Μεγάλης Αποδοτικότητας Crawler Βασισμένος στο Hadoop

A High Performance Hadoop Based Parallel Crawler

11/3/2010

Πολυτεχνείο Κρήτης –Τμήμα ΗΜΜΥ

Καλλιόπη Καλαντζάκη

Περιεχόμενα

ΠΕΡΙΛΗΨΗ

Διανύουμε την εποχή της πληροφορίας στην οποία έχουμε πρόσβαση μέσω του διαδικτύου(Internet). Η διάθεση και διακίνηση μεγάλου όγκου δεδομένων μέσω αυτού γέννησε πολλά προβλήματα ως προς την συλλογή, επεξεργασία και ανάλυση της παρεχόμενης γνώσης στο παγκόσμιο ιστό. Για αυτούς τους λόγους δημιουργείται η ανάγκη κατασκευής συστημάτων που συλλέγουν και διαχειρίζονται την προσφερόμενη πληροφορία (**Crawlers**) . Αυτοί είναι και τα δομικά στοιχεία ενός συστήματος αναζήτησης (**Web Search Engine**) αφού μέσω εκείνων συγκεντρώνεται το σύνολο της πληροφορίας του παγκόσμιου ιστού.

Σημαντική πρόκληση λοιπόν καθίσταται η δημιουργία αποδοτικών αρχιτεκτονικών που εξασφαλίζουν ταχύτητα, επεκτασιμότητα και ανοχή σε σφάλματα κατά την διαδικασία ανάλυσης και συγκέντρωσης της παγκόσμιας πληροφορίας. Απάντηση σε αυτές τις απαιτήσεις δίνει το **Hadoop,** ένα σύστημα που βασίζεται στο μοτίβο **Map/Reduce** και προσφέρει παραλληλισμό, καταμερισμό εργασιών και συνέπεια σε αστοχίες. Μέσω αυτού τόσο η συγκέντρωση των διαθέσιμων δεδομένων όσο και η ανάλυση αυτών γίνεται σήμερα πολύ εύκολα και αποδοτικά καταμερίζοντας το προσφερόμενο φόρτο εργασίας σε πολλούς διαφορετικούς σταθμούς εργασίας που υποστηρίζουν το **Hadoop**.

Στην παρούσα εργασία, θα παρουσιαστεί η προσπάθεια κατασκευής ενός Crawler και εκμεταλλευόμενοι τα πλεονεκτήματα του **Hadoop** θα δούμε πώς μπορούμε να κατασκευάσουμε ένα αποδοτικό σύστημα. Επίσης, θα παρουσιαστούν και οι προσπάθειες ανάλυσης και επεξεργασίας της συλλεγόμενης πληροφορίας βασισμένοι στο **Hadoop** ώστε να μοντελοποιηθεί ένας μεγάλος όγκος δεδομένων. Τελευταίο στάδιο αυτής της διαδικασίας αποτελεί η υποβολή ερωτημάτων από τον χρήστη για την παρουσίαση πληροφοριών σχετικές με τις απαιτήσεις του χρήστη. Και πάλι για αυτόν τον σκοπό εκμεταλλευόμαστε τις λειτουργικότητες του **Hadoop.**

ΕΙΣΑΓΩΓΙΚΟ ΜΕΡΟΣ

1. ΓΕΝΙΚΑ

Ζούμε στην εποχή της μαζικής πληροφορίας και μάλιστα έχοντας το πλεονέκτημα της ελεύθερης πρόσβασης σε οποιαδήποτε πηγή μέσω του διαδικτυακού κόσμου. Και όχι μόνο. Στις μέρες μας το διαδίκτυο καθίσταται το πρωταρχικό μέσον επικοινωνίας ,ψυχαγωγίας, ένας διαπολιτισμικός κόσμος που μας προσφέρει ασύλληπτες δυνατότητες μονάχα μέσω μιας οθόνης. Πολύ εύκολο για εμάς σήμερα, το ίδιο και ακόμα πιο δύσκολο όμως για ολόκληρη την επιστημονική κοινότητα που βρίσκεται πίσω από όλο αυτόν τον μαγικό κόσμο που εμείς εξερευνούμε μέσω μίας μηχανής αναζήτησης (Web Search Engines).

Αυτός είναι και ο συνδετικός κρίκος ανάμεσα σε ένα απλό χρήστη και στο σύνολο των πληροφοριών του διαδικτύου αφού μέσω αυτού καθίσταται εφικτό να ανακτήσουμε δεδομένα που μας ενδιαφέρουν. Εκεί έγκειται και η δυσκολία, να συγκεντρώσουμε το σύνολο των πληροφοριών παγκοσμίως ώστε να καταφέρουμε από αυτόν τον όγκο δεδομένων να αναλύσουμε και να επεξεργαστούμε ότι χρειάζεται για να χτίσουμε ένα αξιόπιστο και γρήγορο μηχανισμό αναζήτησης.

Τα πρώτα βήματα προς αυτήν την κατεύθυνση έγιναν πριν το 1995 όταν ο αριθμός των διαθέσιμων ιστοσελίδων ανάγονταν σε κάποιες χιλιάδες ακόμα. Σύντομα η αναλογία αυτή αυξήθηκε αγγίζοντας μέσα σε μια δεκαετία τα δύο δισεκατομμύρια παγκοσμίως. Το γεγονός αυτό πυροδότησε το ενδιαφέρον πολλών ερευνητών για την δημιουργία μοντέλων και αρχιτεκτονικών που με κάποιο τρόπο θα συγκέντρωναν όλο αυτόν τον όγκο πληροφορίας για περαιτέρω επεξεργασία και ανάλυση.

Τα πρώτα πορίσματα από τέτοιες προσπάθειες ήρθαν λίγο αργότερα με την κατασκευή συστημάτων που ουσιαστικά εξερευνούσαν τον παγκόσμιο ιστό συγκεντρώνοντας τα διαθέσιμα αυτά δεδομένα. Οι γνωστοί σε όλους μας *Web Crawlers,* έγιναν τα εφαλτήρια για την ανάπτυξη και υλοποίηση μηχανών αναζήτησης μαζικού όγκου δεδομένων, στρατηγικών για τον εντοπισμό νέων ιστοσελίδων, αποθήκευσης των εξαγώμενων δεδομένων, δεικτοδότησης κλπ.

Από τα παραπάνω γίνεται κατανοητό πώς το μέγεθος των δεδομένων έγινε ένα πρόβλημα οπότε για να ανταπεξέλθουμε σε τέτοιες απαιτήσεις έπρεπε να κατασκευαστούν συστήματα υψηλής αποδοτικότητας ώστε να προσεγγίσουμε την πλειοψηφία των διαθέσιμων ιστοσελίδων και σε εύλογο χρονικό διάστημα. Και όχι μόνο. Συστήματα τέτοιων προδιαγραφών έπρεπε όχι μόνο να έχουν μεγάλη

κάλυψη στο σύνολο των ιστοσελίδων αλλά ταυτόχρονα να υποστηρίζουν ένα αντίστοιχο αποθηκευτικό χώρο, συνέπεια των υποκείμενων βάσεων αλλά κυρίως ταχύτητα στα υποβαλλόμενα επερωτήματα των χρηστών .

Συνεπώς, στην προσπάθεια να υλοποιηθεί ένα σύστημα με τα παραπάνω χαρακτηριστικά πρέπει να συμπεριλάβουμε κυρίως τρεις παράγοντες: Πρώτον, την υλοποίηση ενός Crawler που να κατεβάζει ένα μεγάλο αριθμό ιστοσελίδων ανά δευτερόλεπτο και δεύτερον ανοχή σε πιθανά σφάλματα και σε αστοχίες του συστήματος και τρίτον ταχύτητα και αποδοτικότητα κατά την επεξεργασία της συγκεντρωμένης πληροφορίας. Προφανώς , για να υποστηριχθούν τέτοια χαρακτηριστικά έπρεπε να στραφούμε στην λογική του παραλληλισμού των διαδικασιών τόσο κατά την φάση της συλλογής των δεδομένων όσο και κατά την επεξεργασία αυτών για την κατασκευή μιας μηχανής αναζήτησης.

Η πρώτη κίνηση προς αυτήν την κατεύθυνση έγινε το 2002 όταν ξεκίνησε να κατασκευάζεται το Nutch-στην αρχική του μορφή- αλλά η πιο αξιόλογη προσπάθεια ήρθε λίγο αργότερα από την γνωστής σε όλους μηχανή αναζήτησης ***Google***. Το στοιχείο που την έκανε να ξεχωρίσει ήταν η ιδέα να διαχειριστεί αυτόν τον όγκο της πληροφορίας βάσει του μοτίβου **Map/Reduce** αλλά και να ταξινομήσει τα εξαγόμενα αποτελέσματα με γνώμονα την σημαντικότητα του site ,το rank. Κατ' αυτόν τον τρόπο, μέσω μίας απλής ιδέας επιτεύχθηκε παράλληλη επεξεργασία με δυνατότητες επέκτασης εδραιωθεί (ενδεικτικά 1000 κόμβοι) σε βαθμό που η διαχείριση δεδομένων μεγάλου όγκου πληροφορίας να γίνεται εύκολα και πολύ γρήγορα. Αυτά τα δυο συντέλεσαν στο να μία αξιόπιστη και αρκετά γρήγορη μηχανή εξυπηρετώντας καθημερινά εκατομμύρια αιτήσεις για αναζήτηση πληροφοριών.

Τι ακριβώς υποστήριζε αυτό το μοτίβο; Μέσα σε γενικά πλαίσια την παράλληλη ομαδοποίηση δεδομένων τα οποία και επεξεργάζονταν κατά τις δύο αυτές φάσεις. Αυτό το στοιχείο ήταν και ο ακρογωνιαίος λίθος πρώτον για την υλοποίηση μίας τόσης μεγάλης και αποδοτικής μηχανής αναζήτησης παγκόσμιας κάλυψης αλλά συνάμα και η έμπνευση για την κατασκευή συστημάτων που στο μέλλον θα καθιστούσαν δυνατό να διαχειριστούμε δεδομένα της τάξης των **TByte** μέσα σε ελάχιστο χρονικό διάστημα.

Αυτή ήταν η αρχή για μία καινούρια εποχή στην διαχείριση της πληροφορίας την όποια και διαδέχτηκαν άλλες εταιρείες όπως η ***Yahoo***.Η τελευταία όπως θα δούμε στην πορεία, υιοθέτησε μία καινούρια ιδέα στηριζόμενη στην αρχική *Map/Reduce* της *Google*. Η ιδέα αυτή ήταν ένα πρωτοποριακό σύστημα διαχείρισης μεγάλου όγκου πληροφοριών, το ***Hadoop.*** Βασικότερο χαρακτηριστικό καθίσταται το γεγονός πώς έγινε Open Source Project, προσφέροντας την δυνατότητα να αναπτυχθούν συστήματα βασισμένα στα πλεονεκτήματα που προφέρει.Την λεπτομερή ανάλυση πάνω σε αυτό το σημείο θα την δούμε παρακάτω, αυτό που απλά θα αναφέρουμε εδώ είναι πως μέσω αυτής της δομής κα-

ταφέραμε για πρώτη φορά να επεξεργαστούμε σύνολα δεδομένων της τάξης των **TBytes** μέσα σε λίγα δευτερόλεπτα οπότε και καταφέραμε για πρώτη φορά να απομονώσουμε το πρόβλημα της μεγάλης πολυπλοκότητας που στο παρελθόν ήταν απαγορευτικό.

Κατά αυτό τον τρόπο μπορούμε σήμερα να πούμε πως έχουμε λύσει μερικώς αυτή την δυσκολία της συγκέντρωσης και διαχείρισης της παγκόσμιας διαδικτυακής πληροφορίας άρα καθίσταται πιο εύκολο για κάποιον να χτίσει μία διαφορετική μηχανή αναζήτησης που να υποστηρίζει λειτουργικότητες διαφορετικές από εκείνες που προσφέρουν οι πλέον διαδεδομένες των *Google, Yahoo.* Στην πράξη τέτοιες κινήσεις παρατηρήθηκαν αρκετά νωρίς μετά την εδραίωση των δυο προαναφερόμενων αν αναλογιστούμε σήμερα πόσες διαφορετικές μηχανές μπορούμε να βρούμε σε παγκόσμια κλίμακα.

Αν όμως έχουμε λύσει μερικώς το πρόβλημα της πολυπλοκότητας τί διαφορετικό μένει για να δούμε από μία μηχανή που απλά θα εξετάζει παγκοσμίως την διαθέσιμη πληροφορία; Η προφανής απάντηση σε αυτό το ερώτημα είναι το μοντέλο που υποστηρίζει κάθε μηχανή για την ανάλυση και επεξεργασία των κείμενων-ιστοσελίδων ώστε να απαντά στα ερωτήματα του εκάστοτε χρήστη. Και εστιάζουμε κυρίως στο μοντέλο γιατί μέχρι σήμερα οι προσπάθειες που έχουν καταγραφεί για την δημιουργία τέτοιων μηχανών βασίζονται κατά συντριπτική πλειοψηφία στο πολύ γνωστό σε όλους ***Boolean Model.***

Ο λόγος τώρα που έχει επικρατήσει μία τέτοια τακτική είναι για χάριν ευκολίας και ταχύτητας αφού βάσει αυτού του αλγορίθμου το μόνο που χρειάζεται να εξεταστεί ανάμεσα σε έναν αχανή όγκο πληροφορίας όπως αυτού του διαδικτύου, είναι μονάχα εκείνα τα sites που απλά εμπεριέχουν τους όρους που θα δώσει ο χρήστης σε κάθε επερώτημα. Σε αντίθεση βέβαια με άλλα μοντέλα όπως τα ***Vector Space*** και ***Probabilistic Model,*** η πολυπλοκότητα είναι ένα σοβαρό μειονέκτημα δίνοντας προβάδισμα στο *Boolean.* Αυτό που δεν μπορεί να υποστηρίξει όμως ένα τόσο απλοϊκό μοντέλο είναι η απομόνωση των κατάλληλων κείμενων στην περίπτωση που κάποιος δώσει ένα αίτημα με πάρα πολλούς όρους.

Αυτός είναι και ο λόγος που μέχρι σήμερα ένας απλός χρήστης περιορίζεται κατά την αναζήτηση κάποιου περιεχομένου να δώσει ένα μικρό αριθμό λέξεων. Αυτό είναι και το κίνητρο για να εξετάσουμε μια High Performance Search Engine, βασισμένη στο μοτίβο Map/Reduce συνδυάζοντας διάφορες τεχνικές και αλγορίθμους και κυρίως εκμεταλλευόμενοι τα πλεονεκτήματα του παραλληλισμού..

HADOOP

2.1 ΕΙΣΑΓΩΓΗ-ΠΡΟΒΛΗΜΑΤΙΣΜΟΙ

Ζούμε στην εποχή των πληροφοριών οπότε από πολύ νωρίς προέκυψε το πρόβλημα της διαχείρισης μεγάλου όγκου δεδομένων . Πολλές προσπάθειες στο παρελθόν για την δημιουργία προγραμμάτων που απαιτούσαν την επεξεργασία δεδομένων της τάξης Tbytes είχαν απαγορευτική πολυπλοκότητα. Αντιθέτως ,τα τεχνολογικά επιτεύγματα κατάφεραν μέσα σε λίγο χρονικό διάστημα να μας εξασφαλίσουν μεγάλη ευελιξία ως προς τον αποθηκευτικό χώρο.

Και ενώ οι δυνατότητες σε αποθηκευτικό χώρο με το πέρας του χρόνου αυξήθηκαν δραματικά, οι προσβάσεις στους δίσκους για επεξεργασία και συλλογή των δεδομένων δυστυχώς δεν συμβάδισαν. Αντιπροσωπευτικό παράδειγμα αποτελεί ένας σκληρός δίσκος το 1990 με χαρακτηριστικά 1370 MB αποθηκευτικού χώρου και 4.4MB/s. 20 χρόνια μετά ένα ο αποθηκευτικός χώρος έχει αυξηθεί στα 1TByte και ένας τυπικός χρόνος πρόσβασης είναι τα 100 MB/s. Εξετάζοντας τα δύο παραδείγματα με βάση το χρόνο, για να διαβάσουμε το σύνολο των δεδομένων βλέπουμε πως στην πρώτη περίπτωση απαιτούνταν 5 λεπτά ενώ στην δεύτερη 2,5 ώρες. Βλέπουμε λοιπόν πως τα πράγματα για εμάς σήμερα δεν είναι και ιδιαίτερα ευνοϊκά ιδίως όταν έχουμε να κάνουμε με πρόσβαση στο σύνολο των δεδομένων που μας ενδιαφέρουν από ένα και μόνο δίσκο.

Τι γίνεται όμως αν καταφέρουμε να διαβάσουμε τα δεδομένα μας από πολλαπλούς δίσκους ταυτόχρονα; Αν για παράδειγμα είχαμε 100 δίσκους στους οποίους και θα είχαμε διαμερίσει τα δεδομένα μας; Η ίδια διαδικασία θα γινόταν παράλληλα και θα είχε τελειώσει μέσα σε 2 λεπτά.

Πάνω σε αυτή την ιδέα στηρίχτηκαν οι εμπνευστές του ***Hadoop***, κατασκευάζοντας ένα σύστημα βασισμένο σε απλή επεξεργασία, με μεγάλες δυνατότητες επεκτασιμότητας και βαθμούς παραλληλισμού.

2.2 ΣΥΝΤΟΜΗ ΙΣΤΡΟΡΙΚΗ ΑΝΑΔΡΟΜΗ

Το ***Hadoop*** σχεδιάστηκε από τον Doug Cutting ο οποίος και έχει σχεδιάσει το **Lucene** ,την πολύ γνωστή βιβλιοθήκη αναζήτησης κειμένων. Η βασική ιδέα για τους σχεδιαστές του ***Hadoop*** ήρθε από το **Nutch**, κομμάτι του **Lucene** (Crawler).

Η αρχή έγινε το 2004 όταν η Google δημοσιοποίησε για πρώτη φορά την ιδέα του Map/Reduce. Μετά από ένα χρόνο οι σχεδιαστές του **Nutch** την υιοθέτησαν οπότε και το 2006 επιχειρήθηκε να δημιουργηθεί ένα ανεξάρτητο project ,το ***Hadoop***.

Την ίδια περίοδο ο Doug Cutting εντάχθηκε στην Yahoo οπότε και έδωσαν την βασική μορφολογία του Hadoop, ένα διακριτό και κλιμακωτό σύστημα που θα μπορούσε να εξυπηρετεί σε παγκόσμια κλίμακα τις απαιτήσεις ενός συστήματος αναζήτησης. Έτσι το 2008 ανακοινώθηκε από την Yahoo πως το σύστημα αναζήτησής της βασιζόταν στο Hadoop.

Γρήγορα, και άλλες μεγάλες εταιρείες όπως οι Last.fm, Facebook, *New York Times,* Amazon,υιοθέτησαν το ***Hadoop*** και κατάφεραν βάσει των πλεονεκτημάτων που παρείχε να διαχειρίζονται τεράστιο όγκο δεδομένων. Χαρακτηριστικό παράδειγμα αποτελεί η προσπάθεια της *York Times*[1] σε συνεργασία με την Amazon να μετατρέψουν 4TBytes συμπιεσμένων αρχείων σε Pdfs. Η διαδικασία διήρκησε λιγότερο από 24 ώρες με 100 μηχανές που 'έτρεχαν' το ***Hadoop***.

Τα επιτεύγματα του Hadoop συνεχίστηκαν οπότε και το 2008 έγινε το πιο γρήγορο σύστημα διαχείρισης και ταξινόμησης δεδομένων της τάξης των TBytes. Ενδεικτικά, σε ένα cluster με 910 κόμβους κατάφερε σε λιγότερο από 3,5 λεπτά να ταξινομήσει 1TByte δεδομένων. Ο ανταγωνισμός μεταξύ των εταιρειών συνεχίστηκε με την Google[2] ανακοινώνοντας το 2008 πως για τον ίδιο όγκο δεδομένων κατάφερε να ταξινομήσει δεδομένα σε 68 sec χρησιμοποιώντας την δική της υλοποίηση Map/Reduce. Λίγο αργότερα η yahoo βασιζόμενη στο ***Hadoop*** 'έσπασε ' το ρεκόρ με 62 sec.

2.3 ΑΝΑΛΥΣΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ HADOOP

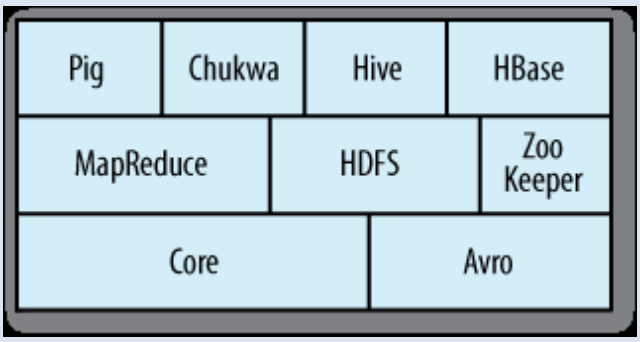
Το Hadoop είναι ένα πρόγραμμα το οποίο αποτελείται από πολλά διαφορετικά τμήματα. Ο συνδυασμός όλων προσδίνει την μοναδική δυνατότητα του Hadoop να συνδυάζει ταυτόχρονα πολλά δεδομένα και να λειτούργει αυτόνομα και ανεξάρτητα σε διαφορετικές μηχανές .Το βασικότερο χχαρακτηριστικό που το ξεχωρίζει από τα άλλα συστήματα είναι η δυνατότητα **Map/Reduce** σε συνδυασμό με ένα κατανεμημένο σύστημα, γνωστό ως **HDFS**. Επιπρόσθετες εφαρμογές του Hadoop παρουσιάζονται παρακάτω περιληπτικάι στην συνέχεια θα δώσουμε έμφαση στα στοιχεία Map/Reduce –Hdfs ,τα οποία και μας βοήθησαν στην εργασία μας.

---

[1] Derek Gottfrid, "Self-service, Prorated Super Computing Fun!," 1 November 2007, *http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/*.

[2] Sorting 1PB with MapReduce," 21 November 2008, *http://googleblog.blogspot.com/2008/11/sorting-1pb-with-mapreduce.html*.

