των πειραματικών αποτελεσμάτων της παρούσας εργασίας. Παρουσιάστηκαν αποτελέσματα από όλες τις κατηγορίες επερωτημάτων με βάση τη συχνότητα εμφάνισης των όρων που τα αποτελούν (frequent, medium-frequency, non-frequent, random-frequency terms) και με βάση το πλήθος των όρων (1-5 όροι/επερώτημα). Τα αποτελέσματα αποτελούνται από scores που προέρχονται από τα μέτρα Cosine Similarity και PageRank (Personalized ή Original), καθώς και από το συνδυασμό τους, ενώ χρησιμοποιήθηκαν 3 ζεύγη βαρών (50%-50%, 25%-75% και 75%-25%) για το συνδυασμό Cosine Similarity-PageRank. Επίσης, παρουσιάστηκαν διαγράμματα του χρόνου εκτέλεσης των online διεργασιών αναζήτησης σε σχέση με τον αριθμό των Mappers και Reducers

που χρησιμοποιούνται.

Στο επόμενο κεφάλαιο θα γίνει η παρουσίαση των συμπερασμάτων που εξήχθησαν συνολικά από την εργασία καθώς και μελλοντικές επεκτάσεις που θα μπορούσαν να υλοποιηθούν ώστε να επιτευχθεί μεγαλύτερη ταχύτητα στους υπολογισμούς, εξοικονόμηση χώρου κ.α.

6. Συμπεράσματα και Future Work

Οι σύγχρονες μηχανές αναζήτησης διαδικτύου (web search engines) καλούνται να διαχειριστούν τεράστιο όγκο δεδομένων, ενώ παράλληλα το μέγεθος της πληροφορίας που βρίσκεται στο διαδίκτυο συνεχώς αυξάνεται. Απαιτείται λοιπόν αντίστοιχα μεγάλος αποθηκευτικός χώρος ώστε να καλύψει τέτοιου είδους ανάγκες. Γι' αυτό και οι μεγάλες εταιρείες (π.χ. Google, Yahoo!) διαθέτουν αποθήκες δεδομένων (data warehouses) σε διαφορετικές περιοχές με σκοπό την ανταπόκρισή τους στις απαιτήσεις των χρηστών σε όλο τον κόσμο. Εκτός όμως από την κάλυψη του συνεχώς αυξανόμενου μεγέθους της πληροφορίας, χρειάζεται και η απόδοση μίας μηχανής να είναι η βέλτιστη δυνατή. Αυτό σημαίνει ότι η διαχείριση των δεδομένων θα πρέπει να βασίζεται σε συστήματα τα οποία είναι ανεκτικά σε αστοχίες και ταυτόχρονα να είναι όσο το δυνατόν πιο αποδοτικά, τόσο από άποψη ταχύτητας όσο και από πλευράς εξαγωγής ουσιαστικών αποτελεσμάτων. Η ευστοχία των αποτελεσμάτων των εμπορικών μηχανών αναζήτησης προκύπτει από τον συνδυασμό παραμέτρων, ένας από τους οποίους είναι το PageRank της Google, ενώ τα τελευταία χρόνια τα αποτελέσματα επηρεάζονται και από οικονομικούς παράγοντες μέσω των διαφημίσεων (π.χ. AdWords).