- **Core**
  Σύνολο από components και interfaces για κατανεμημένα συστήματα και υπηρεσίες I/O.
- **Avro**
  Σύστημα συγχρονισμού για αποδοτική αποθήκευση των δεδομένων.
- **Pig**
  Περιβάλλον που τρέχει πάνω σε HDFS και Map/Reduce clusters και εξετάζοντας μεγάλο όγκο δεδομένων.
- **HBase**
  Κατανεμημένη βάση δεδομένων που χρησιμοποιεί το HDFS για αποθήκευση. Υποστηρίζει batch-style υπολογισμούς χρησιμοποιώντας το Map/Reduce και τυχαία queries.
- **Zookeeper**
  Κατανεμημένη υπηρεσία που προσφέρει βασικές λειτουργίες όπως locks για δημιουργία κατανεμημένα εφαρμογών.
- **Hive**
  Κατανεμημένο σύστημα αποθήκευσης που διαχειρίζεται τα δεδομένα στο HDFS. Υποστηρίζει επερωτήματα Sql τα οποία αναλύονται σε Map/Reduce Jobs.
- **Chukwa**
  Κατανεμημένη συλλογή δεδομένων που 'τρέχει' τους Collectors – υπηρεσία αποθήκευσης δεδομένων στο Hdfs χρησιμοποιώντας το Map/Reduce .



Hadoop Subprojects

2.3.1 MAP/REDUCE

Το Map/Reduce είναι ένα μοντέλο για διαχείριση δεδομένων. Ως έννοια είναι αρκετά απλή αφού βασίζεται στην παράλληλη επεξεργασία της πληροφορίας υποστηρίζοντας πολλές διαφορετικές γλώσσες προγραμματισμού όπως Java,Python,C++. Αυτά τα χαρακτηριστικά καθιστούν το Map/reduce και κατ' ε-

πέκταση το Hadoop, ένα εργαλείο που μπορεί πολύ εύκολα να εξετάσει μεγάλου όγκου πληροφορία. Με αυτό τον τρόπο έγινε προσιτή σε κάθε απλό χρήστη η δυνατότητα να σχεδιάσει προγράμματα με τα παραπάνω χαρακτηριστικά και να υλοποιήσει εφαρμογές με μικρή χρονική καθυστέρηση.

Ειδικότερα, για να δείξουμε πώς υλοποιείται μέσα σε γενικά πλαίσια ένα προγραμματιστικό μοντέλο MapReduce, ας δούμε μέσα από ένα παράδειγμα υπολογισμού του αριθμού των εμφανίσεων κάθε όρου μέσα σε μία συλλογή από μεγάλα data sets. Ο χρήστης στην προσπάθεια να κάνει Implementation την Map μέθοδο θα έγραφε κώδικα παρόμοιο με τον εξής ψευδοκώδικα:

Map(String key, String value)

//key: document name
//value: document contents
For eachword w in value:
EmitIntermediate(w,"l");

Reduce(String key, Integer value)

//key: a word
//value: a lsit of counts
For each v in values:
Result+=ParseInt(v);
Emit(AsString(result));

Κατ' αυτόν τον τρόπο η συνάρτηση map διοχετεύει στον κάθε όρο και έναν αριθμό συσχετιζόμενο με τις εμφανίσεις κάθε λέξης(εδώ 1). Τα αποτελέσματα από αυτή την διαδικασία του mapping ταξινομούνται και διοχετεύονται ως είσοδοι στους reducers. Επομένως, η reduce συνάρτηση αυτό που θα κάνει είναι απλά να αθροίσει όλες τις εμφανίσεις κάθε λέξης που της διοχετεύτηκαν από τον Mapper. Τα αποτελέσματα από αυτές τις διαδικασίες τοποθετούνται στο Hdfs σύστημα.

Επίσης, το Hadoop αυτόματα παραλληλοποιεί και εκτελεί το πρόγραμμα σε μεγάλα clusters αφήνοντας στο σύστημα τις λεπτομέρειες του διαχωρισμού των δεδομένων εισόδου, τον προγραμματισμό κάθε διεργασίας σε κάθε κόμβο και αντιμετωπίζοντας τις αστοχίες που πιθανώς να συμβούν. Κατ΄ αυτόν τον τρόπο δύναται να επεξεργαστούν δεδομένα της τάξης των TBytes σε λίγο χρονικό διάστημα χρησιμοποιώντας εκατοντάδες χιλιάδες κόμβων που υποστηρίζουν το Hadoop. Ενδεικτικά, στην Google, πάνω από100.000 MapReduce εργασίες εκτελούνται καθημερινά και με άμεση ανταπόκριση προς τον χρήστη.

Πώς όμως μπορούμε να ξέρουμε τα αποτελέσματα αυτής της διαδικασίας του mapping και βάση ποιάς λογικής γίνεται η επιλογή των δεδομένων προς ταξινόμηση; Η απάντηση σε αυτό θα δωθεί παρακάτω με την επεξήγηση μέσω ενός παραδείγματος.

Η λογική στο παραπάνω είναι λίγο πολύ γνωστή. Ψάχνοντας ανάμεσα σε μια συστάδα δεδομένων προσπαθούμε να φτιάξουμε ομάδες με κριτήριο την παράμετρο *key.* Έτσι και σε αυτή την περίπτωση, οι mappers καταλαβαίνουν πως τα δεδομένα που θα προωθήσουν στο επόμενο επίπεδο θα επιλεγούν από τα αρχεία εισόδου ομαδοποιώντάς τα βάσει της προκείμενης παραμέτρου, ενώ από αυτή την ομαδοποίηση αυτά που θα μεταδώσουν στους reducers θα είναι η παράμετρος *data_to_reducers*.

Χρησιμοποιώντας παρόμοια λογική και στο επόμενο επίπεδο του reduce τα δεδομένα *data_to_reducers* ως ομάδες ανεξάρτητες τροποποιούνται κατά τρόπο που ορίζει ο χρήστης. Για παράδειγμα , αν θέλουμε να μετρήσουμε πόσα κοινά στοιχεία υπάρχουν σε κάθε ομάδα που δίνεται σε ένα reducer τότε κάνουμε το implementation του Reducer με τρόπο που να υπολογίζει τις κοινές εμφανίσεις. Τα αποτελέσματα αυτής της διαδικασίας θα είναι η έξοδος του προγράμματος .

2.3.2 MAP/REDUCE –ΠΑΡΑΔΕΙΓΜΑ ΕΦΑΡΜΟΓΗΣ

Για να γίνουν πιο κατανοητά τα πράγματα ας παραθέσουμε ένα παράδειγμα επεξεργασίας μετεωρολογικών δεδομένων .

Στην πράξη η συλλογή δεδομένων μίας τέτοιας εφαρμογής απαιτεί να λαμβάνουμε στοιχεία ανά μία ώρα σε διάφορα σημεία του κόσμου. Είναι φανερό λοιπόν πώς σε μία τέτοια περίπτωση έχουμε πολλή πληροφορία οπότε και θα επιχειρήσουμε να την επεξεργαστούμε μέσω του Hadoop.

Ας υποθέσουμε λοιπόν πως τα δεδομένα που συλλέγουμε έχουν την παρακάτω μορφή zip αρχείων και αφορούν μετρήσεις που έχουμε λάβει από τα έτη 1901-2001. Πχ Για το έτος 1990 έχουμε.

010010-99999-1990.gz
010014-99999-1990.gz
010015-99999-1990.gz
010016-99999-1990.gz
010017-99999-1990.gz
010030-99999-1990.gz
010040-99999-1990.gz
010080-99999-1990.gz
010100-99999-1990.gz
010150-99999-1990.gz

Στο παραπάνω παράδειγμα εφαρμόζουμε το Hadoop και ο σκοπός μας είναι να απομονώσουμε για κάθε έτος την τιμή της θερμοκρασίας και τελικά να υπολογί-

σουμε την μέγιστη θερμοκρασία κάθε έτους . Για να το κάνουμε αυτό πρέπει να ενημερώσουμε τον mapper να απομονώσει από κάθε γραμμή του κειμένου το ***έτος(key)*** και την ***θερμοκρασία (data_to_reducers)***. Έπειτα, οι reducers θα λάβουν ως είσοδο τις ομαδοποιημένες και ταξινομημένες ανά έτος θερμοκρασίες από εκεί θα επιλέξουν ποια από όλες είναι οι μεγαλύτερη.

Να σημειώσουμε επίσης πώς οι είσοδοι για την διαδικασία είναι αρχεία .txt στα οποία ανά γραμμή υπάρχουν πληροφορίες για διάφορα μετρήσιμα μεγέθη ανά ώρα, μέρα, έτος.

Για να καταλάβουμε πώς λειτουργεί ένας mapper ας δούμε το αρχείο εισόδου.

0067011990999991950051507004...9999999N9+00001+99999999999...
0043011990999991950051512004...9999999N9+00221+99999999999...
0043011990999991950051518004...9999999N9-00111+99999999999...
0043012650999991949032412004...0500001N9+01111+99999999999

Πάνω σε κάθε γραμμή δρα κάποιος mapper και με κατάλληλο κώδικα μέσα στην συνάρτηση map() του λέμε να απομονώσει από κάθε γραμμή το έτος και την θερμοκρασία .

(1950, 0)
(1950, 22)
(1950, −11)
(1949, 111)
(1949,78)

Οι έξοδοι σε αυτό το σημείο ταξινομούνται αυτόματα από το Hadoop(δεν απαιτείται κώδικας από τον χρήστη) και διοχετεύονται στον reducer σε ομάδες ως είσοδοι όπως φαίνεται παρακάτω.

(1949, [111, 78])

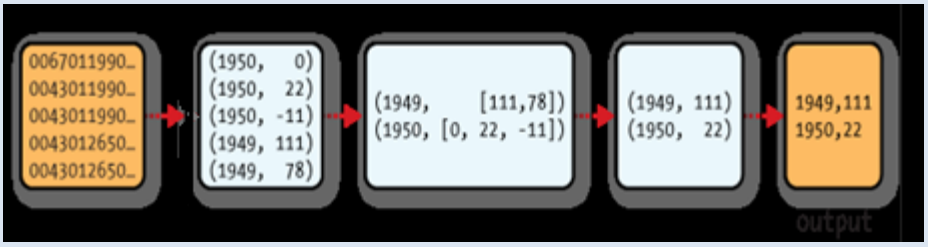
(1950, [0, 22, −11])

Στο τελευταίο βήμα αυτό που απομένει είναι απλά με κατάλληλο κώδικα στην συνάρτηση reduce() να απομονώσουμε για κάθε έτος την μέγιστη θερμοκρασία. Τα δεδομένα αυτά τοποθετούνται σε μορφή κειμένων στο σύστημα HDFS .

(1949, 111)

(1950, 22)

Για να έχουμε μία καλύτερη οπτική της διαδικασίας παρατίθεται μία πιο συγκεντρωτική εικόνα.



MapReduce logical data flow

Ένα άλλο στοιχείο στο οποίο πρέπει να σταθούμε είναι οι τύποι των ορισμάτων που υπάρχουν στον mapper, όπως και τα ορίσματα του Output(). Όπως βλέπουμε δεν είναι της μορφής των βασικών τύπων της java(πχ int, String) αλλά τύπου **IntWritable,Text.** Οι συγκεκριμένοι τύποι είναι ειδικοί τύποι δεδομένων του Hadoop οι οποίοι και αντιστοιχούν στους κλασικούς τύπους Integer, String. Ομοίως αν θέλαμε να διαχειριστούμε δεδομένα τύπου Long θα χρησιμοποιούσαμε τον τύπο **LongWritable**.

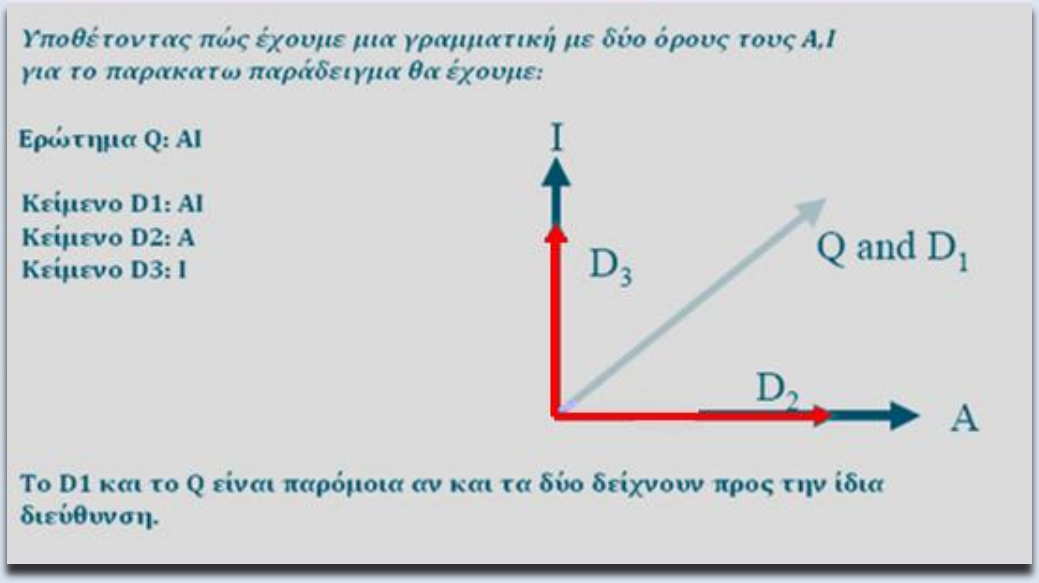
JavaTypes	Hadoop Types	Serialized Size(bytes)
boolean	BooleanWritable	1
byte	ByteWritable	1
int	IntWritable	4
	VIntWritable	4-5
float	FloatWritable	4
long	LongWritable	8
	VFloatWritable	1-9
double	DoubleWritable	8

Βασικοί Τύποι Δεδομένων Hadoop:

3.1 TEXT REPRESENTATION- VECTOR SPACE

Στο συγκεκριμένο κεφάλαιο θα μιλήσουμε για την ανάλυση και αναπαράσταση των κειμένων μέσω του *Vector Space Representation*. Πρόκειται για ένα μοντέλο το οποίο εισηγήθηκε για πρώτη φορά ο *Salton* (1975) και στηρίζεται στην αναπαράσταση των κειμένων και των επερωτήσεων σε μορφή διανυσμάτων ***M***-διαστάσεων, όπου ***M*** είναι ο αριθμός των όρων . Κατά αυτό τον τρόπο ένα κείμενο έχει παρόμοιο περιεχόμενο με την ερώτηση του χρήστη μονάχα αν τα αντίστοιχα διανύσματα είναι παρόμοια.

Χαρακτηριστικό παράδειγμα για να καταλάβουμε το μέτρο σύγκρισης των διανυσμάτων αποτελεί η παρακάτω αναφορά.



Εικόνα 1: Αναπαράσταση Διανυσμάτων *Vector Space Model*

Πώς όμως γίνεται αυτή η μετάβαση από το σύνολο των όρων των κειμένων σε μία δομή διανυσμάτων; Στην πιο εύκολη περίπτωση θα μπορούσαμε να δίνομε την ίδια βαρύτητα σε όλους τους όρους και απλά να επιλέγαμε τα κείμενα ανάλογα τη συχνότητα εμφάνισης κάποιας λέξης που μας ενδιαφέρει . Στην πράξη πάνω σε αυτή την σκέψη στηρίχτηκε η ανάλυση αυτού του αλγορίθμου απλά η σημαντικότητα των όρων είναι και τα κριτήριο για τον χαρακτηρισμό ενός κείμενου. Από αυτά και μόνο καταλαβαίνουμε πως για να πετύχουμε την διανυσματική αναπαράσταση πρέπει πρώτα να δώσουμε ένα βάρος σε κάθε όρο για να αποδώσουμε την βαρύτητα της κάθε λέξης σε κάθε κείμενο.

Για να γίνουμε και πιο συγκεκριμένοι , η αναγωγή των κειμένων σε διανύσματα στηρίχθηκε στον υπολογισμό των παρακάτω παραμέτρων.

- **Term Frequency** (TF_i): Ο αριθμός των εμφανίσεων ενός όρου ***i*** σε ένα κείμενο **j**(TF_{ij}).
- **Document Frequency** (DF_i): Ο αριθμός των κειμένων μέσα στα οποία εμφανίζεται ο όρος ***i.***
- **Inverse Document Frequency** (IDF_i)**:** Μέτρο για να αποδώσουμε πόσο χαρακτηριστικός είναι ένας όρος ***i***. Για να υπολογίσουμε αυτό το μέτρο χρησιμοποιούμε $IDF_i=\log_{10}\frac{D}{DFi}$, όπου το D είναι το σύνολο των κειμένων που υπάρχουν μέσα σε μία συλλογή.

Από τα παραπάνω λοιπόν, βλέπουμε πως για να χαρακτηρίσουμε ένα κείμενο σημαντικό ρόλο παίζει πόσο συχνά εμφανίζεται ένας όρος μέσα σε ένα κείμενο αλλά και ως προς όλο το σώμα των κειμένων μιας συλλογής.

Επομένως, για να υπολογίσουμε το κάθε βάρος που προσδίδει ένας όρος στο κείμενο αρκεί να βρούμε $\boldsymbol{w_i = TFi * IDFi}$, ενώ για να βρούμε το συνολικό βάρος ενός κειμένου απλά υπολογίζουμε την ρίζα του αθροίσματος των τετράγωνων των επιμέρους w_i από όλους τους όρους του κειμένου δηλ,

$$\boldsymbol{WDj} = \sqrt{\sum_{i=1}^{M} (\boldsymbol{wi})^2}$$, όπου M σύνολο όρων του κειμένου Dj.

Τί πετυχαίνουμε με αυτό τον τρόπο; Ουσιαστικά αναπαριστούμε το κάθε κείμενο σε μορφή διανύσματος οπότε μπορούμε εύκολα να κάνουμε την ίδια διαδικασία και για ένα ερώτημα *Q*. Για να συγκρίνουμε τώρα αν το κάθε Q πλησιάζει σημασιολογικά ένα κείμενο D απλά πρέπει να υπολογίσουμε το *συνημίτονο* ανάμεσα στα δύο διανύσματα. Όσο πιο μικρή είναι αύτη η τιμή τόσο πιο όμοια είναι τα $Q_{,}$, D_j .Άρα τελικά έχουμε:

$$\boldsymbol{Sim\ (Q, D_j) = cos\ (\theta_{Dj,Q}) = \frac{Q,Dj}{|Q||Dj|} = \frac{\sum_j W_{i,j} W_{Q,i}}{\sqrt{\sum_i w_{Q,i}^2}\sqrt{\sum_j w_{i,j}^2}}}$$

Η παραπάνω αναπαράσταση είναι ο βασικός τρόπος με τον οποίον προσεγγίζομε το συγκεκριμένο μοντέλο. Ωστόσο, υπάρχουν και άλλες παραλλαγές που στηρίζονται στην κανονικοποίηση των βαρών όπου τα βάρη μετατίθενται στο πεδίο[0, 1]

$$w_{ij} = \frac{(1+\log(tf_{ij}))idf_i}{\sqrt{\sum_{k\neq i}^{M}(1+\log(tf_{kj}))^2}}$$

ή στην κανονικοποίηση των βαρών στηριζόμενοι στο μέγεθος του κειμένου. Την τελευταία την κάνουμε κυρίως για τα κείμενα μεγάλου μεγέθους καθώς κείμενα με μεγαλύτερο αριθμό όρων είναι πιο πιθανό να περιέχουν κάποιους όρους από το ερώτημα του χρήστη.

$$w'_{ij} = \frac{w_{ij}}{\sqrt{\sum_{k=1}^{M} w_{kj}^{2}}}$$

Κατά αυτόν τον τρόπο καταφέρνουμε να προσεγγίσουμε από ένα σύνολο δεδομένων εκείνα που περιγράφουν καλύτερα το ερώτημά μας. Η ακρίβεια του μοντέλου αυτού είναι αρκετά καλή σε σχέση με άλλα ,πχ *Boolean Model* που απλά εντοπίζει τα κείμενα που περιέχουν τους όρους του χρήστη, και πολύ καλύτερη όταν οι όροι του ερωτήματος του χρήστη αυξάνονται. Αυτό είναι και το βασικό σημείο που το ξεχωρίζει από άλλες μοντελοποιήσεις. Επίσης, η εξαγωγή των ομοιοτήτων ανάμεσα σε ερώτημα και κείμενο δεν χρειάζεται να στηρίζεται ανάμεσα σε ακριβώς ίδιους όρους ενώ μέσω του υπολογισμού του συνημίτονου $\cos(\theta_{Q,Dj})$ μπορούμε να εξάγουμε βαθμονομημένα αποτελέσματα ομοιότητας .

Τα βασικά μειονεκτήματα αυτής της μοντελοποίησης είναι πως θεωρούμε ότι οι όροι μεταξύ τους είναι ανεξάρτητοι . Στο κύριο σημείο όμως που υστερεί είναι η πολυπλοκότητα της υλοποίησης καθώς είναι πιο μεγάλη σε σχέση με άλλους αλγορίθμους(πχ *Boolean Model*). Αυτός είναι και ο βασικός λόγος που πολλές σύγχρονες μηχανές αναζήτησης επιλέγουν να υποστηρίξουν το Boolean μοντέλο, χάριν ευκολίας και ταχύτητας.

3.2 ΠΡΟΕΠΕΞΕΡΓΑΣΙΑ ΚΕΙΜΕΝΩΝ

Προφανώς πριν ξεκινήσουμε την ανάλυση του προκείμενου μοντέλου έπρεπε να διευκρινίσουμε πώς απομονώνουμε το σύνολο των όρων και τι διαδικασίες ακολουθούνται για να ξεχωρίσουμε τις πιο αντιπροσωπευτικές λέξεις. Από αυτά που έχουμε αναφέρει μέχρι στιγμής γίνεται κατανοητό πώς πολλοί από τους όρους ενός κειμένου δεν μπορούν να αποδώσουν το νόημά . Χαρακτηριστικό παράδειγμα αποτελούν οι ανωνυμίες ,άρθρα ,προθέσεις ,επιρρήματα, σημεία στίξης κλπ.

Ρίχνοντας μία απλή ματιά σε ένα κείμενο εύκολα παρατηρούμε πώς η πλειοψηφία των λέξεων είναι συνδετικές αντωνυμίες πχ και-and άρθρα κλπ. Αν αποφασίζαμε να συμπεριλάβουμε στην ανάλυση και αυτούς τους όρους το μόνο που θα πετυχαίναμε θα ήταν να δώσουμε μεγαλύτερη βαρύτητα σε αυτές τις λέξεις

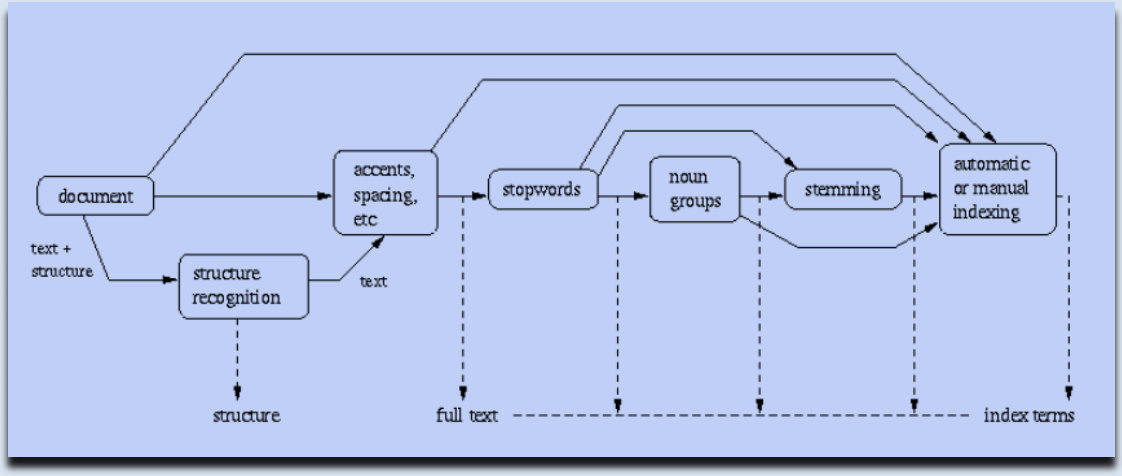
οπότε θα «παραπλανούσαμε» το μοντέλο αφού δεν θα στηρίζαμε την ανάλυση σε όρους που αποδίδουν την βαθύτερη έννοια.

Με άλλα λόγια έπρεπε με κάποιο τρόπο να απομονώσουμε μονάχα τις λέξεις που συνοψίζουν την ουσία οπότε και επιλέξαμε να κρατήσουμε κυρίως τα ουσιαστικά και τα ρήματα. Οι υπόλοιποι όροι (Stop Words) αγνοήθηκαν και για να γίνει αυτό οριοθετήσαμε μία λίστα με ένα σύνολο από λέξεις που θέλαμε να μην συμπεριλάβουμε- *stopwordlist*. Κάτι τέτοιο βέβαια προϋποθέτει για κάθε κείμενο ξεχωριστά να διαχωρίσουμε πρωτίστως ανά πρόταση τους όρους και έπειτα ανά όρο που συναντάμε να προβούμε στην εξέταση αν ανήκει η όχι στο σύνολο της λίστας που φτιάξαμε.

Αυτό ήταν και το πρώτο βήμα για να μειώσουμε τον όγκο των δεδομένων προς επεξεργασία . Το δεύτερο βήμα προς αυτή την κατεύθυνση είναι να αποκόψουμε τις περιττές καταλήξεις σε κάθε όρο και να αφήσουμε μονάχα το σώμα της λέξης. Η διαδικασία αυτή ονομάζεται *Stemming* και πραγματοποιείται στα πλαίσια μείωσης περιττής πληροφορίας.

Για να καταλάβουμε καλύτερα το βήμα αυτό ας σκεφτούμε το εξής. Ας πούμε πως εξετάζουμε ένα κείμενο στο οποίο και συναντάμε τις λέξεις "*apple*" ,"*apples*". Προφανώς αν επιλέγαμε να αντιπροσωπεύσουμε τους όρος αυτούς ως ξεχωριστές οντότητες θα είχαμε πράξει λάθος γιατί και οι δύο συνοψίζουν το ίδιο νόημα και εμείς θα τις δεικτοδοτούσαμε άσκοπα δύο φορές. Κατ' επέκταση θα χάναμε το πραγματικό αριθμό των εμφανίσεων του όρου αυτού που ενσωματώνει το νόημα της λέξης 'apple' ,δηλαδή θα μετρούσαμε μια εμφάνιση αντί για δύο, και έτσι θα επηρεαζόταν και οι παράμετροι TF_i, IDF_i.

Για όλους αυτούς τους λόγους πριν από την εφαρμογή του μοντέλου επεξεργαζόμαστε τα κείμενα με το περιεχόμενό τους και έπειτα προβαίνουμε σε υπολογισμούς των παραμέτρων για το μοντέλο μας *Vector Space*. Οι διαδικασίες αυτές συνοψίζονται στην κλάση *TextAnalyser, σελ113,* στην οποία και θα επικεντρωθούμε στον τρόπο με τον οποίο έγιναν οι υλοποιήσεις.



Εικόνα 2. *Σχηματικό Διάγραμμα Προεπεξεργασίας Κειμένων Baeza-Yates Ribeiro -Neto*

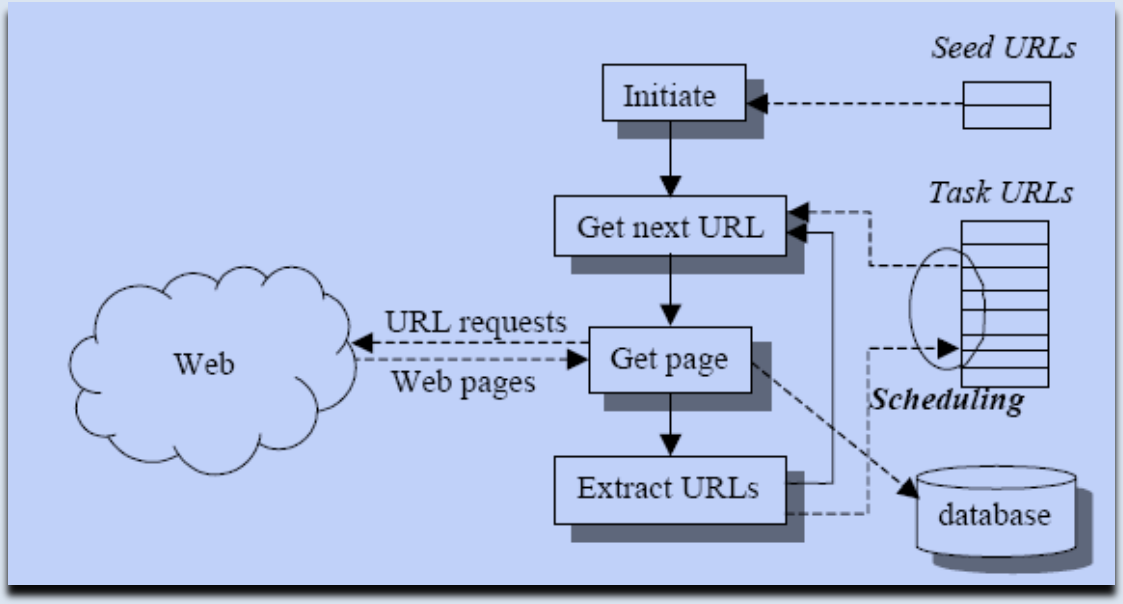
3.3 ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ –ΕΝΤΟΠΙΣΜΟΣ ΝΕΩΝ ΔΙΕΥΘΥΝΣΕΩΝ

Όπως θα δούμε παρακάτω, βασικό τμήμα της εργασίας αποτελεί ο Crawler μέσω του οποίου κατεβάζουμε το περιεχόμενο των ιστοσελίδων. Προφανώς, έπρεπε πρωτίστως να αποφασίσουμε με ποιόν τρόπο θα προγραμματίζαμε τα εξαγόμενα Urls που θα εντοπίζαμε κάθε φορά. Πάνω σε αυτό το ζήτημα υπάρχουν πολλές διαφορετικές πολιτικές με καθεμία να εξυπηρετεί διαφορετικούς σκοπούς(πολιτικές φιλικές προς το εκάστοτε site βάσει επισκεψιμότητας ή όχι κλπ). Σε αυτή την παράγραφο θα δούμε κατά κύριο λόγο τον αλγόριθμο που υλοποιήσαμε προς αυτό τον σκοπό αλλά και ορισμένες αναφορές σε άλλες τακτικές.

Πρίν ξεκινήσουμε την ανάλυση ας εστιάσουμε πρώτα στην γενικότερη φιλοσοφία προγραμματισμού των μελλοντικών διευθύνσεων. Όπως θα δούμε τα κύρια σημεία αυτής της διαδικασίας συνοψίζονται στα παρακάτω βήματα.

1. ***Αρχικοποίηση Crawler θέτοντας μία λίστα από αρχικά Urls.***
2. ***Επέλεξε κάποιο-α από Urls από το σύνολο και κατέβασέ τα.***
3. ***Από την τρέχουσα σελίδα εξήγαγε τα εσωτερικά Urls και τοποθέτησε τα στην αρχική λίστα προς επίσκεψη.***
4. ***Επανέλαβε τα 2,3 μέχρι να αδειάσει η λίστα είτε να σταματήσει η εφαρμογή.***

Η στρατηγική τώρα που χρησιμοποιούμε για να τοποθετήσουμε τις νέες διευθύνσεις είναι ο λεγόμενος προγραμματισμός του Crawler και πάνω σε αυτό το θέμα υπάρχουν αρκετές προσεγγίσεις.



Γενικό Μοντέλο Crawler.

Το internet έχει γίνει αρκετά απαιτητικό αφού για έναν τυπικό *Crawler* το fetching 1.000.000.000 σελίδων είναι περίπου πάνω από μία εβδομάδα. Εν τω μεταξύ βέβαια, έχουν δημιουργηθεί καινούριες οι οποίες και δεν έχουν δεικτοδοτηθεί από το σύστημα οπότε όπως είναι φυσικό αυτές οι νέες διευθύνσεις δεν έχουν προγραμματιστεί προς επίσκεψη στο μέλλον. Αυτό σημαίνεις πώς όταν μιλάμε για προγραμματισμό ενός Crawler εννοούμε τα Urls που έχουμε ήδη εντοπίσει στον παγκόσμιο ιστό και έχουμε προγραμματίσει να δούμε στο άμεσο μέλλον.

Όσον αφορά τώρα τις στρατηγικές που μπορεί κάποιος να ακολουθήσει προς αυτό τον σκοπό .σίγουρα υπάρχουν αρκετές. Εμείς εδώ θα εξετάσουμε τις ***Breadth-First*** και ***Quality based scheduling.***

3.3.1 Breadth-first Scheduling

Πρόκειται για έναν αρκετά διάσημο αλγόριθμο κατά το οποίο και τα νέα Urls τοποθετούνται σε μία ουρά βάσει της σειράς με την οποία εντοπίζονται. Ο τρόπος τώρα με το οποίο εντοπίζονται τα Urls είναι μέσω των εσωτερικών link της κάθε σελίδας που εξετάζεται. Επομένως, αν θέλουμε να εξετάσουμε μία καινούρια σελίδα απλά επισκεπτόμαστε το στοιχείο που υποδεικνύει η κεφαλή της ουράς και τα νέα δεδομένα που θα εντοπίσουμε στο εσωτερικό θα τα τοποθετήσουμε στο τέλος. Αυτό σημαίνει πως κάνουμε μία οργάνωση τύπου FIFO.

Ποία είναι τα πλεονεκτήματα αυτής της στρατηγικής; Εκτός από την ευκολία υλοποίησης το κυριότερο είναι ότι τα εξαγόμενα στοιχεία που τοποθετούνται στην ουρά έχουν πολύ καλές διασυνδετικές ιδιότητες. Για αυτούς τους λόγους θεωρείται μία αρκετά καλή τεχνική εντοπισμού νέων διευθύνσεων.

3.3.2 Quality Based Scheduling

Αυτός ο αλγόριθμος στηρίζεται στον προγραμματισμό νέων διευθύνσεων δίνοντας προτεραιότητα σε site που έχουν καλή ποιότητα. Το κριτήριο τώρα για την εκτίμηση της ποιότητας ενός site θεωρούμε πως δίνεται από μία άλλη εφαρμογή οπότε όπως γίνεται αντιληπτό κάθε φορά μπορούμε να δίνουμε το προβάδισμα σε διευθύνεις που θέλουμε εμείς ανάλογα τους σκοπούς της εφαρμογής μας. Και πάλι θεωρούμε πως τις εξαγόμενες διευθύνσεις τις τοποθετούμε σε ουρές όπου και ακολουθείται η ίδια τακτική εισαγωγής και εξαγωγής των στοιχείων.

Κατά αυτόν τον τρόπο, οι ουρές που κατασκευάζονται κατά την εύρεση νέων διευθύνσεων είναι ταξινομημένες βάσει των ιδιοτήτων κάθε σελίδας. Έτσι ,αν αποδίδαμε σε έναν Server κάποιο μέτρο αξιολόγησης, θα ορίζαμε για παράδειγμα ως πρώτο προς εξέταση Url εκείνο που ήταν και καλύτερο στο Server . Επομένως, πετυχαίνουμε από το σύνολο των Urls ενός Server να δώσουμε προτεραιότητα σε εκείνα που είναι πιο σημαντικά για εμάς ακολουθώντας τα λιγότερο σημαντικά.

ΑΝΑΛΥΣΗ ΤΜΗΤΑΤΩΝ ΕΡΓΑΣΙΑΣ

4.1 ΠΕΡΙΓΡΑΦΗ ΕΡΓΑΣΙΑΣ -ΣΚΟΠΟΣ

Ήδη από τα παραπάνω έχει σκιαγραφηθεί μέσα σε γενικά πλαίσια η φόρμα που επιλέξαμε να έχει η μηχανή μας. Στην πορεία θα δούμε τον τρόπο που δομήσαμε κάθε τμήμα ώστε να κατασκευάσουμε το ζητούμενο. Τα δύο βασικά στοιχεία όμως που πρέπει να πούμε αρχικά είναι ότι υιοθετήσαμε την αναπαράσταση *Vector Space* για να κάνουμε την ανάλυση των στοιχείων κάθε ιστοσελίδας και επίσης χρησιμοποιήσαμε το *Hadoop* για να μπορέσουμε να κάνουμε τις διαδικασίες που θέλαμε εύκολα και γρήγορα.

Πιο συγκεκριμένα, κάθε μηχανή αναζήτησης ανεξάρτητα από την κλίμακα των πληροφοριών που εξετάζει διαχωρίζεται σε δύο μεγάλα τμήματα. Στο τμήμα εξερεύνησης και συλλογής των διευθύνσεων από τον παγκόσμιο ιστό ,τον **Crawler** και στο τμήμα της επεξεργασίας και ανάλυσης του περιεχομένου των ιστοσελίδων για την υλοποίηση κάποιου **IR** μοντέλου. Προφανώς για να μπορέσουμε να κατασκευάσουμε ένα αποδοτικό και ευέλικτο σύστημα θα πρέπει να επιλέξουμε μία αρχιτεκτονική που να είναι όσο το δυνατόν πιο κατανεμημένη και να μπορεί να ανανήπτει από τυχόν αστοχίες .

Τα δυο αυτά στοιχεία είναι ιδιαίτερα σημαντικά για τέτοια συστήματα και μάλιστα όταν έχουμε να κάνουμε με μεγάλο όγκο πληροφορίας. Σε άλλη περίπτωση αν δεν φροντίζαμε να υπάρχει κάποια μορφή παραλληλισμού στα προκείμενα τμήματα θα αντιμετωπίζαμε σημαντικά προβλήματα. Σκεφτείτε για παράδειγμα τί δυσκολίες θα αντιμετωπίζαμε αν είχαμε επιλέξει να μην καταμερίζαμε πχ τον *Crawler* αλλά να δίναμε όλο τον υπολογιστικό φόρτο σε ένα μόνο μηχάνημα. Προφανώς θα χάναμε άσκοπα πολύτιμο χρόνο την στιγμή που θα μπορούσαμε πολύ εύκολα να κάνουμε την ίδια διαδικασία και άλλα.

Και όχι μόνο. Όπως είπαμε μας ενδιαφέρει η υλοποίηση μας να είναι αποδοτική άρα βασικό στοιχείο που έπρεπε να εξετάσουμε είναι πως θα παραλληλοποιήσουμε την όλη διαδικασία ώστε στον ίδιο χρόνο να κάνουμε περισσότερες διεργασίες. Για όλα αυτά τα παραπάνω λύση έδωσε το Hadoop γιατί όπως θα δούμε και στο αντίστοιχο κεφάλαιο η αρχιτεκτονική του είναι τέτοια ώστε να μπορεί να σπάει τις δουλειές τις οποίες του αναθέτουμε και στο ίδιο χρόνο ενεργεί σε κάθε επιμέρους τμήμα. Έτσι πετυχαίνουμε τον καταμερισμό και παραλληλισμό των διεργασιών εξοικονομώντας χρόνο, μειώνοντας τον φόρτο και τον κίνδυνο να έχουμε αστοχίες στο σύστημα.

4.1.1 Λειτουργικότητες -Απαιτήσεις

Συνοψίζοντας τα παραπάνω αυτά που θέσαμε ως στόχους είναι : παραλληλοποίηση, καταμερισμό εργασιών, σχετική ανεξαρτησία των επιμέρους τμημάτων, αντιμετώπιση πιθανών αστοχιών. Αυτές είναι οι βασικές προϋποθέσεις που θέλαμε να έχει το σύστημά μας για να γίνει λειτουργικό. Εκτός όμως από αυτά τα κύρια χαρακτηριστικά θέλαμε να αποδώσουμε και άλλα που να το κάνουν να διαφέρει από τις υπάρχουσες μηχανές αναζήτησης.

Βασική διευκόλυνση που θέλαμε να παραχωρήσουμε προς τον χρήστη είναι η παρουσίαση του περιεχομένου μίας ιστοσελίδας βάσει ενός χρονικού ορίου. Για να γίνουμε πιο συγκεκριμένοι, αυτό που θέλαμε να πετύχουμε είναι να δώσουμε την δυνατότητα σε κάποιον να εξετάζει τα δεδομένα που θέλει βάσει κάποιου χρονικού διαστήματος. Σε πολλά blocks για παράδειγμα αυτό που παρατηρούμε είναι ότι συνεχώς ανανεώνεται το περιεχόμενο και πολλές φορές είναι δύσκολο για κάποιον να δει την πληροφορίας που ζητά για μία συγκεκριμένη μέρα. Για να λύσουμε αυτό το πρόβλημα λοιπόν υλοποιήσαμε μία δομή στην οποία δίνεται το χρονικό διάστημα-ημερομηνίες και επιστέφεται στον χρήστη το περιεχόμενο-α του site για αυτό το χρονικό εύρος για το οποίο το σύστημά μας έχει αντλήσει πληροφορία από την δεδομένη διεύθυνση.

4.1.2 Παράμετροι-Περιορισμοί

Οι λειτουργικότητες που επιλέξαμε να υλοποιήσουμε αφορούν κατά βάση διευκολύνσεις προς τον χρήστη. Σημαντική παράμετρος που θέσαμε γα να αυξήσουμε την αποδοτικότητα ήταν το ***DateWindow*** ένα χρονικό παράθυρο το οποίο και μπορεί να θέσει ο χρήστης κατά την δική του επιλογή. Σκοπός αυτής της παραμέτρου είναι να υποδείξει στον *Crawler* του συστήματός μας μετά από πόσο χρονικό διάστημα θα επισκεφτεί εκ νέου ένα site το οποίο έχει ήδη εξετάσει στο παρελθόν. Πχ αν εκχωρήσουμε την τιμή 2 τότε υποδεικνύουμε να μην επισκεπτόμαστε μία διεύθυνση αν δεν έχουν παρέλθει 2 ημέρες μετά από την τελευταία επίσκεψη. Κατά αυτόν τον τρόπος δίνουμε την ευελιξία στον σύστημα να μην επανεξετάζει συνεχώς τις ίδιες διευθύνεις αφού οι πιθανότητες να έχει αλλάξει το περιεχόμενο μίας σελίδας μέσα σε μία μέρα είναι μικρές, ιδιαίτερα για λιγότερο δημοφιλείς σελίδες.

Τέλος, να σημειώσουμε πως επιλέξαμε να κατασκευάσουμε ένα σύστημα το οποίο και θα διαπραγματεύεται όρους μονάχα της αγγλικής γλώσσας. Ο λόγος που το κάναμε αυτό ήταν για δική μας διευκόλυνση καθώς τις διαδικασίες που θα περιγράψουμε παρακάτω έπρεπε να τις επαναλάβουμε τόσες φορές όσες και τις γλώσσες που θα θέλαμε να υποστηρίζουμε. Κάτι τέτοιο όμως είναι εκτός της σφαίρας των ενδιαφερόντων μας αφού τα ζητούμενο ήταν απλά να δούμε τι

πλεονεκτήματα θα είχαμε αν κατασκευάζαμε μία μηχανή βασισμένη στα χαρακτηριστικά του ***Hadoop***.

Επομένως βλέπουμε από τα παραπάνω οι απαιτήσεις που θέσαμε υπό το πρίσμα του χρήστη είναι ουσιαστικά τρεις:

- ***Vector Space Representation:*** Υλοποίηση του προκείμενου μοντέλου για να διαθέτει το σύστημα καλύτερη συνέπεια προς τα ερωτήματα του χρήστη.
- ***Χρονικό Παράθυρο:*** παράμετρος που καθορίζει πόσο συχνά θα επισκεπτόμαστε ένα site.
- ***Ερωτήματα Χρονικών Ορίων:*** Υποστηρίζονται queries για επιστροφή του περιεχομένου των ιστοσελίδων μέσα στα χρονικά πλαίσια που δίνει ο χρήστης.