Στην παρούσα εργασία έγινε η προσπάθεια προσομοίωσης ενός πραγματικού συστήματος μηχανής αναζήτησης, χρησιμοποιώντας μεγάλο αριθμό ιστοσελίδων ως πειραματικά δεδομένα. Για την επεξεργασία του συνόλου της πληροφορίας δε θα αρκούσε φυσικά η σειριακή προσπέλαση των δεδομένων, ούτε η αποθήκευσή τους σε ένα μόνο μηχάνημα. Επομένως, για να διαχειριστούμε τον όγκο της πληροφορίας, χρησιμοποιήσαμε το σύστημα Hadoop, το οποίο προσφέρει το μοτίβο προγραμματισμού Map-Reduce, που είναι σχεδιασμένο για την παράλληλη επεξεργασία μεγάλου μεγέθους συνόλων δεδομένων. Όλη η πληροφορία αποθηκεύεται στο κατανεμημένο σύστημα αρχείων του Hadoop (HDFS). Κάποια από τα πιο σημαντικά πλεονεκτήματα του HDFS είναι η ταχύτητα στην πρόσβαση των δεδομένων και η αυτόματη επαναφορά του συστήματος ύστερα από αστοχίες του υλικού (hardware failures). Το HDFS προσφέρει λοιπόν έναν αξιόπιστο αποθηκευτικό χώρο μέσω των μηχανημάτων που αποτελούν τον εκάστοτε Hadoop cluster.

Κατά την προσπάθειά μας για την κατασκευή ενός αξιόπιστου συστήματος που εξάγει ουσιώδη αποτελέσματα, εκμεταλλευτήκαμε τόσο το λεκτικό περιεχόμενο (text content) των σελίδων όσο και τον γράφο (webgraph) που απεικονίζει τους συνδέσμους μεταξύ τους. Έτσι, οδηγηθήκαμε στον υπολογισμό των scores Cosine Similarity και PageRank (Original και Personalized) αντίστοιχα. Η εκτέλεση των απαραίτητων διεργασιών που εκτελέστηκαν από την αρχική επεξεργασία των δεδομένων μέχρι τους τελευταίους υπολογισμούς δε θα ήταν εφικτή δίχως τη

χρήση του Hadoop, για τους λόγους που εξηγήσαμε παραπάνω.

Όσον αφορά το λεκτικό περιεχόμενο των σελίδων, χρησιμοποιήσαμε τα tf-idf scores, με βάση τα οποία υπολογίστηκε στη συνέχεια το μέτρο Cosine Similarity, δεδομένων κάποιων επερωτημάτων διαφόρων κατηγοριών (με βάση τη συχνότητα εμφάνισης των όρων και το πλήθος των όρων). Το συγκεκριμένο score μετράει την ομοιότητα ενός επερωτημάτος με ένα κείμενο, επομένως ο υπολογισμός του είχε ως αποτέλεσμα τις πιο σχετικές σελίδες ανάλογα με το εκάστοτε επερώτημα. Παρατηρήσαμε ότι, σε γενικές γραμμές, οι τιμές της Cosine Similarity είναι πιο υψηλές όταν χρησιμοποιούνται επερωτήματα αποτελούμενα από όρους που εμφανίζονται συχνά στη συλλογή των κειμένων.

Περνώντας στην ανάλυση του webgraph, εκτελέστηκε ο επαναληπτικός αλγόριθμος PageRank σε περιβάλλον Hadoop, με αποτέλεσμα την ανάθεση μίας τιμής pagerank για κάθε σελίδα της συλλογής. Η τιμή εκείνη αντιπροσωπεύει ένα μέτρο κύρους για κάθε σελίδα. Η λογική του αλγορίθμου είναι ότι οι σελίδες οι οποίες προέρχονται από συνδέσμους που βρίσκονται σε άλλες σελίδες με υψηλό pagerank, επωφελούνται και αποκτούν και εκείνες μερίδιο από την τιμή pagerank της σελίδας από την οποία προήλθαν. Οπότε, οι σελίδες οι οποίες έλαβαν τις υψηλότερες τιμές, είναι τελικά εκείνες που το σύστημά μας θεωρεί ως περισσότερο σημαντικές.

Προχωρώντας ένα βήμα παρακάτω, αναθέσαμε προτιμήσεις σελίδων σε φανταστικούς χρήστες του συστήματος. Αυτό άλλωστε συμβαίνει και με τους πραγματικούς χρήστες, καθώς υπάρχει η δυνατότητα να εμφανίζονται αποτελέσματα σχετικά με τα ενδιαφέροντα τους, στις διάφορες αναζητήσεις που εκτελούν καθημερινά. Με βάση λοιπόν τις προτιμήσεις για τους χρήστες του συστήματός μας, υλοποιήθηκε το Εξατομικευμένο (Personalized) PageRank, μία παραλλαγή του πρωτότυπου (Original) αλγορίθμου. Εξετάζοντας τα αποτελέσματα, οι τιμές pagerank των προτιμώμενων σελίδων κάθε χρήστη ήταν σαφώς υψηλότερες, τόσο σε σχέση με τις υπόλοιπες όσο και με τις τιμές που απέκτησαν από το Original PageRank. Το εύρος προτιμήσεων κάθε χρήστη επηρέασε με το δικό του τρόπο τα αποτελέσματα. Οι χρήστες με μικρό εύρος προτιμήσεων, έχουν το πλεονέκτημα ότι οι σελίδες που τους ενδιαφέρουν άμεσα λαμβάνουν υψηλότερο pagerank απ' ότι οι σελίδες των χρηστών με μεγαλύτερα εύρη προτιμήσεων. Από την άλλη πλευρά, όταν το πλήθος των σελίδων προτίμησης είναι σχετικά μεγάλο για έναν χρήστη, τότε η πιθανότητα εμφάνισης εκείνων των σελίδων στα τελικά αποτελέσματα (ανεξάρτητα από τη θέση την οποία καταλαμβάνουν) είναι αυξημένη.

Στη συνέχεια της υλοποίησης, επιχειρήθηκε ο συνδυασμός των μεθόδων-scores Cosine Similarity και PageRank (Original και Personalized). Έγιναν πειράματα κατά τα οποία χρησιμοποιήθηκαν τρία ζεύγη βαρών (25%-75%, 50%-50% και 75%-25%), ώστε να έχουμε αποτελέσματα τα οποία επηρεάζονται κατά διαφορετικά ή ίσα ποσοστά από τις 2 μεθόδους αξιολόγησης. Παρατηρήσαμε, λοιπόν, ότι ανάλογα με το βάρος κάθε score, υπήρχε αντίστοιχης κλίμακας επίδραση στο ενοποιημένο score. Για παράδειγμα, με το ζεύγος βαρών 25%-75% (25% Cosine Similarity-50% PageRank), τα αποτελέσματα ήταν σαφώς επηρεασμένα από το PageRank, οπότε και στις πρώτες θέσεις βρίσκονταν πολλές σελίδες προτίμησης του εκάστοτε χρήστη. Για το ζεύγος 50%-50%, υπήρχε επίσης αντίστοιχο φαινόμενο, παρόλο που οι σελίδες προτίμησης του χρήστη στις πρώτες θέσεις των αποτελεσμάτων να ήταν λιγότερες σε σχέση με τα βάρη 25%-75%. Φυσικά, στο πλήθος των σελίδων προτίμησης στα τελικά αποτελέσματα, παίζει ρόλο και το μέγεθος του εύρος προτιμήσεων κάθε χρήστη, όπως είπαμε προηγουμένως. Όσο για το ζεύγος βαρών 75%-25%, φαίνεται ξεκάθαρα από τα πειραματικά αποτελέσματα ότι η κατάταξη με βάση το PageRank επηρεάζει ελάχιστα την κατάταξη στις περισσότερες περιπτώσεις. Όσα αναφέραμε σχετικά με τα

αποτελέσματα Cosine Similarity-Personalized PageRank, ισχύουν και για τα αποτελέσματα Cosine Similarity-Original PageRank, με τη διαφορά ότι δεν υπάρχουν σελίδες προτίμησης που να αποκτούν υψηλότερες τιμές pagerank λόγω των ενδιαφερόντων του χρήστη. Είναι όμως εμφανές ότι οι σελίδες με υψηλότερο PageRank "σκαρφαλώνουν" στις πρώτες θέσεις των αποτελεσμάτων του ενοποιημένου score.