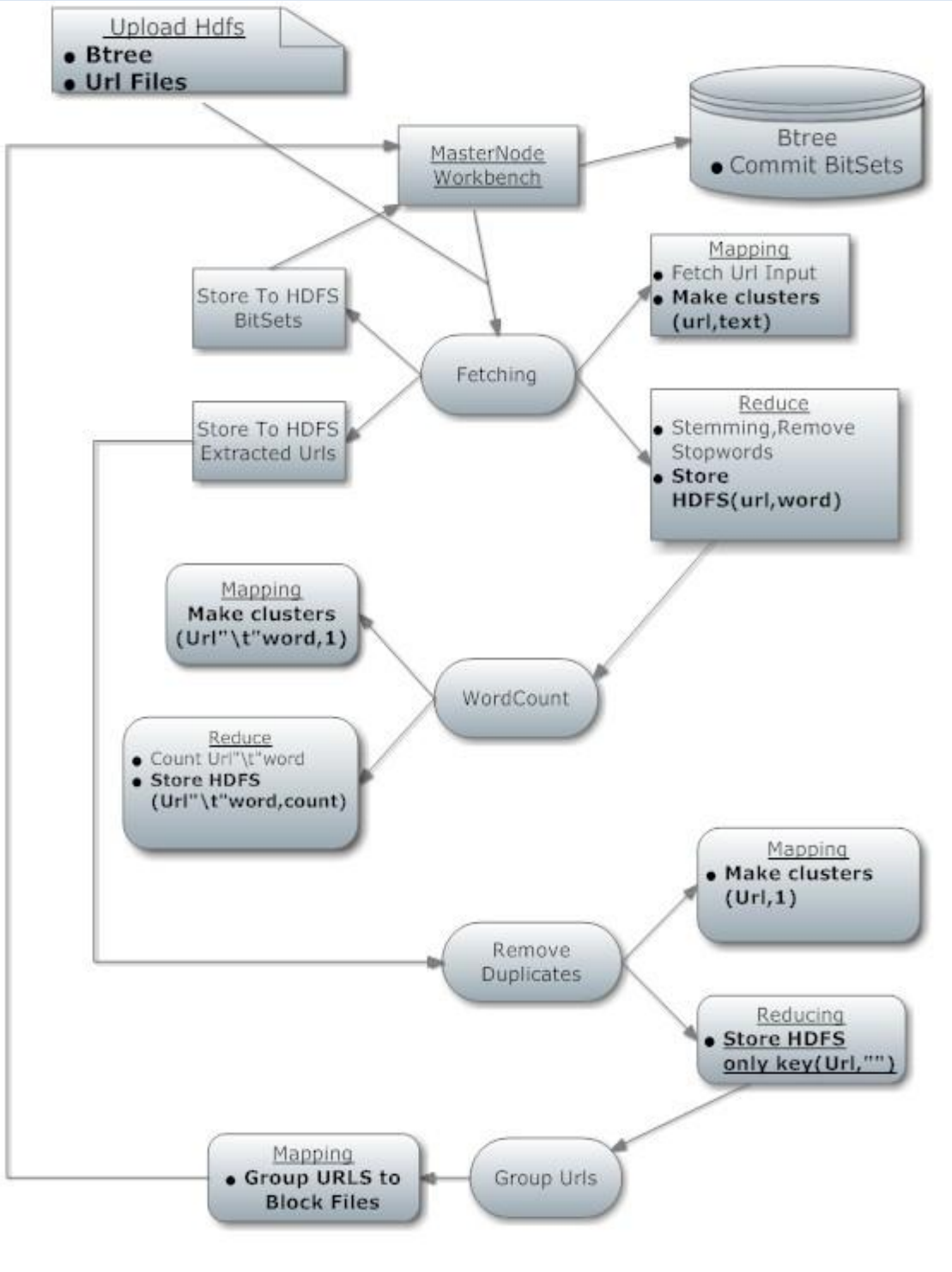
4.2 ΔΟΜΗ ΕΡΓΑΣΙΑΣ-ΠΕΡΙΓΡΑΦΗ ΕΠΙΜΕΡΟΥΣ ΤΜΗΜΑΤΩΝ

Στο εισαγωγικό τμήμα αναφέραμε πως κάθε μηχανή αναζήτησης ουσιαστικά απαρτίζεται από δυο βασικά τμήματα, τον *Crawler* και τα κομμάτι της ανάλυσης και επεξεργασίας των κειμένων. Για το καθένα από αυτά τα τμήματα τώρα χρειάστηκαν διάφορες τεχνικές και υλοποιήσεις ώστε να εξυπηρετηθούν διάφοροι σκοποί. Σε κάθε περίπτωση όμως αποφασίσαμε να χρησιμοποιήσουμε τα πλεονεκτήματα του Hadoop οπότε όπως θα δούμε και στη συνέχεια καθένα κομμάτι της υλοποίησης μας φροντίζαμε να προσαρμοστεί στην φόρμα του Hadoop.

Ξεκινώντας λοιπόν την ανάλυση από τον **Crawler**, οι οντότητες που χρειάστηκαν ήταν κατά βάση οι εξής:

Φάση 1η :

i. ***Έλεγχος && Εξαγωγή Διευθύνσεων - Επίσκεψη Προγραμματισμένων Διευθύνσεων -Τοπική Αποθήκευση -Προεπεξεργασία του εσωκλειόμενου κειμένου. (Fetching)***
i. ***Προγραμματισμός Εξαγόμενων Διευθύνσεων(Remove Duplicates & Group Urls)***
ii. ***Καταμέτρηση Εμφανίσεων Κάθε Όρου Ανά Site(WordCount)***

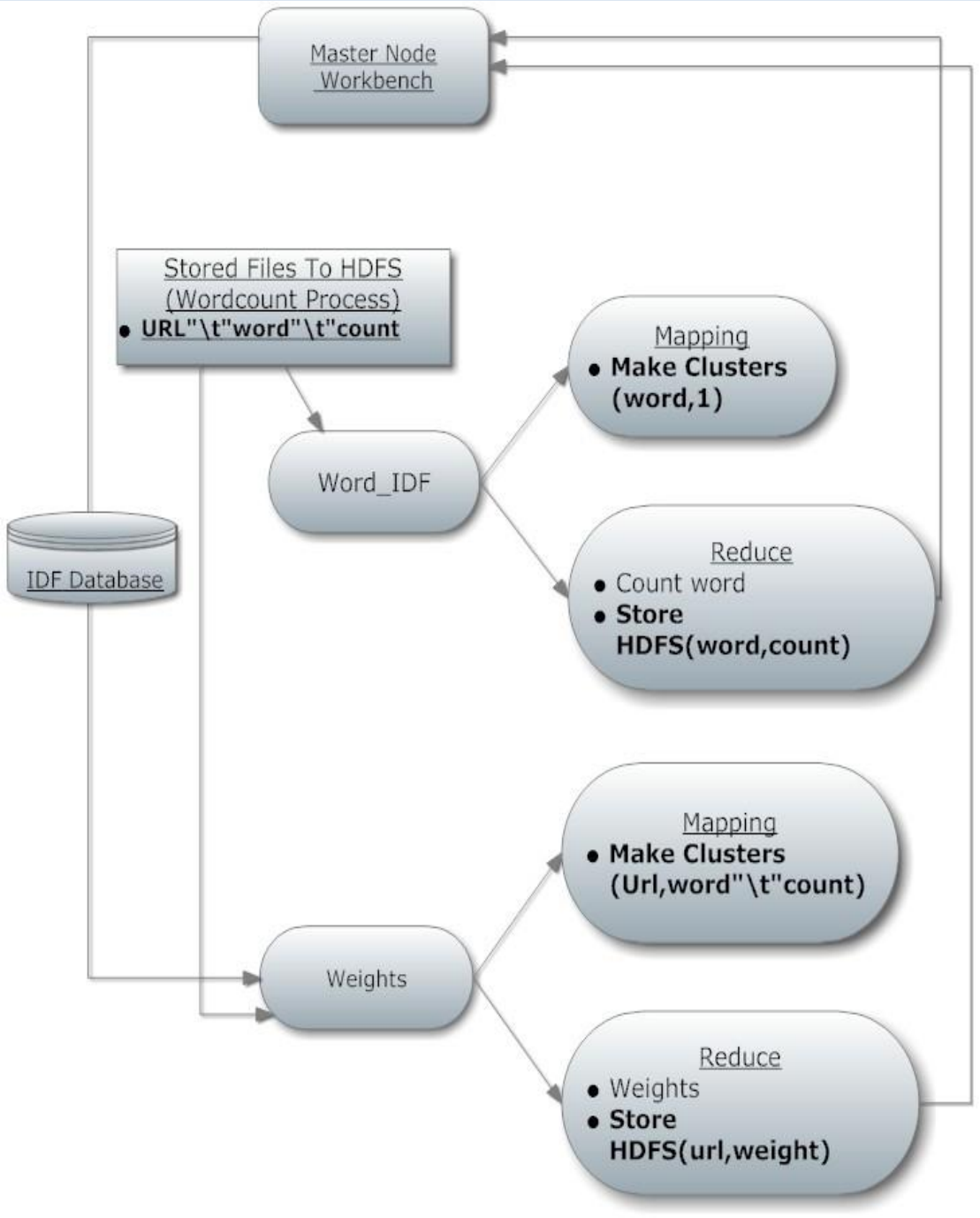


Εποπτική Παρουσίαση 1ης Φάσης: Γενική Παρουσίαση Map /Reduce Βημάτων Κάθε Διεργασίας

Όσο για το σώμα της ανάλυσης και επεξεργασίας των περιεχομένων ορίσαμε τα εξής :

Φάση 2η :

i. ***Κατασκευή Vector Space Representation (Word_IDF && Weights).***



Εποπτική Παρουσίαση 2ης Φάσης: Γενική Παρουσίαση Map /Reduce Βημάτων Κάθε Διεργασίας

Ας δούμε όμως καλύτερα τα καθένα κομμάτι ξεχωριστά και τις σχέσεις μεταξύ τους. Παρακάτω, σε καθένα κομμάτι ξεχωριστά θα αναλύσουμε πιο λεπτομερώς καθένα τμήμα.

ΦΑΣΗ Ι: Crawler-Εξαγωγή && Έλεγχος Διευθύνσεων - Επίσκεψη Προγραμματισμένων Διευθύνσεων-Τοπική Αποθήκευση

Οι διαδικασίες της πρώτης φάσης που εξετάζουμε πραγματοποιούνται σε αυτήν την βαθμίδα. Σε αυτό το σημείο γίνεται η σύνδεση με τις νέες διευθύνσεις, ο έλεγχος των νέων διευθύνσεων που επισκεπτόμαστε, η εξαγωγή νέων διευθύνσεων καθώς και η τοπική αποθήκευση του περιεχομένου των ιστοσελίδων που κατεβάζουμε .

Οι έλεγχοι που πραγματοποιούσαμε σε αυτό το βήμα γίνονταν με διάφορα κριτήρια. Σε γενικά πλαίσια αυτό που προσπαθήσαμε να κάνουμε ήταν να ελέγξουμε για την κάθε διεύθυνση αν την είχαμε επισκεφτεί στο παρελθόν ή όχι και αναλόγως αποφασίζαμε αν τελικά θα επισκεπτόμασταν τον εκάστοτε ιστοτόπο για να απομονώσουμε το περιεχόμενο του . Στην περίπτωση που οι έλεγχοι ήταν επιτυχείς για κάθε διεύθυνση τότε και δημιουργούσαμε την κατάλληλη σύνδεση και απομονώναμε το ζητούμενο περιεχόμενο.

Προφανώς επειδή μας ενδιαφέρει η ταχύτητα ενοποιήσαμε αυτό το κομμάτι σε κώδικα σύμφωνο με το Hadoop. Τα παραπάνω βήματα πραγματοποιούνται κατά την φάση του Mapping όπου και παίρνουμε από το πεδίο *values* το καθένα Url. Στο τέλος της διαδικασίας αφού πραγματοποιηθούν τα παραπάνω διοχετεύουμε στον *collector* τα πεδία (Url,text),ώστε κατά την φάση του Reduce να επιτελεστεί το Stemming των όρων και η αφαίρεση των StopWords.

Προεπεξεργασία Κειμένου

Η διαδικασία αυτή ξεκινά κατά το Reduce. Σε αυτό το σημείο σκοπός μας είναι να απομακρύνουμε περιττούς όρους καθώς και να κάνουμε Stemming των λέξεων. Για να πετύχουμε το ζητούμενο το μόνο που χρειαζόταν να κάνουμε ήταν να εξετάσουμε το clusters που είχαν δημιουργηθεί κατά το Mapping. Βάσει του εκάστοτε Url(key), επεξεργαζόμασταν το κείμενο μέσω του ορίσματος *value* στο οποίο και είχε αντιστοιχηθεί από το Mapping το εσωκλειόμενο text κάθε ιστοσελίδας. Στο τέλος της επεξεργασίας μεταφέραμε στο Hdfs σύστημα μέσω του collector τα πεδία (Url, word), για να τα χρησιμοποιήσουμε κατ' αυτή την μορφή σε επόμενο βήμα.

Ο λόγος που θέσαμε αυτή την αρχιτεκτονική είναι προφανής. Εφ' όσον θέλαμε να εκμεταλλευτούμε τις λειτουργικότητες που προσφέρει το Hadoop έπρεπε να

βρούμε ένα τρόπο να συνδυάσουμε αυτά που πρέπει να κάνουμε κατά τις φάσεις του Map/Reduce. Αφού λοιπόν κατά το map γινόταν η προσέγγιση των ιστοσελίδων και η αποθήκευσή τους, κατά το Reduce μπορούσαμε πολύ εύκολα να επεξεργαστούμε το περιεχόμενο. Έτσι, το κείμενο που συλλέγαμε κάθε φορά από τις διευθύνσεις το διοχετεύαμε στους διαθέσιμους Reducers και εκεί ξεκινούσαμε την απομόνωση των όρων που θέλαμε να επεξεργαστούμε καθώς και το Stemming αυτών. Στο τέλος λοιπόν της διαδικασίας αυτής είχαμε κρατήσει όλους τους ζητούμενους όρους και στην μορφή που θέλαμε με σκοπό στα επόμενα επίπεδα να ξεκινήσουμε το χτίσιμο του Vector Space Model.

Mapping	key	value
Input	Hashcode(Url)	Url
Collector	Url	Text

Reducing	key	value
Input	Url	Text
Collector	Url	word

ΦΑΣΗ Ι: Προγραμματισμός Νέων Διευθύνσεων

Η συγκεκριμένη διαδικασία πραγματοποιείται σε δύο βήματα: Πρώτο είναι η αφαίρεση των διπλοτύπων από το σύνολο των νέων διευθύνσεων που έχουμε εξάγει και δεύτερο βήμα είναι η ομαδοποίηση των νέων διευθύνσεων που έχουμε βρει σε blocks αρχείων(κάθε αρχείο περιέχει συγκεκριμένο αριθμό διευθύνσεων). Και στις δύο περιπτώσεις υλοποιήσαμε τα συγκεκριμένα βήματα βάση του μοτίβου Map/Reduce που προσφέρει το Hadoop.

Remove Duplicates-Mapper

Σε αυτή την φάση ουσιαστικά ομαδοποιούμε τα δεδομένα που παίρνουμε από κάθε Mapper –Urls - και απλά τα διοχετεύουμε στον Collector με κλειδί το Url. Κατά αυτόν τον τρόπο φτιάχνουμε ομάδες με στοιχείο το κάθε Url επομένως οι διευθύνσεις που έχουν πάνω από μία εμφανίσεις θα δημιουργήσουν clusters με πάνω από ένα στοιχεία. Αντιθέτως, διευθύνσεις που έχουν ακριβώς μία εμφάνιση σε όλο το σύνολο των αρχείων εισόδου τότε θα δημιουργήσουν ομάδες με ακριβώς ένα στοιχεία.

http://yahoo.com/ <-Key

[1 1] <-Values(2 εμφανίσεις του yahoo στο σύνολο των νέων URL)

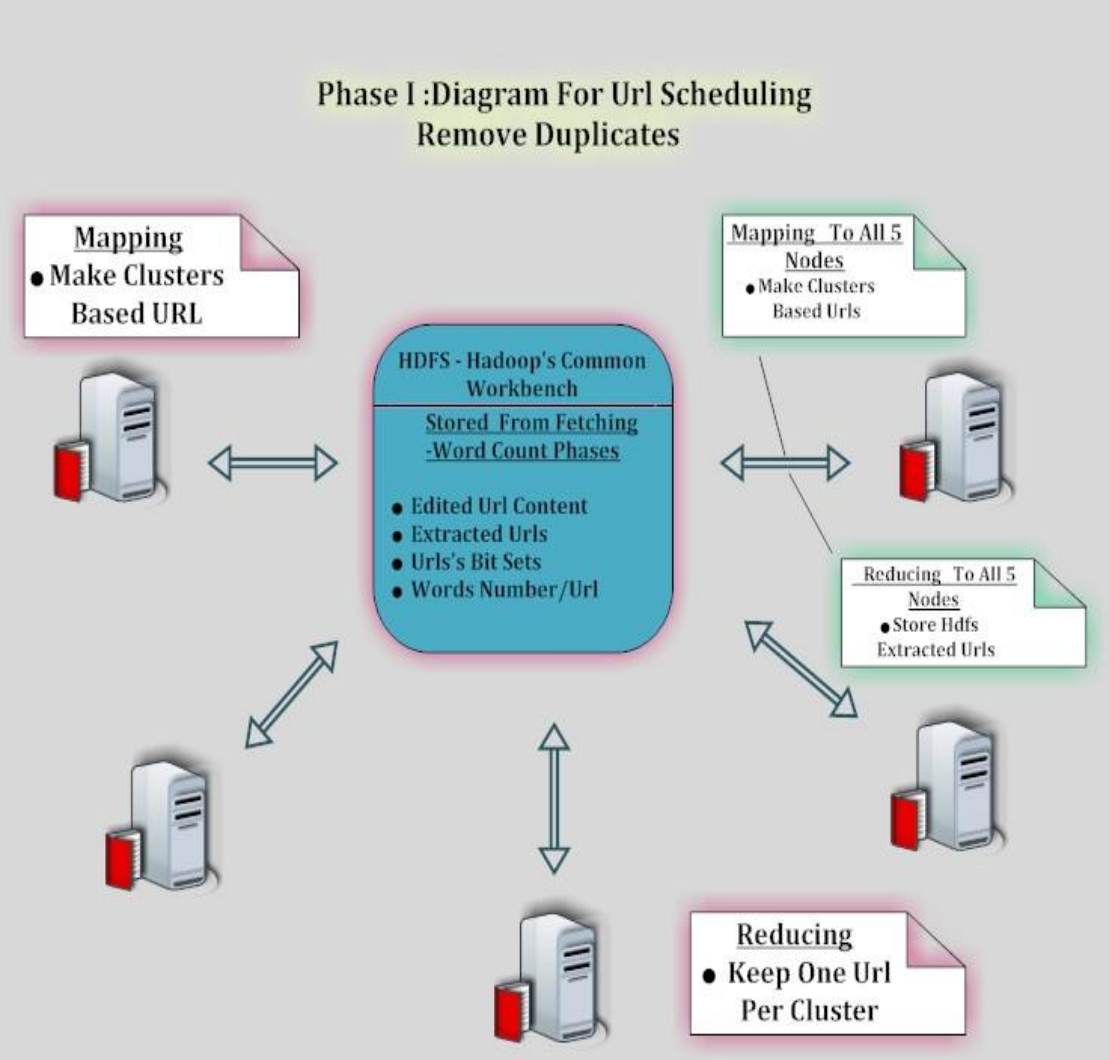
Cluster Που Θα Διοχετευτεί Στον Reducer

Remove Duplicates-Reducer

Τα δεδομένα από την φάση του Mapping μεταφέρονται στον Reducer επομένως σε αυτό το τμήμα πρέπει να διοχετεύσουμε στην έξοδο μονάχα ένα στοιχείο από τις ομάδες των Urls που έχουν σχηματιστεί. Για αυτόν τον λόγο λοιπόν το μόνο που απαιτείται να κάνουμε είναι απλά να περάσουμε στον collector ένα μόνο στοιχείο από κάθε cluster. Επομένως, βάσει αυτής της λογικής απομακρύνουμε όλες τις διπλοτυπίες και τα δεδομένα της εξόδου πλέον μπορούν να ομαδοποιηθούν σε αρχεία συγκεκριμένου αριθμού διευθύνσεων.

Mapping	key	value
Input	Hashcode(Url)	Url
Collector	Url	1

Reducing	key	value
Input	Url	1
Collector	Url	“”



Αφαίρεση Διπλοτύπων: 2ο Βήμα Προγραμματισμού Νέων Διευθύνσεων-Map/Reduce Διαδικασίες

Group_Urls-Mapper

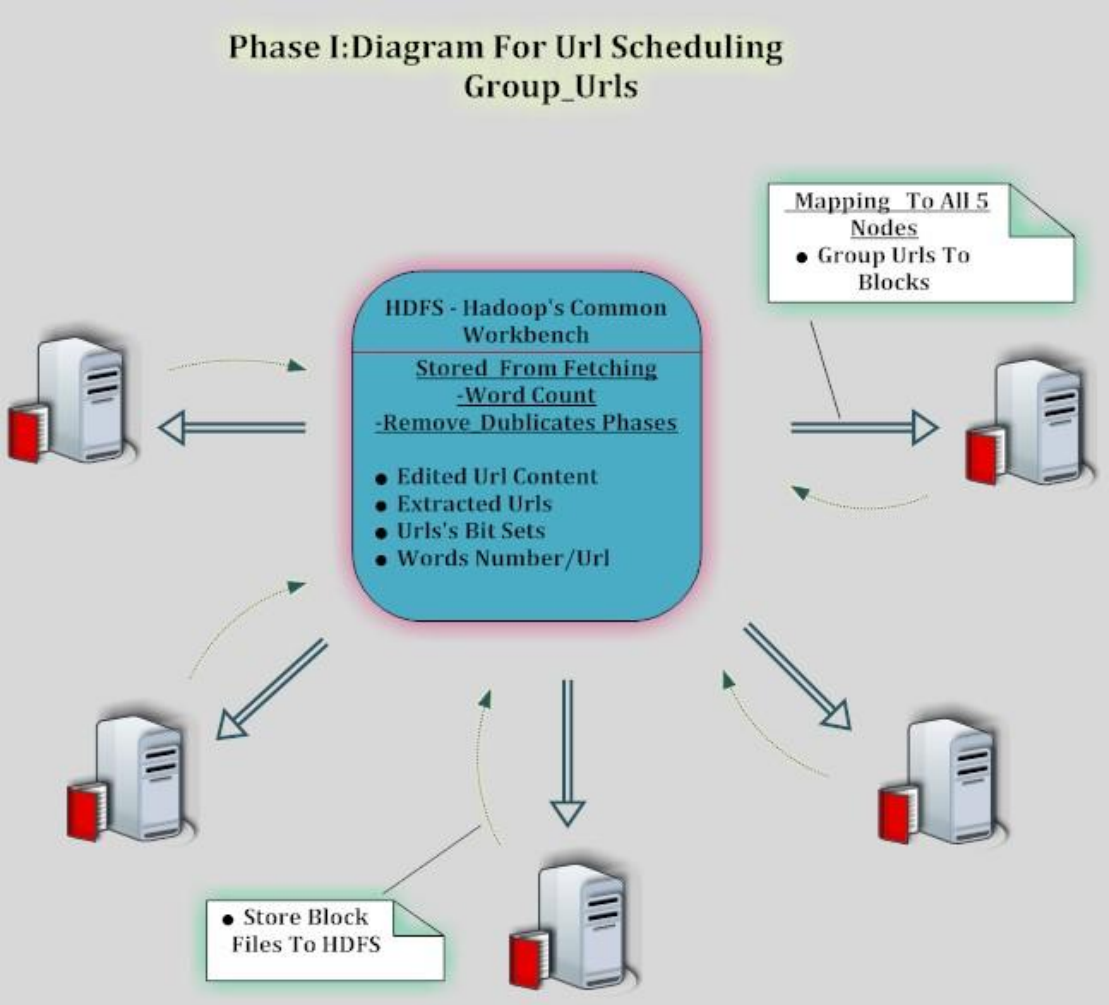
Από το προηγούμενο βήμα είδαμε πως κατατάξαμε τα δεδομένα μας στην επιθυμητή μορφή οπότε στο τελικό τμήμα απλά θα εξασφαλίσουμε να τα διαχωρίσουμε σε ομάδες αρχείων. Για την ακρίβεια αυτό που θέλαμε ήταν να φτιάξουμε blocks των 10.000 Urls ανά αρχείο ώστε σε κάθε φάση του fetching να προσεγγίζουμε περίπου τέτοιο αριθμό διευθύνσεων.