Όσον αφορά τους χρόνους εκτέλεσης των online διεργασιών (υπολογισμός Cosine Similarity και ενοποιημένου score) του συστήματος μηχανής αναζήτησης που υλοποιήσαμε, παρατηρήσαμε ότι όσο αυξάνονται οι Mappers και οι Reducers (και κατ' επέκταση οι υπολογιστικοί κόμβοι στις μετρήσεις μας), τόσο αυξάνεται η ταχύτητα των υπολογισμών. Άρα, η απόδοση του συστήματος αυξάνεται με την ταυτόχρονη αύξηση των κόμβων εργασίας.

Η παρούσα εργασία επιδέχεται μελλοντικές επεκτάσεις, αρκεί να αναλογιστεί κάποιος ότι χρησιμοποιούνται όλο και περισσότερες και πιο σύνθετες παράμετροι για την επίτευξη των βέλτιστων δυνατών αποτελεσμάτων στην αναζήτηση. Όσον αφορά την αναζήτηση με βάση το λεκτικό περιεχόμενο μίας σελίδας, υπάρχει η δυνατότητα να χρησιμοποιηθούν βάρη για συγκεκριμένες λέξεις του κειμένου (keywords), όπως είναι, για παράδειγμα, ο τίτλος (title) της σελίδας, η επικεφαλίδα (header) και το κείμενο των συνδέσμων (anchor text). Εκείνες οι λέξεις θα λαμβάνουν με αυτόν τον τρόπο υψηλότερα tf-idf scores, κάτι που θα τις κάνει αυτόματα σημαντικές κατά την αναζήτηση.

Όσον αφορά το Personalized PageRank, υπάρχει η δυνατότητα ώστε η υλοποίηση και κατά συνέπεια ο υπολογισμός των τιμών για όλες τις σελίδες και για κάθε χρήστη να γίνουν πιο αποδοτικά με χρήση διαφορετικών μεθόδων. Στην παρούσα εργασία, για κάθε χρήστη υπολογίστηκαν οι τιμές pagerank όλων των σελίδων , κάτι που σαφώς δεν είναι ό,τι πιο οικονομικό από άποψη χώρου και χρόνου, όσο οι χρήστες του συστήματος πληθαίνουν. Μία πιο αποδοτική υλοποίηση παρουσιάζεται στο [6], όπου προτείνεται η χρήση των Personalized PageRank Vectors, το καθένα από τα οποία θα αντιστοιχεί σε μία κατηγορία ή θέμα, όπως είναι τα αθλητικά, η πολιτική κ.α. Για παράδειγμα, εάν υπάρχουν 10 κατηγορίες, θα υπολογιστούν αντίστοιχα 10 vectors που θα περιλαμβάνουν τις τιμές pagerank για κάθε σελίδα με βάση τη συγκεκριμένη κατηγορία κάθε φορά. Στη συνέχεια, ανάλογα με τις προτιμήσεις κάθε χρήστη οι οποίες μπορεί να είναι σύνθετες (π.χ. 60% αθλητικά και 40% πολιτική), θα υπολογίζεται ο συνδυασμός των vectors με σκοπό την εξαγωγή των τιμών pagerank για το συγκεκριμένο χρήστη. Έτσι, δε θα χρειάζεται η εκτέλεση του αλγορίθμου για κάθε χρήστη, παρά μόνο για κάθε κατηγορία ενδιαφερόντων. Η παραπάνω ιδέα συνιστά την έννοια του Topic-Sensitive PageRank, κάτι που απαιτεί και την χρήση ενός classifier για την ταξινόμηση των σελίδων με βάση κατηγορίες που είναι διαθέσιμες.

Βιβλιογραφία

[1] J. Dean and S. Ghemawat. *Mapreduce: Simplified data processing on large clusters,* In Proc. of OSDI 2004.

[2] R Khare, D Cutting, K Sitaker and A. Rifkin, *Nutch: A Flexible and Scalable Open-Source Web Search Engine,* Oregon State University, 2004.

[3] J. Lin and C. Dyer. *Data-Intensive Text Processing with MapReduce*. Morgan & Claypool Publishers, 2010.

[4] S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data.* Morgan-Kaufmann Publishers, 2002.

[5] T. White. *Hadoop: The Definitive Guide.* O'Reilly Media, May 2009.

[6] C. D. Manning, P. Raghavan and H. Schütze. *An Introduction to Information Retrieval*. Cambridge University Press, 2008.

[7] R. M. C. McCreadie, C. Mcdonald, I. Ounis. *Comparing Distributed Indexing: To MapReduce or Not?* In 7th Workshop on Large-Scale Distributed Systems for Information Retrieval, 2009.

[8] T. Elsayed, F. Ture, J. Lin. *Brute-Force Approaches to Batch Retrieval: Scalable Indexing with MapReduce, or Why Bother?* Technical Report HCIL-2010-23, University of Maryland, College Park, October 2010.

[9] G. Jeh, J. Widom. *Scaling Personalized Web Search.* Proceedings of the 12th international conference on World Wide Web, 2003.

[10] K. Shvachko, H. Kuang, S. Radia, R. Chansler. *The Hadoop Distributed File System.* Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010

[11] S. Brin, L. Page. *The Anatomy of a Large-Scale Hypertextual Web Search Engine.* In: Seventh International World-Wide Web Conference (WWW 1998), April 1998

[12] S. Brin, L. Page, R. Motwani, T. Winograd. *The PageRank Citation Ranking: Bringing Order to the Web.* Stanford InfoLab, 1999

[13] T. H. Haveliwala. *Topic-Sensitive PageRank: A Context-Sensitive Ranking Algorithm for Web Search.* IEEE Transactions on Knowledge and Data Engineering, 2003

[14] K. Csalogány, D. Fogaras, B. Rácz, T. Sarlós. *Towards Scaling Fully Personalized PageRank: Algorithms, Lower Bounds, and Experiments.* In Proceedings of the 3rd Workshop on Algorithms and Models for the Web-Graph, 2004

[15] J. Lin, M. Schatz. *Design Patterns for Efficient Graph Algorithms in MapReduce.* Proceedings of the Eighth Workshop on Mining and Learning with Graphs, 2010

[16] U Kang, C. E. Tsourakakis, C. Faloutsos. *PEGASUS: A Peta-Scale Graph Mining System-Implementation and Observations.* IEEE International Conference on Data Mining, 2009

[17] B. Bahmani, K. Chakrabarti, D. Xin. *Fast personalized PageRank on MapReduce.*

Proceedings of the 2011 ACM SIGMOD International Conference on Management of data Pages 973-984, 2011

[18] Y. Wang, X. Qi, B. D. Davison. *Standing on the Shoulders of Giants: Ranking by Combining Multiple Sources.* Lehigh University Technical Report, 2007.

[19] J. Ko, L. Hiyakumoto, E. Nyberg. *Exploiting multiple semantic resources for answer selection.* Proceedings of LREC, 2006

[20] http://hadoop.apache.org , Apache Hadoop Project

[21] http://nutch.apache.org , Apache Nutch Project

[22] http://lucene.apache.org , Apache Lucene Project

[23] http://textcat.sourceforge.net , Java Text Categorizing Library (JTCL)

[24] http://lintool.github.com/Cloud9 , Cloud9 Library

[25] http://terrier.org , Terrier IR Platform

[26] http://katta.sourceforge.net , Katta Project

[27] http://lucene.apache.org/solr , Apache Solr

[28] http://openbixo.org/ , Bixo

[29] http://archive-access.sourceforge.net/projects/nutch/ , NutchWax

[30] http://www.dmoz.org/ , DMOZ Open Directory Project

[31] T. Haveliwala, S. Kamvar, G. Jeh. *An Analytical Comparison of Approaches to Personalizing PageRank.* Stanford University Technical Report, 2003