Η διαδικασία αυτή πραγματοποιήθηκε μονάχα σε μία φάση *Mapping* στην οποία και κάθε Mapper συγκέντρωνε τα Urls εισόδου σε μία λίστα. Όταν το μέγεθος της λίστας υπέρβαινε την μεταβλητή ***MaxUrlNum*** τότε ξεκινούσε και η μεταφορά των δεδομένων σε αρχείο που καταχωρούνταν στο Hdfs directory **//Future.** Στο τέλος αυτής της διαδικασίας αν το μέγεθος της ουράς ήταν μικρότερο από την τιμή ***MaxUrlNum*** τότε μεταφέραμε τα δεδομένα αυτά σε αρχείο. Στο τέλος λοιπόν της φάσης του Mapping είχαμε συγκεντρώσει τα δεδομένα εισόδου σε

αρχεία τα όποια και τελικά μεταφέραμε από το Hdfs//*Future* στο τοπικό directory //Future του Master Node clu26.

Ο λόγος που έγινε τώρα η μεταφορά των αρχείων αυτών τοπικά ήταν απλά για να μπορούμε μέσω του Master Node να ανεβάζουμε τα νέα αυτά αρχεία στην έναρξη της φάσης Fetching στο directory εισόδου **//In** οπότε και να επαναλαμβάνεται εκ νέου η διαδικασία προσέγγισης ιστοσελίδων.

Mapping	key	value
Input	Hashcode(Url)	Url
Collector	Url	“”



Ομαδοποίηση Διευθύνσεων: 3ο Βήμα Προγραμματισμού Νέων Διευθύνσεων-Map/Reduce Διαδικασίες

ΦΑΣΗ I: Καταμέτρηση Εμφανίσεων Κάθε Όρου Ανά Site

Σε αυτό το σημείο προσπαθούμε να αναπαραστήσουμε το περιεχόμενο που έχουμε συγκεντρώσει καταμετρώντας πόσες φορές εμφανίζεται κάθε όρος σε κάθε διεύθυνση από την οποία έχουμε συλλέξει το περιεχόμενο. Και σε αυτήν την περίπτωση χρησιμοποιήσαμε για αυτόν τον σκοπό το Hadoop οπότε και στο τέλος αυτής της διαδικασίας τα αρχεία που κατασκευάζουμε έχουν την μορφή URL\t Word\t num. Από εκεί και έπειτα τα δεδομένα μας έχουν την σωστή μορφή για να προχωρήσουμε στην επόμενη φάση της εργασίας μας, την κατασκευή της Vector Space αναπαράστασης.

Για να πραγματοποιηθεί αυτό κατά το Mapping παίρνουμε από το όρισμα values την κάθε γραμμή του κειμένου που έχει την μορφή *Url\t word* και δημιουργούμε clusters μέσω του collector της μορφής (*Url\t word,1).* Έτσι, στον Reducer απλά αθροίζουμε τις εμφανίσεις του κάθε key (*Url\t word,1)* και στον collector τελικά μεταφέρουμε τα δεδομένα ως (*Url\t word,sum(values)).*

Mapping	key	value
Input	Hashcode(*Url\t word*)	Url \t word
Collector	Url \t word	1

Reducing	key	value
Input	Url \t word	1
Collector	Url \t word	Sum(value)

ΦΑΣΗ II: Vector Space Representation

Φτάσαμε στο τελευταίο βήμα για την κατασκευή του συστήματος αναζήτησης, δηλαδή στην επεξεργασία των όρων ανά site με σκοπό την δημιουργία του προκείμενου μοντέλου. Για να κατασκευάσουμε το εν λόγω μοντέλο χρειάστηκε να υλοποιήσουμε πολλές διαφορετικές Hadoop Jobs ώστε να υπολογιστούν οι διάφοροι παράμετροι που χαρακτηρίζουν στο μοντέλο. Τις παραμέτρους αυτές τις αναλύσαμε στο αντίστοιχο κεφάλαιο, σελ18,αυτό που έχει ενδιαφέρον να δούμε αυτή τη στιγμή είναι να δούμε μέσα σε γενικά πλαίσια πώς κατανείμαμε την

δουλειά μας μέχρι να φτάσουμε στο τελικό στάδιο υποβολής των ερωτημάτων από τον χρήστη.

Πιο ειδικά, για να δώσουμε μορφή στο συγκεκριμένο αλγόριθμο έπρεπε κατά βάση να κατασκευάσουμε μία δομή που θα υποδείκνυε πόσες φορές και ποιοι όροι εμφανίζονται σε κάθε site. Σκεφτείτε λοιπόν τι υπολογιστικές απαιτήσεις θα χρειαζόταν ένα τέτοιο σύστημα που θα έπρεπε να δεικτοδοτήσει όλους τους όρους κάθε διεύθυνσης παγκοσμίως. Προφανώς, για να καλύψουμε τέτοιες απαιτήσεις θα έπρεπε το δεδομένα που θα δεικτοδοτούσαμε να τα αποθηκεύουμε για να μην υπάρχει κίνδυνος απώλειας.

Για να αναπαραστήσουμε τώρα το περιεχόμενο αρκεί να βρούμε για κάθε όρο την IDF παράμετρο και συνολικά για κάθε site/κείμενο το βάρος, βλ σελ 19. Επομένως, αυτό που κάναμε ήταν να σπάσουμε την δεύτερη φάση της εργασίας μας σε δύο Hadoop Jobs υπολογίζοντας αντίστοιχα τις IDF παραμέτρους κάθε όρου και το συνολικό βάρος κάθε κειμένου.

Τα αποτελέσματα από αυτές τις δύο διαδικασίες δεικτοδοτούνται στο τέλος από μία δομή Btree ώστε κατά την τελευταία φάση που γίνεται ο υπολογισμός ομοιότητας κάθε κειμένου με ένα ερώτημα του χρήστη να μπορούμε εύκολα να ανακτήσουμε τα δεδομένα που μας ενδιαφέρουν(Weight,IDF).

Παρακάτω, φαίνεται σε καθένα βήμα η λογική κατά το Mapping/Reducing που χρησιμοποιήθηκε για τους υπολογισμούς των IDF,Weighs. Στις αντίστοιχες παραγράφους παρακάτω θα αναλυθούν οι λόγοι και ο τρόπος υπολογισμού των παραμέτρων αυτών.

Mapping (IDF)	**key**	**value**
Input Collector	**Hashcode(*Url\t word\t count*)**	**Url\t word\t count**
	word	**1**

Reducing(IDF)	**key**	**value**
Input Collector	**word**	**1**
	word	**Sum(value)**

Mapping (Weights)	**key**	**value**
Input	**Hashcode(*Url\t word\t count*)**	**Url\t word\t count**

Collector	Url	word\t count

Reducing(Weights)	key	value
Input Collector	Url	word\t count
	Url	weight

4.3 ΑΝΑΛΥΣΗ ΕΠΙΜΕΡΟΥΣ ΤΜΗΜΑΤΩΝ

4.3.1 ΤΙ ΘΑ ΔΟΥΜΕ

Όπως αναλύσαμε και στο εισαγωγικό μέρος ,η εργασία μας αναλύεται ουσιαστικά σε δύο βασικά τμήματα. Στο κομμάτι που ελέγχουμε και αποθηκεύουμε τα νέα Urls και στο τμήμα επεξεργασίας των ιστοσελίδων που επισκεφθήκαμε για ανάλυση και δεικτοδότηση των περιεχομένων τους . Για να δημιουργήσουμε τα παραπάνω τμήματα επιμερίσαμε το καθένα σε διάφορα components –κλάσεις και χρησιμοποιήσαμε με διάφορες τεχνικές και αλγόριθμους που θα εξηγήσουμε παρακάτω .

Το κύριο χαρακτηριστικό πάνω και το οποίο χτίστηκε η εργασία είναι πως για να υλοποιήσουμε το συγκεκριμένο σύστημα αναζήτησης χρησιμοποιήσαμε σχεδόν σε όλα τα τμήματα το Hadoop βάσει και του όποιου προσαρμόσαμε τους αλγορίθμους και τα components που θα δούμε. Στο τμήμα αυτό που δεν επιλέξαμε να εντάξουμε το Hadoop ήταν ο έλεγχος των νέων Urls που εξάγαμε από κάθε ιστοσελίδα. Αυτό το σημείο υλοποιήθηκε σε κλασικό διαδικασιακό προγραμματισμό και τους λόγους θα τους αναφέρουμε στην πορεία.

Ο βασικός διαχωρισμός που πρέπει ωστόσο να τονίσουμε, είναι ότι η εργασία μας δομήθηκε με τέτοιο τρόπο ώστε κάθε κομμάτι να λειτουργεί ανεξάρτητα από κάποιο άλλο ώστε να παρέχεται η δυνατότητα να «τρέχουμε» το καθένα την ίδια χρονική στιγμή σε διαφορετικά συστήματα. Αυτό, και σε συνδυασμό με την παράλληλη επεξεργασία που προσφέρει το Hadoop, έκανε την διαχείριση ενός μεγάλου όγκου πληροφορίας αρκετά πιο γρήγορη και ευέλικτη καθώς ο έλεγχος των λαθών περιοριζόταν μονάχα στο τμήμα εκείνο που «έτρεχε» σε κάθε σύστημα και όχι στο σύνολο της εργασίας.

4.4 ΣΥΣΤΗΜΑ ΔΕΙΚΤΟΔΟΤΗΣΗΣ URL-BTREE

Για ένα σύστημα αναζήτησης που εξετάζει ένα τόσο μεγάλο αριθμό ιστοσελίδων σε παγκόσμια κλίμακα καθίσταται λογικό πως υπάρχει η ανάγκη να γνωρίζουμε ανά πάσα στιγμή ποία Urls έχουμε επισκεφθεί. Στην περίπτωσή μας όμως εφόσον επιλέξαμε να υποστηρίζουμε και άλλες λειτουργικότητες, όπως την διατήρηση timestamps που υποδηλώνουν πότε άλλαξε το περιεχόμενο μιας συγκεκριμένης ιστοσελίδας από την τελευταία φορά που την επισκεφθήκαμε , οι λόγοι έγιναν περισσότεροι. Για να επιτύχουμε λοιπόν αυτό έπρεπε να βρούμε έναν αλγόριθμο δεικτοδότησης που να καλύπτει αυτές τις απαιτήσεις αλλά συνάμα να είναι αρκετά γρήγορος στην αναζήτηση κάποιας ιστοσελίδες-Url. Η απάντηση σε αυτές τις απαιτήσεις δόθηκε από τον πλέον διαδεδομένο και γρήγορο αλγόριθμο δεικτοδότησης **B-Tree**.

Όπως αναλύσαμε και στο αντίστοιχο κεφάλαιο του αλγορίθμου, το B-tree μας παρέχει ανάμεσα από ένα μεγάλο πλήθος δεδομένων γρήγορη πρόσβαση σε αυτά ,στοιχείο που ήταν καθοριστικό για την επιλογή του στην περίπτωσή μας . Ειδικότερα, στην εργασία μας επιλέξαμε να κρατάμε ιστορικό version των ιστοσελίδων με βάσει την παράμετρο του χρονικού παραθύρου. Πχ αν είχαμε επισκεφτεί το www.yahoo.com στις 9/9/2009 και το χρονικό μας παράθυρο ήταν 2 μέρες αποκλείαμε από την λίστα των μελλοντικών επισκέψεων το προκείμενο Url μέχρι και τις 10/9/2009. Αντίθετα, αν μετά από 10/9/2009 συναντούσαμε σε κάποιο άλλο site το www.yahoo.com το προσθέταμε στην λίστα των διευθύνσεων των sites και μόνο αν το περιεχόμενο του(κείμενο) είχε αλλάξει δεχόμασταν εκ νέου να το αποθηκεύσουμε τοπικά.

Από τις παραπάνω παρατηρήσεις βλέπουμε πως ήταν αναγκαία η χρήση κάποιας μορφής δεικτοδότησης των ιστοσελίδων ,των ημερομηνιών αλλά και του περιεχομένου των ιστοσελίδων τις οποίες είχαμε επισκεφτεί. Πώς αλλιώς θα μπορούσαμε να ελέγξουμε αν κάποιο Url που συναντούσαμε το έχουμε ήδη εξετάσει ή όχι, και αν ναι ,αν είχε αλλάξει κάτι από τη τελευταία φόρα; Και πάνω σε όλα αυτά έπρεπε να αναλογιστούμε αυτή η διαδικασία έλεγχου αν έχουμε «δει» ή όχι κάποιο site , πως έπρεπε να γίνει σε αποδοτικό χρόνο δεδομένου του μεγάλου αριθμού ιστοσελίδων που υπάρχει παγκοσμίως. Όλα αυτά, μας οδήγησαν να επιλέξουμε την δεικτοδότηση μέσω της ιεραρχίας του B-tree με κλειδί το Url και με value μία δομή με διάφορα στοιχεία που θα επεξηγήσουμε παρακάτω.

Πριν ξεκινήσουμε την περιγραφή να πούμε πως δεν υλοποιήσαμε τον αλγόριθμο B-tree άλλα χρησιμοποίησαν από το project *Jdbm*[3] την υλοποίηση του προκειμένου αλγορίθμου. Πρόκειται για ένα open source project το οποίο ανάμεσα στ' άλλα παρέχει τη δυνατότητα να αποθηκεύουμε σε βάση τα δεδομένα του B-tree.

[3] http://jdbm.sourceforge.net/

Αυτό μας διευκόλυνε αρκετά αφού, εκτός από το πλεονέκτημα της αποθήκευσης δεδομένων που μας ενδιαφέρουν, μέσω της βάσης μπορούσαμε να ελέγξουμε τις ιστοσελίδες που είχαμε επισκεφθεί στο παρελθόν ακόμα κι αν αρχικοποιούσαμε εκ νέου το project μας.

4.4.1 Πεδία Δεικτοδότησης

Σε αυτή την ενότητα θα επεξηγήσουμε με τον τρόπο με τον οποίο σχεδιάσαμε τη δεικτοδότηση, τί τιμές και δομές πήραν τα πεδία (key,value) του B-tree. Τα χαρακτηριστικά αυτά που θα δούμε αντιστοιχούν στην εργασία μας στην κλάση Indexing.java που όπως καταλαβαίνουμε έχει ανάμεσα στις άλλες καθολικές μεταβλητές ένα δείκτη σε B-tree δομή . Μέσω αυτής φροντίζουμε να κάνουμε τη διαχείριση των Urls και των δεδομένων που θέλουμε για δεικτοδότηση.

Όπως έχουμε ήδη πει, το ζητούμενο για εμάς ήταν να μπορούμε να ελέγχουμε ποίες σελίδες έχουμε επισκεφτεί . Για αυτό τον λόγο έπρεπε να κάνουμε την αναζήτηση στα δεδομένα του δέντρου με key το Url. Αυτό όμως που δεν είναι εμφανές, είναι πως το μέγεθος των τιμών των Urls μπορούσε να γίνει αρκετά μεγάλο. Πχ για ένα url βλέπουμε πως ο χώρος που θα δεσμευόταν για την αποθήκευση του θα ήταν απαγορευτικά μεγάλος. Αυτό με την σειρά του θα έκανε την αναζήτηση ανάμεσα σε όλο το πλήθος των keys πιο αργό αφού θα έπρεπε να γίνονται συγκρίσεις ανάμεσα σε Strings . Όλα αυτά και σε συνδυασμό με τον μεγάλο αριθμό δεδομένων που θέλαμε να δεικτοδοτήσουμε θα έκανε την δουλεία μας πολύ πιο δύσκολη και λιγότερο αποδοτική. Για αυτό τον λόγο επιλέξαμε να μην αποθηκεύουμε keys σε μορφή Strings αλλά σε μορφές Signatures-BitSet βάσει του αλγόριθμου MD5. Έτσι , ο χώρος που δεσμευόταν για τα keys ήταν πιο μικρός (128 bits για κάθε key) και ταυτόχρονα η αναζήτηση ανάμεσα σε όλα τα Urls γινόταν πιο γρήγορα.

Επόμενο σημείο που πρέπει να θίξουμε είναι τι δομή επιλέξαμε να χρησιμοποιούμε για το πεδίο *value* toy B-tree. Ενδεικτικά να πούμε πως σε αυτό το πεδίο χρησιμοποιήσαμε ένα πίνακα 3 θέσεων σε κάθε θέση του οποίου αποθηκεύαμε μια μεταβλητή-δομή . Για να καταλάβουμε ποιες τιμές εισήγαμε σε κάθε πεδίο ας υπενθυμίσουμε πρώτα τί λειτουργικότητα θέλαμε να υποστηρίξουμε . Είχαμε πει ο σκοπός μας ήταν για κάθε σελίδα να ελέγχουμε αν έχει αλλάξει το περιεχόμενο της και αν ναι να αποθηκεύουμε ένα timestamp για την συγκεκριμένη έκδοση της σελίδας. Από αυτό λοιπόν γίνεται κατανοητό πως πρέπει κάπως να αποθηκεύουμε το περιεχόμενο της διεύθυνσης και αν αυτό έχει αλλάζει να το αντικαθιστούμε με το νέο. Θα ήταν όμως αποδοτικό να καταχωρούμε όλο το περιεχόμενο μιας σελίδας σε μια μεταβλητή ;

Θα ήταν αφελές αν ακολουθούσαμε μία τέτοια τακτική αφού για κάθε σελίδα μόνο το κείμενο που εμπεριέχει στις πιο πολλές περιπτώσεις είναι της τάξης

των Kbytes. Για να αποφύγουμε λοιπόν αυτό το πρόβλημα επιλέξαμε να κρατάμε στην πρώτη θέση του πίνακα το MD5 Signature της κάθε σελίδας το οποίο και αναγόταν σε 128 bits. ΄Έτσι, συγκρίνοντας κάθε φορά τις υπογραφές μιας παλαιότερης και νεότερης έκδοσης μιας σελίδας μπορούσαμε να δούμε αν υπάρχει αλλαγή στο περιεχόμενο. Αν όντως διαπιστώναμε αλλαγή τότε αντικαθιστούσαμε την πρώτη θέση του πίνακα με το νέο Signature

Στην δεύτερη θέση του πίνακα καταχωρούμε το timestamp της τρέχουσας σελίδας που εξετάζαμε. Πχ αν για πρώτη φορά καταχωρούσαμε στον B-tree το Url www.yahoo.com τότε στην δεύτερη θέση θα εισάγουμε ένα timestamp για την συγκεκριμένη χρονική στιγμή που εξετάζαμε αυτή την σελίδα. Αν στο μέλλον διαπιστώναμε αλλαγή στην σελίδα αυτή και έπρεπε να αντικαταστήσουμε το Signature της σελίδας στο πρώτο κελί του πίνακα τότε θα αντικαθιστούσαμε το υπάρχον timestamp με ένα νέο που θα αντιπροσώπευε αύτη την χρονική στιγμή πού εξετάσαμε εκ νέου αυτό το Url.

Τέλος, στην τρίτη θέση του πίνακα επιλέξαμε να κρατάμε όλα τα timestamps που αφορούν τα versions για μια συγκεκριμένη διεύθυνση σε ένα Vector. Αυτό το κάναμε γιατί έπρεπε με κάποιο τρόπο να αποθηκεύουμε αυτή την πληροφορία και το πότε άλλαξε μια σελίδα ώστε να απαντάμε σε ερωτήματα του τύπου «δώστε μου το περιεχόμενο-α ενός Url με βάση μια ημερομηνία-ες». Έτσι κάθε φορά που εντοπίζαμε μια καινούργια έκδοση ενός Url εισάγαμε μέσα στο Vector το παλαιότερο timestamp που είχαμε ήδη καταχωρημένο στη δεύτερη θέση του πίνακα ,και όπως προείπαμε βάζαμε στη θέση του το νεότερο που αφορούσε την τρέχουσα σελίδα .

Με βάση αυτόν τον σχεδιασμό μπορέσαμε να διατηρήσουμε τα δεδομένα που μας ήταν απαραίτητα για να κάνουμε του ελέγχους που θα δούμε παρακάτω. Ταυτόχρονα, χρησιμοποιώντας την δυνατότητα να αποθηκεύουμε τοπικά στην βάση μας αυτές τις πληροφορίες αποφύγαμε τον κίνδυνο απώλειας αυτών των διευθύνσεων που είχαμε ήδη εξετάσει. Έτσι , το σύστημα μας μπορούσε να ανανήψει από πιθανά σφάλματα και κάθε φορά να μην επαναλαμβάνει ίδιες διαδικασίες.

4.4.2 B-Tree & HDFS Σύστημα

Μέχρι στιγμής έχουμε επεξηγήσει το σκεπτικό και τους λόγους που μας οδήγησαν να δεικτοδοτήσουμε τα δεδομένα μας μέσω της εν λόγω δομής. Από τα παραπάνω λοιπόν σχηματίζεται η εικόνα πως για να κάνουμε τους απαιτούμενους ελέγχους απλά χρειάζεται να φορτώνουμε τα στοιχεία που έχουμε καταχωρήσει στην μνήμη και από εκεί και έπειτα οι έλεγχοι θα γίνονται καθολικά και γρήγορα για όλα τις νέες διευθύνσεις . Επίσης, τα νέα δεδομένα που εισάγονται αφήνεται

να εννοηθεί πώς γίνονται *commit* ώστε να μελλοντικά να μπορούμε να φορτώσουμε και τα νέα στοιχεία που συλλέξαμε.

Όλα αυτά θα συνέβαιναν και την πράξη αν δεν είχαμε τον παράγοντα του Hdfs συστήματος του Hadoop όπου και ανεβάζουμε το ***Ip_Btree*** για να μπορούν όλοι οι κόμβοι να αρχικοποιούν ένα κοινό Btree. Οι περιορισμοί τώρα που μας θέτει το HDFS σύστημα είναι πώς το Hadoop έχει δικό του API- **hadoop.FileSystem** - για την διαχείριση των αρχείων που βρίσκονται αποθηκευμένα σε Hdfs Directories οπότε ο κώδικας που υλοποιεί τις διαδικασίες μεταφοράς και εγγραφής των δεδομένων από και προς την βάση δεν είναι συμβατός με τις αντίστοιχες βιβλιοθήκες της java. Επιπρόσθετα, στην έκδοση του **Hadoop 0.19.1** που χρησιμοποιούμε δεν έχει υλοποιηθεί ακόμα η βιβλιοθήκη για την τυχαία εγγραφή πάνω σε αρχεία –αντίστοιχη κλάση ***java.ioRandomAccessFile*** στην *Java* που χρησιμοποιείται στην υλοποίηση του **JDBM project**- .Επομένως ,αυτοί οι περιορισμοί μας δημιούργησαν αρκετά προβλήματα και πράγματα που θα μπορούσαν να γίνουν πιο εύκολα έπρεπε να βρούμε ένα τρόπο να τα υλοποιήσουμε βάσει του API του *Hadoop.*

Κατ' αρχάς να ξεκαθαρίσουμε για ποιο λόγο έπρεπε να μεταφέρουμε το Btree στο Hdfs σύστημα αφού κάποιος θα μπορούσε να ισχυριστεί πως θα ήταν πιο εύκολο να μην αρχικοποιούμε το Btree πάνω στο Hdfs αλλά τοπικά σε κάθε Node του Cluster. Πράγματι, μία τέτοια λύση θα μας διευκόλυνε πάρα πολύ αφού οι εισαγωγές και τα commits θα γίνονταν τοπικά σε κάθε κόμβο όπου υποστηρίζονται τα κλασικά API της Java για τον χειρισμών αρχείων. Αυτό όμως που δεν διαφαίνεται ως πρόβλημα είναι πως με αυτή την επιλογή μετά το πέρας της διαδικασίας του Fetching ενός Block διευθύνσεων θα είχαμε πέντε διαφορετικά Btree ,καθένα στο τοπικό directory του κάθε κόμβου.

Το κάθε Btree ξεχωριστά τώρα θα περιείχε κοινές διευθύνσεις και με τις άλλες δομές αλλά σίγουρα θα περιείχε και πολλές διαφορετικές αφού καθένας Mapper έχει ως σημείο αναφοράς το Btree στον κόμβο που ανήκει. Έτσι πχ αν ένας mapper του Node Clu26 βρει ως νέο Url ***www.amazon.com*** και το ίδιο βρει ένας άλλος Mapper του Clu25 τότε συνολικά θα είχαμε καταχωρήσει 2 φορές το ίδιο link. Επομένως ,για να μην έχουμε τέτοιου είδους ασυνέπειες έπρεπε να ανεβάσουμε το δέντρο μας σε ένα κοινό directory για όλους τους κόμβους ,στο **//Btree** του Hdfs.

Αφού λοιπόν καταλήξαμε για το ποιά τακτική θα ακολουθήσουμε ας δούμε και πώς επιλύσαμε τα προβλήματα για την ενημέρωση της . Το πρώτο πρόβλημα που έπρεπε να αντιμετωπίσουμε ήταν πώς θα κάνουμε loading το Btree σε κάθε Mapper για να βλέπει ποία Links είναι ήδη καταχωρημένα. Έπρεπε λοιπόν , να απομονώσουμε τα τμήματα του κώδικα του **Jdbm** που επιτελούσαν την διαδικασία αυτή και στα σημεία που γινόταν χρήση των κλασικών API να τα αντικαταστήσουμε με κώδικα αντίστοιχο του File System του Hadoop. Αυτή η τροπο-

ποίηση ήταν εφικτή καθώς επιτρέπεται η ανάγνωση αρχείων και στο HDFS με τρόπο παρόμοιο όπως καις την Java.

Αυτό που δεν γινόταν όμως να τροποποιήσουμε ήταν το τμήμα του κώδικα για το commits της βάσης γιατί χρησιμοποιούσε την RandomAccessFile που κάτι αντίστοιχο δεν υποστηρίζεται ακόμα από το Hadoop. Επομένως, έπρεπε να βρούμε ένα άλλο τρόπο για να κάνουμε συνεπές το σύστημά μας. Η λύση δόθηκε σκεπτόμενοι πώς το commit μπορεί να γίνει μόνο σε τοπικό directory οπότε προφανώς έπρεπε τα νέα δεδομένα που συλλέγει κάθε mapper να τα καταχωρήσουμε σε αρχεία τα οποία και μετά το τέλος του Fetching θα μεταφερθούν στο τοπικό directory του Master για να ενημερώσουμε την βάση.

Με λίγα λόγια κατά τη διαδικασία του fetching των διευθύνσεων όταν κάποιος Mapper έπρεπε να εισάγει δεδομένα μέσα στο δέντρο ,απλά καταχωρούσαμε σε αρχεία τα στοιχεία αυτά(Url_Signature,Page_Signature, TimeStamp) και μετά το πέρας της φάσης αυτής τότε ενημερώναμε το Btree εισάγοντας τα στοιχεία που εξήγαγαν οι Mappers διαβάζοντας τα αρχεία. Κατά αυτόν τον τρόπο μπορούμε να κάνουμε commit αφού πλέον δουλεύουμε σε τοπικό directory ,άρα και το σύστημα δεικτοδότησης αποκτά συνέπεια. Έτσι , την επόμενη φορά που θα ξεκινήσει η διαδικασία του Fetching για την επίσκεψη νέων διευθύνσεων το Btree που θα ανεβαίνει στο Hdfs θα είναι κοινό για όλα τα Nodes και θα εμπεριέχει όλα τα δεδομένα που έχουμε συλλέξει μέχρι εκείνη την στιγμή.

4.5 ΥΠΟΓΡΑΦΕΣ-MD5 ΑΛΓΟΡΙΘΜΟΣ

Όμως έχουμε ήδη την ίδια λόγους αποδοτικότητας κατά την αναζήτηση στο Btree αλλά και για λόγους ορθολογικής αποθήκευσης των δεδομένων χρησιμοποιήσαμε τα Signatures τόσο των Urls όσο και του περιεχομένου των σελίδων .

Ο αλγόριθμος που χρησιμοποιήσαμε για αυτή την δουλειά είναι ο MD5 και αυτό γιατί προσφέρει αξιόπιστη κωδικοποίηση -128 bits- και σε λίγο χρόνο. Κατ' αυτόν τον τρόπο, μπορούμε να αναπαραστήσουμε 2^{128} διαφορετικές διευθύνσεις μειώνοντας τον κίνδυνο συγκρούσεων κατά την κωδικοποίηση διαφορικών Urls.Η κλάση που υλοποιεί αυτό τον αλγόριθμοο είναι η μέθοδος ***Signature.java*** και τα σημεία που πρέπει να επεξηγήσουμε είναι μονάχα η μέθοδος **MD5(String text)** .

Σε αυτή την μέθοδο αυτό που ουσιαστικά κάνουμε είναι να εκμεταλλευτούμε τις υπάρχουσες βιβλιοθήκες της Java και να δημιουργήσουμε ένα Instance του αλγόριθμου. Σε αυτό εκχωρούμε το **String text** που επιθυμούμε να μετατρέψουμε και υπολογίζεται το Signature. Εν συνεχεία, απλά μετατρέπουμε τα bytes από το αποτέλεσμα αυτής της διαδικασίας σε BitSet μέσω της μεθόδου **Convert_MD5_To_Bitset(byte bt[]) .**

4.6 URL STRUCTURE -URL_Entity

Απ' αυτά που έχουμε περιγράψει μέχρι στιγμής καταλαβαίνουμε πως κατά την επεξεργασία ενός Url χρειάστηκαν πολλά διαφορετικά χαρακτηριστικά όπως η υπογραφή του Url, η υπογραφή με το περιεχόμενο της ιστοσελίδας αλλά και το Vector με τα timestamps. Η μία λύση λοιπόν θα ήταν να μην καταχωρούμε αυτά τα δεδομένα σε κάποιο structure αλλά να τα υπολογίζουμε κάθε φορά όταν αυτό θα ζητηθεί. Ωστόσο , επειδή αυτά τα στοιχεία μας ήταν αναγκαία σε πολλά σημεία του κώδικα επιλέξαμε να τα συγκεντρώσουμε σε μία δομή βάσει και της όποιας στην συνέχεια κάναμε τις απαραίτητες ενέργειες . Έτσι , κάθε φορά που συναντούσαμε κάποια καινούρια διεύθυνση δημιουργούμε ένα αντικείμενο αυτής της κλάσης και αρχικοποιούσαμε τα πεδία της.

4.7 ΤΟΠΙΚΗ ΑΠΟΘΗΚΕΥΣΗ ΔΕΔΟΜΕΝΩΝ

Σε αυτή την ενότητα θα αναλύσουμε τον τρόπο αποθήκευσης των ιστοσελίδων που επισκεφτήκαμε αλλά και εργαλεία που χρησιμοποιήσαμε για την επεξεργασία του περιεχομένου αυτών. Οι διαδικασίες που θα περιγράψουμε υλοποιούνται στην κλάση ***TextRepository.java*** .

Αρχικά αυτό που πρέπει να διευκρινήσουμε είναι τι ακριβώς θέλαμε να αποθηκεύσουμε. Στο παγκόσμιο ιστό υπάρχουν πάρα πολλές μορφές πληροφορίας όπως απλό κείμενο, εικόνες , εκτελέσιμα αρχεία κλπ. Όπως είναι φυσικό εφόσον επιθυμούμε να κάνουμε ένα σύστημα αναζήτησης που στηρίζεται στην επεξεργασία κειμένων επιλέξαμε από όλες τις υπάρχουσες μορφές πληροφορίας να επεξεργαζόμαστε αυτές που εσωκλείουν κείμενο (αρχεία κειμένων- ιστοσελίδες). Για να γίνει αυτό έπρεπε να απομονώσουμε από τον κώδικα κάθε ιστοσελίδας το κείμενο το οποίο και μας ενδιέφερε. Βοήθεια σε αυτό το βήμα μας έδωσε η χρήση ενός parser [4]που σκοπό είχε να απομονώσει το ζητούμενο παίρνοντας ως είσοδο όλη την πληροφορία- κώδικα ιστοσελίδας.

Από την άλλη αυτό που επίσης θέλαμε είναι να αποθηκεύσουμε τοπικά το σύνολο των ιστοσελίδων που επισκεπτόμασταν ως μελλοντική πληροφορία που μπορεί να χρησιμοποιήσουμε για άλλους σκοπούς. Επιλέξαμε λοιπόν , να καταχωρούμε τοπικά όλο το περιεχόμενο των διευθύνσεων που εξετάζαμε στο directory του project μας ***Hadoop_Crawler/Texts/.*** Ο τρόπος τώρα με τον οποίο γινόταν η αποθήκευση ήταν σε μορφή αρχείων **.html** εφόσον απομονώνουμε και αρχεία ιστοσελίδων.

[4] Πηγή:http://www.rgagnon.com/javadetails/java-0424.html

Τέλος, κάτι άλλο που πρέπει να διευκρινίσουμε είναι η ονομασία που δίναμε στα αρχεία που αποθηκεύαμε. Πιο συγκεκριμένα, το όνομα κάθε ιστοσελίδας είναι συνάρτηση 2 παραμέτρων. Πρώτον ,της διεύθυνσης του Url και δεύτερον της ημερομηνίας κατά την οποίας το «κατεβάζουμε». Η πρώτη παράμετρος ωστόσο δεν είναι η διεύθυνση του Url αλλά το hash code(Md5signature(Url)). Γιατί κάτι τέτοιο; Πολύ απλά γιατί ένα Url μπορεί να έχει πολύ μεγάλο μήκος αλλά κυρίως γιατί μπορεί να εμπεριέχει απαγορευμένους χαρακτήρες που δεν θα επιτρέψουν την αποθήκευσή του.

Ομοίως και για την δεύτερη παράμετρο, χρησιμοποιήσαμε το hash code(timestamp(Url)). Ο λόγος που κάναμε κάτι τέτοιο είναι προφανής. Αφού δεχτήκαμε να εξετάζουμε πάνω από μία εκδόσεις μιας συγκεκριμένης ιστοσελίδας ανάλογα με την ημερομηνία και το περιεχόμενο της, έπρεπε να διαχωρίσουμε τις διαφορετικές ρέπλικες του Url-μία διεύθυνση μπορεί να την κατεβάζουμε πάνω από μία φορές σε διαφορετικές ημερομηνίες. Έτσι, εισήγαμε την παράμετρο του timestamp τροποποιημένη ως προς το hascode για να μην αναγκαζόμαστε να «προσκολλάμε» στην ονομασία του αρχείου ένα String μεγάλου μήκους που θα αντιπροσώπευε την ημερομηνία.

4.8 ΕΛΕΓΧΟΣ-ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΝΕΩΝ ΔΙΕΥΘΥΝΣΕΩΝ

Αφού είδαμε μέχρι στιγμής τα βασικά τμήματα συγκρότησης του Crawler ας αναφερθούμε στον τρόπο με τον οποίο ελέγχουμε και προγραμματίζουμε τις διευθύνσεις που θα επισκεφθούμε στο μέλλον. Η διαδικασία αυτή που θα περιγράψουμε υλοποιείται στην κλάση ***Scheduling.java***

Για να βρούμε κάθε φορά νέες διευθύνσεις πρέπει να απομονώνουμε από τον κώδικα της ιστοσελίδας τα εσωτερικά Urls. Η διαδικασία αυτή πραγματοποιείται κατά το Fetching των ιστοσελίδων όπου και απομονώνεται το περιεχόμενο. Σκοπός αυτής της ενότητας είναι να περιγράψει πως από το σύνολο των εξαγομένων διευθύνσεων από μία ιστοσελίδα ελέγχεται και προγραμματίζεται η σειρά με την οποία θα επισκεφτούμε αυτές τις νέες διευθύνσεις.

Υπάρχουν πολλοί αλγόριθμοι προγραμματισμού των εξαγομένων διευθύνσεων. Όπως περιγράψαμε και στο αντίστοιχο κεφάλαιο κάθε αλγόριθμος εξυπηρετούσε και άλλους σκοπούς. Στη δική μας περίπτωση επιλέξαμε να υλοποιήσουμε έναν απλό αλγόριθμο ***Breadth-first Scheduling*** τύπου ουράς FIFO κατά τον οποίο τα νέα δεδομένα εισάγονταν στο τέλος . Σε επόμενο βήμα, θα δούμε πως όταν έρθει η στιγμή να αφαιρέσουμε Urls για να επισκεφτούμε, θα ξεκινήσουμε από την κεφαλή της ουράς και θα προχωρήσουμε στα επόμενα στοιχεία. Έτσι, ο τρόπος με τον οποίο επιλεγούμε τις διευθύνσεις δεν βασίζεται σε κάποια ιδιαίτερα κριτήρια που να δίνουν σειρά προτεραιότητας ωστόσο αυτά που κερδίζου-

με είναι να προσεγγίζουμε σελίδες με πολύ καλή εσωτερική ποιότητα διευθύνσεων και ευκολία στην υλοποίηση και στον έλεγχο των εισαγόμενων στοιχείων.

4.8.1 ΑΝΑΛΥΣΗ ΣΧΕΔΙΑΣΜΟΥ

Για να επεξηγήσουμε καλύτερα την ακριβή δομή που χρησιμοποιήσαμε για αυτό το τμήμα πρέπει πρώτα να αναφέρουμε τί ακριβώς εργαλεία θέλαμε να χρησιμοποιήσουμε ,τα πλεονεκτήματα και μειονεκτήματα του σχεδιασμού .

Βασικός παράγοντας για την υλοποίηση που θα δούμε ήταν η χρήση του Hadoop βάσει του οποίου απομονώνουμε τοπικά τα υποψήφια Urls κατά τη φάση του Map και επεξεργαζόμαστε το κείμενο από τις διευθύνσεις κατά την φάση του Reduce . Δεδομένης βέβαια της λειτουργικότητας του Hadoop να δέχεται ωw είσοδο κείμενα, έπρεπε λοιπόν να δίνουμε το σύνολο των Urls που θέλουμε να κατεβάσουμε καταχωρημένα σε αρχεία.

Αυτός ήταν και ο βασικός άξονας πάνω στον οποίο κινηθήκαμε για να εκμεταλλευτούμε την ταχύτητα του Hadoop και συνάμα να απομονώσουμε τα εσωκλειόμενα links από κάθε διεύθυνση. Το ερώτημα σε αυτή την περίπτωση όμως είναι πώς ακριβώς καταφέραμε να συνδυάσουμε τον αλγόριθμο ***Breadth-first Scheduling*** δεδομένου πως το Hdfs σύστημα δέχεται μονάχα αρχεία ως εισόδο την στιγμή που τα υποψήφια Urls που εντοπίζουμε τοποθετούνται στην μνήμη σε μία ουρά-*Scheduling.class* .

Η εύκολη απάντηση θα ήταν απλά μεταφέροντας τα Urls της ουράς σε αρχείο το οποίο και στην συνέχεια θα μπορούσαμε να ανεβάσουμε στο Hdfs σύστημα για να τα επισκεφτούμε. Αυτό όμως που δεν διαφαίνεται ως πρόβλημα είναι το γεγονός πως με αυτό τον τρόπο δεν γίνεται να ελέγξουμε πόσα στοιχεία θα μεταφέρουμε στο μέλλον για να κάνουμε fetching.

Ας δούμε ένα μικρό παράδειγμα για να καταλάβουμε καλύτερα τι εννοούμε: Ας πούμε πως αρχικά δίνουμε 20.000 Urls προς επίσκεψη από τα οποία πολύ εύκολα μπορεί να βρεθούν 300.000 νέες διευθύνσεις. Τις διευθύνσεις αυτές τις έχουμε καταχωρημένες στην μνήμη οπότε μέχρι το τέλος του fetching των αρχικών διευθύνσεων πρέπει να έχουμε μεταφέρει με κάποιο τρόπο τα δεδομένα αυτά σε αρχείο. Αν επιλέγαμε να βάλουμε όλες τις νέες διευθύνσεις σε **ένα** αρχείο τότε κατά το επόμενο fetching των 300.000 θα εξάγονταν πολλές περισσότερες, οπότε και θα αυξανόταν εκθετικά κάθε φορά ο αριθμός των νέων δεδομένων.

Μία τέτοια τακτική όμως θα ήταν ιδιαίτερα επικίνδυνη για το σύστημα γιατί μία πιθανή αστοχία που θα συνέβαινε κατά το fetching μπορούσε να συντελέσει σε απώλεια όλων των δεδομένων που θα είχαμε συλλέξει μέχρι την στιγμή της αστοχίας . Επιπλέον ο χρόνος που θα απαιτούνταν κατά το Fetching μπορεί θα

αυξανόταν δραματικά δηλ. αν στα 50.000 χρειαζόμασταν 1,5 ώρες τότε στα 300.000 ,9. κλπ

Από τα παραπάνω λοιπόν διαφαίνεται πως για να αποφύγουμε τέτοιους σκοπέλους έπρεπε το σύνολο των νέων διευθύνσεων που εξάγαμε να το σπάσουμε σε αρχεία των blocks πχ των 50.000 Url ώστε στο μέλλον να μπορούμε να μεταφέρουμε κατά το fetching συγκεκριμένο αριθμό διευθύνσεων.

Οι κλάσεις που υλοποιούν αυτά τα βήματα είναι οι ***HadoopPackage.Remove_Dublicates, HadoopPackage.Group_Urls.***

Ας δούμε λοιπόν την λογική για τον προγραμματισμό των νέων διευθύνσεων πιο συγκεντρωτικά –την επεξήγηση του κώδικα της κάθε κλάσης θα τον δούμε σε επόμενες παραγράφους.

4.8.2 ΑΝΑΛΥΣΗ ΣΧΕΔΙΑΣΜΟΥ-ΠΕΡΙΓΡΑΦΗ ΥΛΟΠΟΙΗΣΗΣ

Η διαδικασία ξεκινά κατά το fetching των ιστοσελίδων μέσω της κλάσης *URL_Fetching.java* όπου και κατεβάζουμε το περιεχόμενο. Από εκεί θα απομονώσουμε τα εσωτερικά links και με την βοήθεια της *Scheduling.java* θα καταχωρίσουμε στην ουρά τα δεδομένα. Μόλις η ουρά ξεπεράσει ένα συγκεκριμένο αριθμό εισαχθέντων στοιχείων τότε ξεκινάμε την φάση της μεταφοράς των δεδομένων στο Hdfs directory **/Urls .**

Ο λόγος τώρα που ορίζουμε ένα κατώφλι στο σύνολο των δεδομένων που θα μεταφέρουμε στο Hdfs κατά αυτή την φάση είναι για να αποφύγουμε τα προβλήματα που προκαλούνται κατά την διαχείριση μεγάλων αρχείων **(πχ. Java.OutOfMemoryException)** αλλά και για λόγους ευστάθειας του Hdfs συστήματος. Επομένως, κάθε φορά που μια ουρά κάποιου Mapper θα έφτανε το όριο *Scheduling*.MaxServerListSize απλά δημιουργούσαμε ένα καινούριο αρχείο στο οποίο και καταχωρούσαμε τα νέα Urls.Κατά αυτόν τον τρόπο μπορέσαμε να συγκεντρώσουμε το σύνολο των νέων διευθύνσεων σε αρχεία πάνω στο Hdfs.

Από αυτά τα δεδομένα τώρα πρέπει να εξετάσουμε ποίες διευθύνεις είναι καταχωρημένες δύο φορές .Θυμηθείτε πώς καθένας Mapper λειτουργεί ανεξάρτητα από τους υπόλοιπους επομένως μπορεί ένας Mapper να «βλέπει» το ίδιο Btree με κάποιον άλλον –επιτελούνται σωστοί έλεγχοι - αλλά αν δύο βρουν ένα κοινό Url που δεν είναι ήδη καταχωρημένο στο Btree τότε θα το εισάγουν και ο δύο στην ουρά-*Scheduling.class***.** Έτσι, μέχρι το τέλος της διαδικασίας του Fetching θα έχουν καταχωρηθεί στα αρχεία με τις νέες διευθύνσεις διπλότυπα Url. Άρα για να αποφύγουμε τις επαναλήψεις επεξεργαζόμαστε τα αρχεία που έχουμε φτιάξει και τελικά αφαιρούμε τις πολλαπλές εμφανίσεις.

Στο τελευταίο βήμα αυτής της διαδικασίας απομένει να ομαδοποιήσουμε τα αρχεία που βρίσκονται στο Hdfs σύστημα και να τα μεταφέρουμε στο τοπικό di-

rectory **/NextUrls** του MasterNode. Από εκεί μετά θα μπορεί ο Master να ανεβάζει τα τελικά αρχεία ώστε να ξεκινήσει εκ νέου η διαδικασία του Fetching.

Τώρα για να επιτευχθεί κάτι τέτοιο η λογική που ακολουθείται είναι η ακόλουθη: Καθένας Mapper ουσιαστικά επεξεργάζεται τα Urls από τα αρχεία εισόδου και για καθένα Url του αρχείου που βρίσκει αυξάνει μια μεταβλητή κατά 1.Όταν η μεταβλητή αυτή φτάσει το όριο του block που επιθυμούμε τότε μεταφέρουμε στο τοπικό directory το τελικό αρχείο. Μετά το πέρας της διαδικασίας λοιπόν καθένας Mapper θα έχει ομαδοποιήσει τα στοιχεία στην τελική τους μορφή. Από εκεί και έπειτα μπορούμε επιλέξουμε πόσα αρχεία θα ανεβάσουμε στο Hdfs και με αυτόν τον τρόπο να καταμερίσουμε την δουλεία μας καλύτερα μειώνοντας τον κίνδυνο αστοχιών.

4.9 ΣΥΣΤΗΜΑ ΕΛΕΓΧΟΥ ΔΙΕΥΘΥΝΣΕΩΝ ΔΕΥΤΕΡΟΥ ΕΠΙΠΕΔΟΥ

Μέχρι στιγμής έχουμε εξετάσει τους πρώτους ελέγχους που πραγματοποιούμε πριν χαρακτηρίσουμε μία διεύθυνση κατάλληλη για καταχώρηση στην ουρά. Είπαμε λοιπόν, πώς κατά την πρώτη φάση ελέγχου(*Scheduling.java*) δεχόμαστε τα Urls που δεν είναι καταχωρημένα στο B-tree αλλά και αυτά που έχουμε εξετάσει και έχει παρέλθει κάποιο χρονικό διάστημα από την τελευταία φορά. Όμως δεν έχουμε καλύψει την περίπτωση που ένα υπάρχον Url προγραμματίζεται για τοποθετηθεί στην ουρά αλλά τελικά το περιεχόμενό του δεν έχει αλλάξει από την τελευταία φορά. Σε αυτό το σενάριο πρέπει να απορρίψουμε την εν λόγω διεύθυνση. Για αυτό τον λόγο λοιπόν εισήγαμε έναν επιπρόσθετο έλεγχο μέσω της κλάσης **CrawlDB**.

Η προκειμένη κλάση έχει διττό ρόλο. Εκτός από την παραπάνω αρμοδιότητα εξετάζει επίσης αν κάποια διεύθυνση εισάγεται πρώτη φορά οπότε και εισάγεται μέσα στο B-tree αυτό το στοιχείο. Ουσιαστικά αποτελεί το μέσον για να γίνει η δεικτοδότηση και αυτό το κάνει ως εξής- **CheckUpdatedPages**() : Αν δεν βρεθεί το Url_Signature καταχωρημένο τότε καταλαβαίνουμε πως θα επισκεφτούμε την ιστοσελίδα για πρώτη φορά επομένως και υπολογίζουμε το Page_Signature αλλά και το τρέχων timestamp. Αυτά τα δεδομένα καταχωρούνται στα αντίστοιχα πεδία του URL_Entity και εισάγουμε τα δεδομένα στο σύστημα δεικτοδότησης. Αν πάλι το εν λόγω Url_Signature είναι ήδη καταχωρημένο τότε αυτό σημαίνει πως το Url το επισκεφτήκαμε στο παρελθόν(με βάσει τον έλεγχο στο Scheduling) άρα απομένει να δούμε αν το Page_Signature του παρελθόντος είναι διαφορετικό από το τρέχον. Αν ναι τότε και αποδεχόμαστε την διεύθυνση για να την εξετάσουμε στο μέλλον τοποθετώντας την στην ουρά.

4.10 CRAWLER

Σε αυτή την οντότητα υλοποιούμε το βασικό Interface του Crawler. Μέσω αυτής θα αρχικοποιηθεί το σύστημά μας ώστε να ξεκινήσει η διαδικασία εύρεσης νέων διευθύνσεων εισάγοντας στην ουρά της κλάσης Scheduling τα πρώτα links. Αυτό ουσιαστικά επιτελείται και μέσα από την μέθοδο ***InitiateCrawler()*** μέσω της οποίας εισάγουμε μέσα στην ουρά τα links που υπάρχουν καταχωρημένα στο τοπικό αρχείο ***Initial.txt.*** Μετά το πέρας αυτής της διαδικασίας μπορεί να ξεκινήσει το σύστημα να εξερευνεί τις διαθέσιμες διευθύνσεις του παγκόσμιου ιστού.

Όπως είχαμε αναφέρει στο τμήμα ανάλυσης της κλάσης ***Scheduling.java*** ,ορίσαμε την μεταβλητή *MaxServerListSize* ώστε να μπορούμε να σπάμε την διαδικασία της εύρεσης νέων Urls ανά πλήθος μεγαλύτερο από την τιμή αυτή. Για να μεταφέρουμε λοιπόν στο Hdfs σύστημα τα πρώτα Links τα οποία και θα κάνει fetching έπρεπε να εντοπίσουμε τόσα Urls όσο το *MaxServerListSize* και εν συνεχεία να τα καταχωρίσουμε σε ένα αρχείο το οποίο και θα μεταφέραμε στο Hdfs.

4.11 ΦΑΣΗ I: FETCHING ΔΙΕΥΘΥΝΣΕΩΝ

Φτάσαμε στο κύριο τμήμα της εργασίας μας που αφορά το κατέβασμα των ιστοσελίδων και την επεξεργασία του κειμένου που περιέχουν ώστε να χτίσουμε ένα πρότυπο σύστημα αναζήτησης. Από εδώ και έπειτα οι διαδικασίες πραγματοποιούνται υποστηρίζονται από την λειτουργικότητα του Hadoop οπότε και θα δούμε πως χωρίζεται η εργασία μας σε φάσεις Map/Reduce.

Ξεκινάμε λοιπόν την περιγραφή με την κλάση ***Map_Reduce_Crawler.java*** στην οποία και θα δούμε πως επιτύχαμε το παράλληλο fetching των ιστοσελίδων αλλά και την ανάλυση του εσωκλειόμενου κείμενου. Η διαδικασία που θα αναλύσουμε ανάγεται στην φάση Fetching του σχήματος *σελ 28.*

4.11.1 Mapping-Fetching Ιστοσελίδων

Όπως ήδη έχουμε πει, για να ξεκινήσει μία διαδικασία στο Hadoop αυτό που αρκεί είναι να του δώσουμε τα αρχεία εισόδου και μέσω της μεταβλητής **Value** παίρνουμε ανά γραμμή κειμένου τα δεδομένα μας. Αυτό σημαίνει , πως αν θέλουμε να επεξεργαστούμε τα δεδομένα εισόδου απλά χειριζόμαστε την μεταβλητή **value** και βάσει αυτής μετά γράφουμε κώδικα με τρόπο όπως το γνωρίζαμε μέχρι τώρα. Δεν απαιτείται δηλαδή χρήση νημάτων για να εισάγουμε τον παραλληλισμό στην εργασία μας κλπ. Αυτό το φροντίζει το Hadoop.

Στην περίπτωσή μας τώρα, κάθε γραμμή αντιπροσωπεύει και ένα καινούριο Url. Επομένως ,αφού θέλαμε να αποθηκεύσουμε τοπικά το περιεχόμενο του site,

φτιάχναμε κάθε φορά από την μεταβλητή Value ένα αντικείμενο τύπου URL_Entity και ανοίγαμε μια καινούρια σύνδεση με το προκείμενο site. Από εκεί επιτελούσαμε τους ελέγχους δεύτερου επιπέδου όπου και εξετάζαμε αν την προκείμενη διεύθυνση την είχαμε επισκεφτεί στο παρελθόν ή όχι και ανάλογα ελέγχαμε αν είχε υποστεί τροποποίηση το περιεχόμενό της .Όταν οι έλεγχοι ήταν επιτυχείς παίρναμε το περιεχόμενο και τελικά το αποθηκεύαμε τοπικά, βάσει της κλάσης *TextRepository.java.* Επίσης εκτός από το σύνολο του κώδικα της σελίδας , απομονώναμε και το κείμενο σε μορφή String το οποίο και θα διοχετεύσουμε στην φάση του Reduce μέσω του *OutputCollecor.*

Από το περιεχόμενο τώρα κάθε ιστοσελίδας, έπρεπε να βρούμε και τα εσωτερικά links ώστε να εξετάσουμε και νέες διευθύνσεις για το μέλλον. Η διαδικασία αυτή επιτελείται στις σελίδες αυτές που έχουν πέρασε επιτυχώς τους ελέγχους του δεύτερου επιπέδου. Από αυτές στην συνέχεια θα εξάγουμε τις εσωκλειόμενες διευθύνσεις και όταν ο αριθμός αυτών θα ξεπεράσει το όριο του *MaxServerListSize* τότε θα μεταφέρουμε τα δεδομένα αυτά σε αρχεία στο Hdfs directory **//Urls.** Από εκεί και έπειτα μπορεί να πραγματοποιηθούν τα δύο επόμενα βήματα για την ομαδοποίηση των νέων URL σε blocks.

4.11.2 Reducing - Επεξεργασία Κειμένου

Σε αυτό το κομμάτι πριν πούμε πως το υλοποιήσαμε , να επεξηγήσουμε σύντομα τι μορφή θέλαμε να έχουν τα αρχεία εξόδου το Hdfs σύστημα και γιατί. Σκοπός μας αυτής της εργασίας είναι να κατασκευάσει ένα σύστημα αναζήτησης βασισμένο στο Vector Space Model. Όπως έχουμε δει στο αντίστοιχο κεφάλαιο για να κατασκευάσουμε αυτό το μοντέλο χρειάζεται να υπολογιστούν οι μεταβλητές **dfi, tfi , IDfi ,wi=tfi*IDfi.** Αυτό σημαίνει πως πρέπει να υπολογίσουμε **για κάθε ορό σε κάθε site (tfi)** πόσες φορές εμφανίζεται, αλλά και πόσες φορές εμφανίζεται κάθε όρος στο σύνολο των sites δίχως να συμπεριλαμβάνουμε τις διπλοτυπίες της λέξης σε μια ιστοσελίδα (**dfi**).

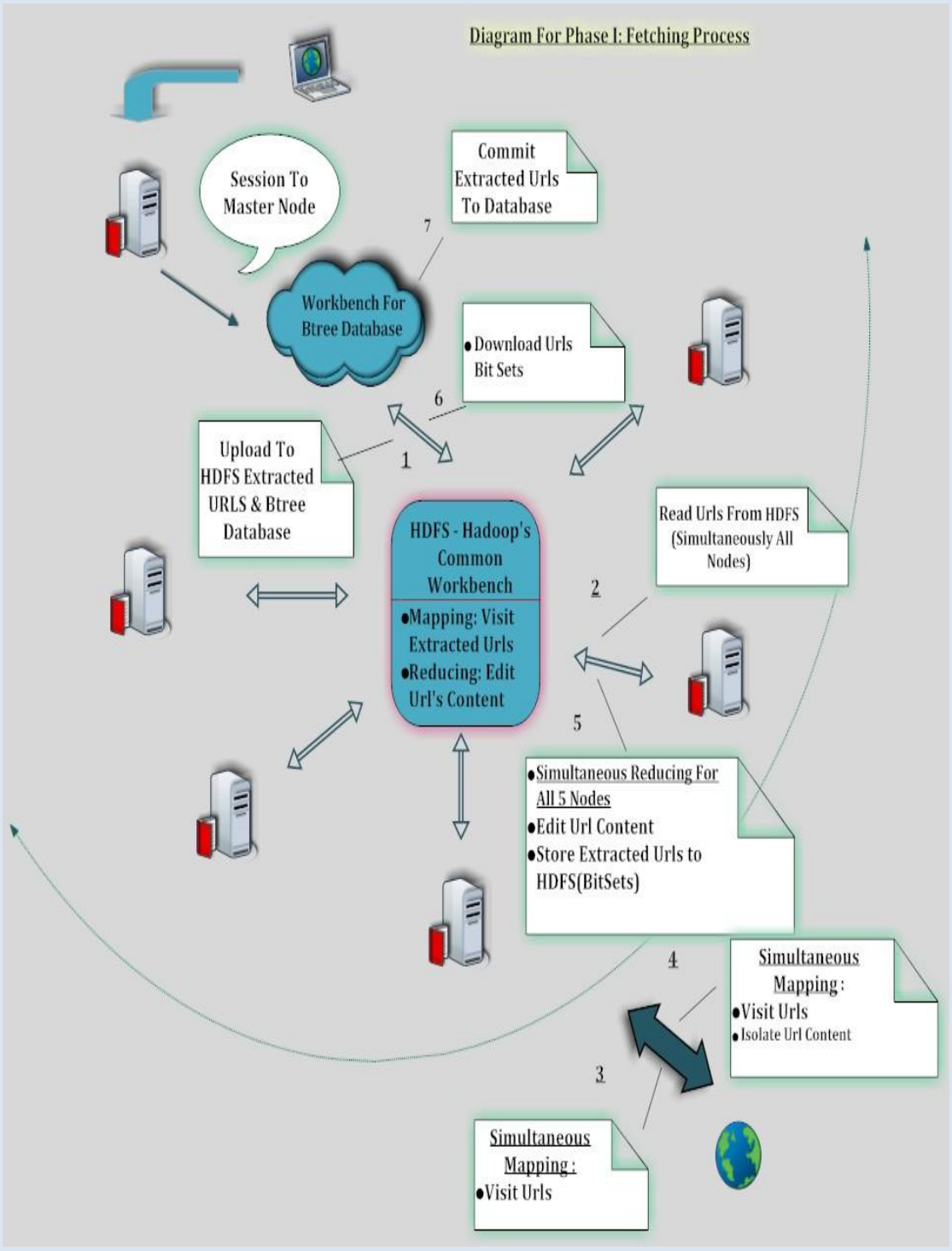
Από τα παραπάνω λοιπόν γίνεται κατανοητό, πως έπρεπε να βρούμε έναν τρόπο να απομονώσουμε σε πρώτη φάση για κάθε site πόσες φορές εμφανίζεται μία λέξη σε αυτό το site ώστε να μπορέσουμε να υπολογίσουμε το tfi. Για να καταφέρουμε κάτι τέτοιο λοιπόν ορίσαμε στον Mapper να διοχετεύει στον Reducer ζευγάρια της μορφής **(Url, κείμενο_Url)**. Έτσι, στο τέλος της φάσης του Mapping θα έχουμε ομάδες κειμένου ανά site οπότε και μπορούμε να αναλύσουμε από το σύνολο των λέξεων ποιες είναι απαραίτητες ,ποιες όχι και να τις τοποθετήσουμε στο Hdfs με τρόπο που να μας βολεύει να τις επεξεργαστούμε ώστε να υπολογιστούν αυτοί οι όροι.

Φτάνουμε λοιπόν στο σημείο που πρέπει να διαχωρίσουμε από το σύνολο του κειμένου τις λέξεις. Όπως είχαμε δει και στο εισαγωγικό κεφάλαιο του Hadoop,

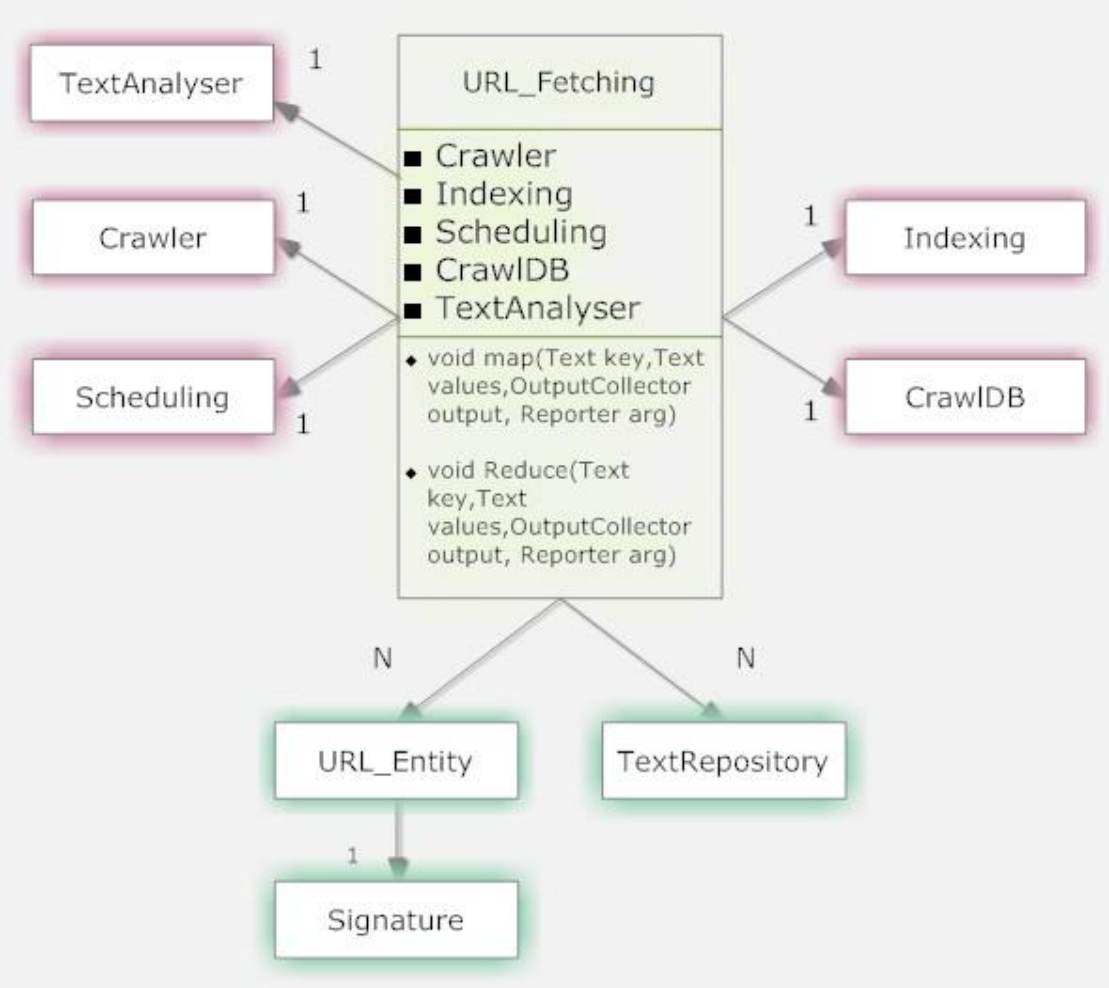
κατά την φάση του Reduce, για να διαχειριστούμε τα δεδομένα κάθε ομάδας χρησιμοποιούμε τα ορίσματα **key ,value** .Το πρώτο αντιστοιχεί στο κλειδί βάσει του οποίου συλλέγονται σε ομάδες τα πεδία values από τον collector κατά το map. Ετσι και εδώ, το key θα είναι το Url και το πεδίο values το κείμενο. Για να ξεχωρίσουμε λοιπόν τις λέξεις που μας αφορούν χρησιμοποιούμε ένα tokenizer και απομονώνουμε τους όρους ανά κενό.

Από εκεί, θα κάνουμε ελέγχους για να δούμε αν η κάθε λέξη πρέπει να καταχωρηθεί στην έξοδο του Reducer. Πρώτος έλεγχος είναι να δούμε αν ο κάθε όρος που προσεγγίζουμε ανήκει στην κατηγορία των **StopWords** . Αν όχι τότε καταλαβαίνουμε πως μπορούμε να διοχετεύσουμε τον όρο στην έξοδο. Δεν θα τον καταχωρήσουμε με αυτή την μορφή αλλά ως **StemmedWord** χρησιμοποιώντας τον *IteratedLovinsStemmer*.

Κατά αυτόν τον τρόπο, στην έξοδο του Hdfs θα εμφανιστούν γραμμές της μορφής **Url \t word** ταξινομημένα με βάσει το site. Χρησιμοποιώντας αυτή την μορφή θα καταφέρουμε με άλλα Hadoop Jobs να υπολογίσουμε τις παραμέτρους του Vector Space Model.



1η Φάση: Σχηματική Αναπαράσταση Fetching-Map/Reduce Βήματα



Εποπτικό Διάγραμμα Εμπλεκόμενων Κλάσεων Κατά το Fetching

4.12 PACKAGE WORDPROCESSING –*TextAnalyser.java*

Στην προηγούμενη ενότητα αναφέραμε στο τμήμα του *Reduce* πώς χρησιμοποιήσαμε την κλάση TextAnalyser.java για να εξετάσουμε αν οι εν λόγω όροι ανήκουν στην κατηγορία των **StopWords** . Επίσης , αναφέραμε την χρήση της κλάσης IteratedLovinsStemmer την οποία και χρησιμοποιήσαμε για να κάνουμε stemming των λέξεων. Όλα αυτά τα εργαλεία τα βρίσουμε στο προκείμενο package και σκοπό έχει να διαχειριστεί τις υποψήφιες λέξεις ώστε να της διοχετεύσουμε σε επόμενα επίπεδα για το χτίσιμο του Vector Space Model.

¨Όπως έχουμε πει και στα εισαγωγικά κεφάλαια, στην πράξη κατά την κατασκευή μηχανών αναζήτησης δεν χρησιμοποιούμε τους όρους με την κλασική τους μορφή αλλά αποκόβωντας τις περιττές καταλήξεις και μετατρέποντας τα κεφαλαία σε πεζά. Αυτή η διαδικασία ονομάζεται stemming και στην περίπτωσή

μας χρησιμοποιήσαμε έτοιμο εργαλείο για αυτό τον σκοπό[5]. Έτσι, μέσω της συγκεκριμένης κλάσης διοχετεύουμε τον όρο σε String και το αποτέλεσμα της αλγορίθμου μας δίνει τον όρο στην επιθυμητή μορφή.

Όσον αφορά τώρα τον έλεγχο για το αν ένας όρος είναι Stop word ή όχι , να πούμε πως ήταν ένα απαραίτητο βήμα ώστε να απομονώσουμε όρους που δεν βοηθάνε να διακρίνουμε την ουσία –έννοια του κειμένου. Πχ λέξεις όπως το συνδετικές αντωνυμίες ,τα σημεία στίξης ,επιρρήματα κλπ , δεν παρέχουν χρήσιμη πληροφορία για να αποκωδικοποιήσουμε το νόημα . Επομένως ,τα αφαιρούμε επιλέγοντας να δεικτοδοτήσουμε κατά βάση ουσιαστικά, ρήματα κλπ. Για αυτόν τον λόγο χρησιμοποιήσαμε μία StopList [6]για το αγγλικό λεξικό και βάση αυτής ελέγχαμε αν κάποιος όρος ανήκει σε αυτήν η όχι.

4.13 ΥΠΟΛΟΓΙΣΜΟΣ ΕΜΦΑΝΙΣΕΩΝ ΟΡΩΝ ΑΝΑ ΔΙΕΥΘΥΝΣΗ -Tfi

Όταν ξεκινήσαμε να αναλύουμε τις κλάση Map_Reduce_Crawler αναφέραμε την ανάγκη του υπολογισμού του Tfi για την κατασκευή του Vector Space Model. Εδώ υποδεικνύουμε τον τρόπο για την μέτρηση του αριθμού των εμφανίσεων κάθε όρου σε μια ιστοσελίδα και βέβαια χρησιμοποιώντας τα πλεονεκτήματα του Hadoop. Για πετύχουμε λοιπόν το ζητούμενο χρησιμοποιήσαμε ένα πολύ διαδεδομένο παράδειγμα για τους χρήστες του Hadoop ,το **WordCount.java [7],** κάπως παραλλαγμένο στα δικά μας δεδομένα για να μετρήσουμε κάθε λέξη σε κάθε site.

Η διαδικασία που θα περιγράψουμε συνοψίζεται σε δύο βήματα : ένα Map και ένα Reduce. Τα δεδομένα εισόδου σε αυτή την Hadoop Job έρχονται από την έξοδο του προηγούμενου βήματος Map/Reduce - **Out\\Page_Output** και με βάση την κάθε γραμμή των κειμένων που διατίθενται ξεκινάμε να υπολογίζουμε για κάθε μοτίβο (**Url word)** πόσες φορές εμφανίζεται συνολικά σε όλα τα κείμενα εισόδου. Η διαδικασία που θα αναλύσουμε ανάγεται στην φάση WordCount του σχήματος *σελ 28.*

4.13.1 WordCount-Mapping

Αυτό που πρέπει να πούμε για να κάνουμε πιο εύληπτη την διαδικασία υπολογισμού είναι πως τα αρχεία εισόδου , είχαμε επιλέξει από το προηγούμενο βήμα να είναι στην μορφή ***URL \t word*** Πχ για το www.yahoo.com με όρο το **yah** θα είχαμε στην έξοδο.**http://www.yahoo.com/ *yah.*** Σκεφτείτε τώρα πώς η λέξη *yah*

[5] Πηγή:http://www.cs.waikato.ac.nz/~eibe/stemmers/
[6] Πηγή: http://truereader.com/manuals/onix/stopwords1.html
[7] Πηγή: http://wiki.apache.org/hadoop/WordCount

εμφανίζεται μέσα στο κείμενο του **www.yahoo.com** πχ 3 φορές. Αυτό που περιμένουμε λοιπόν στα αρχεία εξόδου **Out\\Page_Output** είναι να βρούμε το μοτίβο **http://www.yahoo.com/ *yah*** 3 φορές. Αυτό αυτόματα σημαίνει πως για να υπολογίζουμε **για κάθε site** τον αριθμό εμφανίσεων **ανά όρο** πρέπει στην φάση του Mapping να χρησιμοποιήσουμε ως κλειδί –**key** - την κάθε γραμμή από τα αρχεία εισόδου.

Τι θα χρησιμοποιήσουμε όμως κατά το mapping ως *value* για να καταφέρουμε να μετρήσουμε τον αριθμό των εμφανίσεων στην φάση του Reduce; Η απάντηση είναι απλή: Θα ορίσουμε έναν ακέραιο με τιμή 1. Δηλαδή για το προηγούμενο παράδειγμα για κάθε *key* που εμφανίζεται θα κάνουμε αντιστοίχηση στην τιμή 1. Έτσι , το cluster που δημιουργείται σε αυτή την περίπτωση θα είναι:

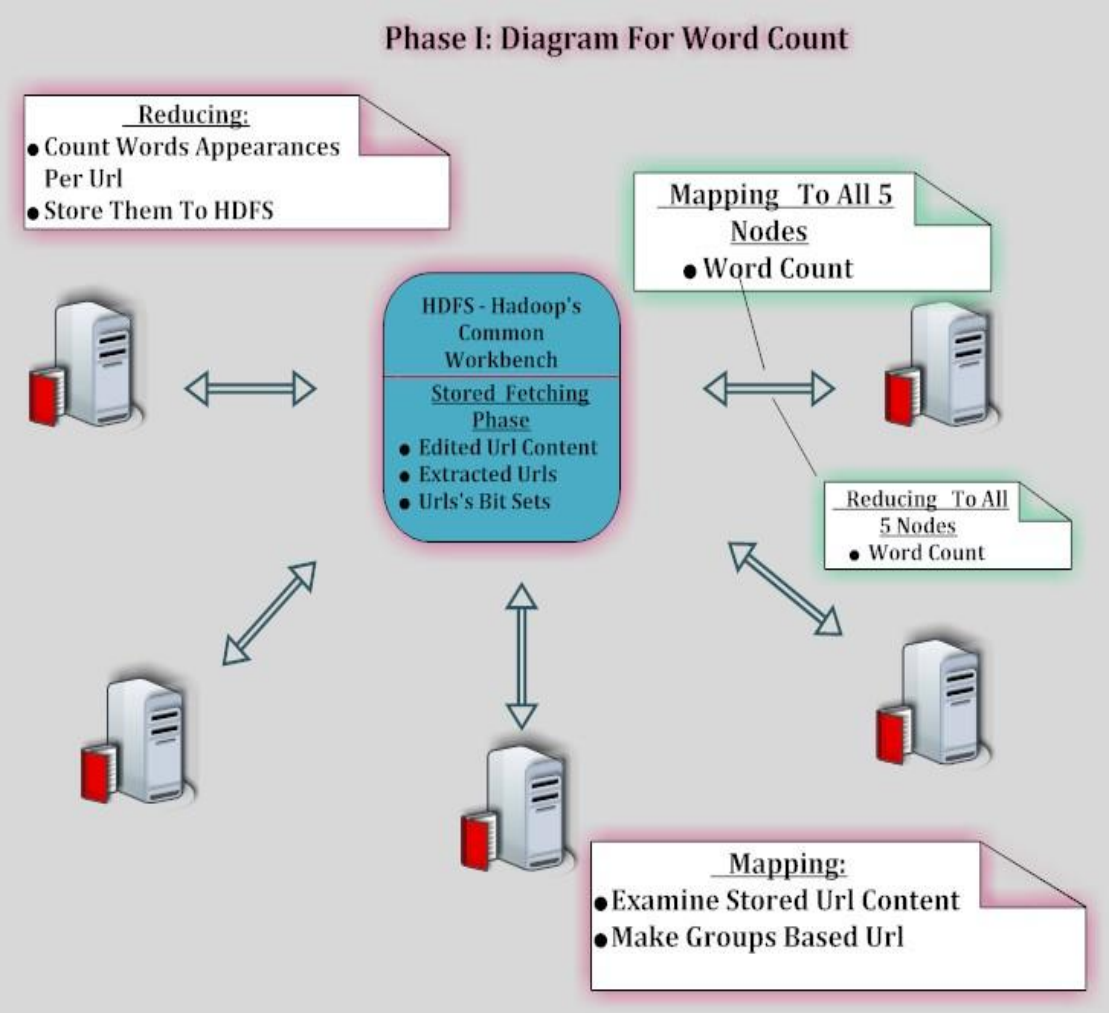
http://www.yahoo.com/ yah <-Key

[1 1 1] <-Values(3 εμφανίσεις του yah στο Mapping)

Cluster Που Θα Διοχετευτεί Στον Reducer

4.13.2 WordCount-Reducing

Φτάνουμε λοιπόν στο σημείο που πρέπει να υπολογίσουμε το σύνολο των εμφανίσεων. Η δουλειά μας εδώ είναι πολύ εύκολη γιατί αυτό που πρέπει μονάχα να κάνουμε είναι για κάθε cluster με διαφορετικό κλειδί ,να αθροίσουμε όλες τις τιμές του των εμφανίσεων του **1.** Επομένως ,στην περίπτωσή μας θα αθροίσουμε για το *key* – ***http://www.yahoo.com/ yah*** τις 3 εμφανίσεις του 1. Άρα στο τέλος αυτής της διαδικασίας θα έχουμε τον **Tfi** που απλά θα εξάγουμε στο Directory - **Out//Output//** του Hdfs. Αυτό βέβαια θα γίνει μέσω του Output-Collector οπότε και στα τελικά αρχεία εξόδου θα βρούμε τα δεδομένα στην μορφή **http://www.yahoo.com /t yah /t 3 .** Γιατί τώρα αποφασίσαμε να εξάγουμε τα δεδομένα αυτής της φάσης σε αυτή την μορφή; Η απάντηση θα δοθεί πιο κάτω κατά την προσπάθεια μας να υπολογίσουμε και τους άλλους όρους του Vector Space Model.



1η Φάση: Καταμέτρηση Όρων Ανά Διεύθυνση- Map/ Reduce Βήματα

4.14 ΦΑΣΗ ΙΙ: *Vector Space Model*

Σε αυτό το σημείο είμαστε έτοιμοι να ξεκινήσουμε την διαδικασία υπολογισμού των συντελεστών του μοντέλου για το χτίσιμο της μηχανής αναζήτησής . Πριν όμως ξεκινήσουμε την ανάλυση των κλάσεων πρέπει να πούμε κάποια στοιχεία που διακρίνουν αυτό το τμήμα της εργασίας μας.

Βασικό χαρακτηριστικό που πρέπει αν αναφέρουμε είναι πως και αυτό το κομμάτι είναι πλήρως ανεξάρτητο από τα προηγούμενα αφού για να ξεκινήσει η διαδικασία που θα περιγράψουμε παρακάτω το μόνο που απαιτείται είναι το σύνολο των αρχείων που βρίσκονται στο Hdfs directory **Out//Output//.** Επιπλέον, για να ξεκινήσουμε την φάση του υπολογισμού των χαρακτηριστικών του Vector Space Representation όπως θα δούμε απαιτείται εξ' αρχής να γνωρίζουμε τον ακριβή αριθμό των Urls που έχουμε εξετάσει για να υπολογίσουμε το IDF. Αυτή , η παράμετρος μας αναγκάζει λοιπόν να μην θέτουμε το σύστημα να συ-

νεχίζει την εξέταση των δεδομένων που έχουμε στο **Out//Output//** αμέσως μετά το πέρας της διαδικασίας του WordCount .

Γιατί γίνεται αυτό; Πολύ απλά γιατί δεν μπορούμε να ξέρουμε μέχρι να τελειώσει η ανάλυση πόσα Urls έχουμε εξετάσει συνολικά. Αυτός ο αριθμός αυξάνεται καθώς προγραμματίζουμε να μεταφέρουμε στο Hdfs συνεχώς νέες διευθύνσεις αλλά σίγουρα δεν μπορούμε πούμε εξ' αρχής ότι στην πορεία θα συναντήσουμε έναν συγκεκριμένο αριθμό sites. Η διαδικασία που θα αναλύσουμε ανάγεται στην φάση Word_IDF του σχήματος *σελ 29.*

4.15 ΥΠΟΛΟΓΙΣΜΟΣ IDF –Word_IDF.java

Έχουμε ήδη πει πως για τον υπολογισμό του IDF απαιτείται IDF=$\log\frac{D}{dFi}$, όπου το D είναι ο αριθμός του συνόλου των κειμένων και το dFi είναι ο αριθμός του συνόλου των εμφανίσεων ενός όρου i σε όλα τα κείμενα χωρίς να συμπεριλαμβάνουμε τις διπλοτυπίες του όρου ανά κείμενο. Το D είναι ένας αριθμός που τον γνωρίζουμε αφού τελειώσει και η διαδικασία του WordCount. Πως θα υπολογίσουμε όμως το dFi; Η λογική είναι παρόμοια με το WordCount παίρνοντας ως κλειδί το τμήμα ***word*** από τα αρχεία εξόδου και θέτοντας στο value στον Collector στην τιμή 1.

Μέχρι στιγμής έχουμε δει πως τα αρχεία εξόδου στο **Out//Output//** έχουν την μορφή **URL \t word \t TFi**. Αυτό που πρέπει να καταλάβουμε είναι πως με την διάταξη που έχουν τα δεδομένα για κάθε site, ένας όρος *word* θα εμφανίζεται μία φορά σε κάθε site. Πχ για το **http://www.yahoo.com/ yah 3** του προηγούμενου παραδείγματος , το key ήταν όλο το τμήμα **http://www.yahoo.com/ yah** άρα δεν γίνεται στα αρχεία εξόδου να υπάρχει και άλλο ένα τέτοιο κλειδί. Άρα για **κάθε όρο** και σε **κάθε site** έχουμε εξαλείψει τις διπλοτυπίες των όρων -τις κρατάμε στην τρίτη στήλη των αρχείων- ,δηλαδή για να υπολογίσουμε για κάθε λέξη το dFi πρέπει απλά να ελέγξουμε σε όλα τα αρχεία εισόδου πόσες φορές εμφανίζεται συνολικά κάθε όρος.

4.15.1 Word_IDF - Mapper

Το Implementation του Mapper έγινε από την κλάση *Total_Appearances* οπότε για να απομονώσουμε από κάθε γραμμή εισόδου- από το **Out//Output// -** τον όρο απλά σπάμε την γραμμή βάσει τον χαρακτήρα **\t.** Τα δεδομένα της γραμμή εκχωρούνται σε έναν πινάκα 3 θέσεων οπότε το στοιχείο της δεύτερης θέσης του πίνακα που αντιστοιχεί και στον όρο διοχετεύεται ως key στον collector. Για το όρισμα value του collector απλά θέτουμε την τιμή 1 που ουσιαστικά αντιστοιχεί και σε μία εμφάνιση της λέξης για αυτό το Url. Παρατίθενται παρακάτω

και ένα στιγμιότυπο για το cluster που δημιουργείται πριν διοχετευτεί στον Reducer.

yah **<-Key**

[1 1 1 1] **<-Values(4 εμφανίσεις του yah στο σύνολο των sites)**

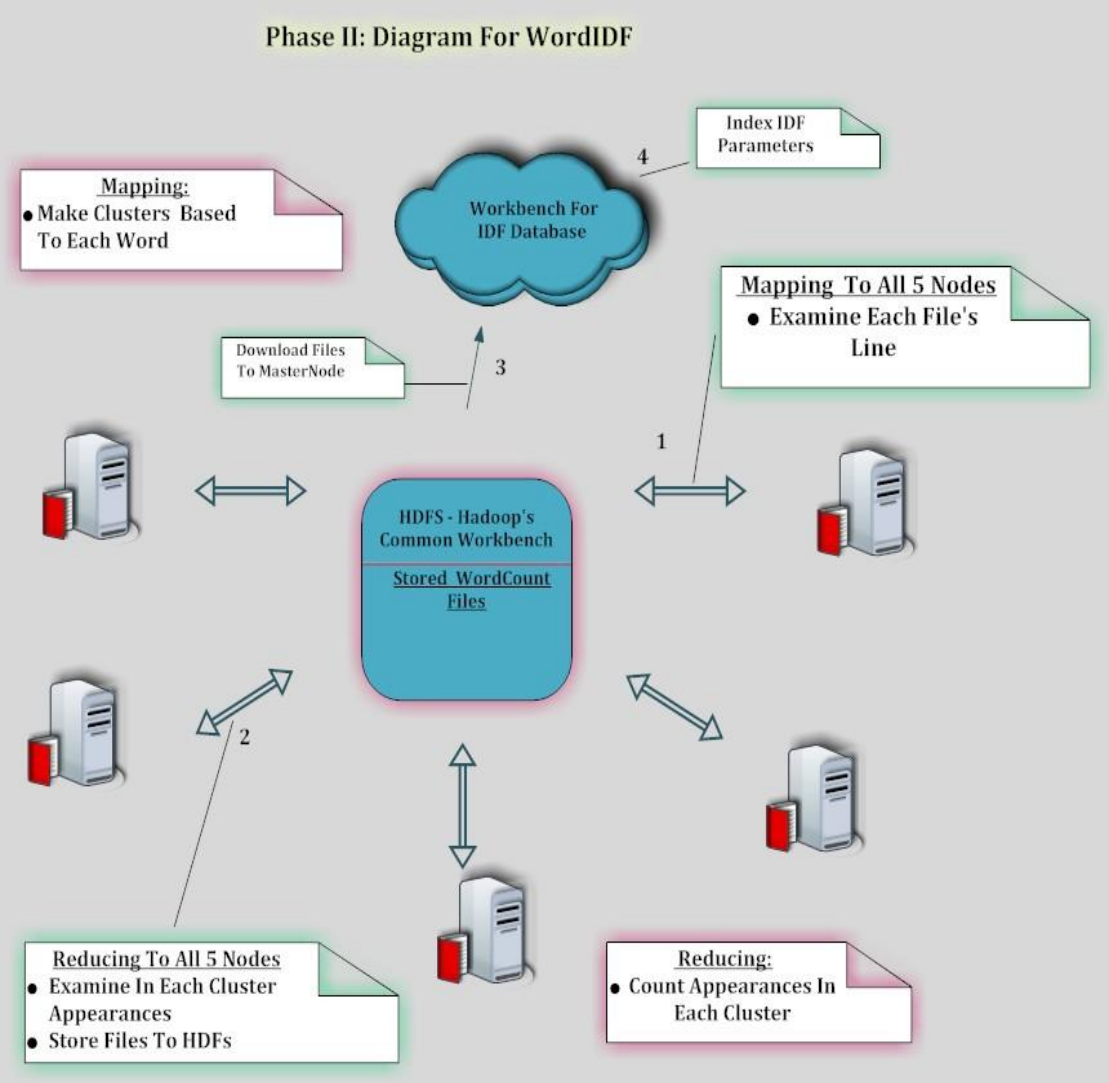
Cluster Που Θα Διοχετευτεί Στον Reducer Για Το Word_IDF

4.15.2 Word_IDF - Reducer

Το Implementation του Reducer έγινε από την κλάση IDF_Count που είναι εσωτερική κλάση της Word_IDF. Σε αυτό το τμήμα σκοπός είναι να υπολογιστεί το dFi και να καταγραφεί στα αρχεία εξόδου και για κάθε όρο ,ο IDF. Για να επιτευχθεί αυτό απλά χρειάζεται να αθροίσουμε τα στοιχεία του κάθε cluster ,με key το *word,* και να υπολογίσουμε τον αντίστοιχο λογάριθμο . Επομένως ,τα στοιχεία εξόδου θα τοποθετηθούν στο HDFS directory **Out\\IDF** και με μορφή ***Word \t IDF.***

Από αυτό το σημείο και μετά τα δεδομένα που υπολογίσαμε για τους όρους IDf θα μεταφερθούν τοπικά όπου και θα δεικτοδοτηθούν από βάση. Ο λόγος που το κάναμε αυτό είναι γιατί στην συνέχει όπως θα δούμε για τον υπολογισμό των βαρών αλλά και για τον υπολογισμό της ομοιότητας με τα Query του χρήστη θα χρειαστούν αυτά τα δεδομένα για να πραγματοποιηθεί το μοντέλο αναπαράστασης διανυσμάτων. Η μία λύση λοιπόν ήταν απλά να δεικτοδοτήσουμε αυτήν την πληροφορία σε βάση είτα να εντάξουμε την πληροφορία των IDf των όρων μέσα στα κείμενα εισόδου.

Στην πράξη πειραματιστήκαμε και με τους δύο τρόπους αλλά ο λόγος που αποφασίσαμε να δεικτοδοτήσουμε αυτά τα δεδομένα αντί να τα καταχωρήσουμε μέσα στα αρχεία ήταν γιατί αυξάναμε συνολικά των όγκο των αρχείων κατά μεγάλο ποσοστό. Αυτό συμβαίνει γιατί επαναλαμβάναμε πολλές φορές τη ίδια πληροφορία. Πχ αν την λέξη color την βρίσκαμε σε 100 διαφορετικά sites έπρεπε δίπλα πό την τελευταία στήλη να τοποθετήσουμε και το αντίστοιχο IDf του όρου IDF κλπ.



2η Φάση :Υπολογισμός IDF Κάθε Όρου- Map/Reduce Βήματα

4.16 ΥΠΟΛΟΓΙΣΜΟΣ ΒΑΡΩΝ ΤΩΝ ΚΕΙΜΕΝΩΝ- Doc_Weights.java

Από την προηγούμενη ενότητα είδαμε πως τα αρχεία εξόδου από την φάση *WordCount* τέθηκαν στην μορφή **URL \t word \t Tfi.** Σε αυτό το βήμα θα υπολογίσουμε το συνολικό βάρος ενός κειμένου-site που προκύπτει από την ρίζα του αθροίσματος των τετραγώνων των επιμέρους wi ανά κείμενο και θα το τοποθετήσουμε στην έκτη στήλη των δεδομένων εξόδου. Επομένως , η τελική μορφή των αρχείων θα είναι ως εξής: **URL \t W.**

Ο λόγος που επιλέγουμε να τοποθετήσουμε στην τελευταία στήλη κάθε γραμμής το συνολικό βάρος του κειμένου είναι για να μπορέσουμε κατά την υποβολή των ερωτημάτων του χρήστη να αποφύγουμε τους υπολογισμούς αυτού του βήματος .Έτσι, έχουμε άμεσα τα βάρη στην διάθεσή μας, αφού πρώτα δεικτοδοτηθούνε με χρήση βάσης, για να τα συγκρίνουμε με το αντίστοιχο του ερωτήματος

του χρήστη. Και πάλι σε αυτό το βήμα χρησιμοποιούμε για τους υπολογισμούς το Hadoop χωρίζοντας την διαδικασία σε φάσεις Map /Reduce. Η διαδικασία που θα αναλύσουμε ανάγεται στην φάση Weights του σχήματος *σελ 29.*

4.16.1 Doc_Weights -Mapper

Ξεκινάμε την διαδικασία του mapping παίρνοντας τα αρχεία εισόδου από το directory **Out\\Output** στο Hdfs σύστημα. Τα δεδομένα αυτά όπως έχουμε πει περιέχουν στην τελευταία τους στήλη την συχνότητα εμφάνισης του κάθε όρου ανά site, επομένως για να μπορέσουμε να υπολογίσουμε το καθολικό βάρος πρέπει να απομονώσουμε τα βάρη που αντιστοιχούν σε κάθε site και έπειτα στο βήμα του reduce να κάνουμε τους απαραίτητους υπολογισμούς.

Αυτό λοιπόν σημαίνει πως σε αυτή τη φάση πρέπει να χρησιμοποιήσουμε ως key στον collector το URL που βρίσκεται στην πρώτη στήλη των δεδομένων εισόδου. Από την άλλη για να μπορέσουμε να φέρουμε τα δεδομένα στη μορφή **URL \t W** απαιτείται να διοχετεύσουμε στον reducer από την ολόκληρη γραμμή εισόδου το κομμάτι **word \t Tfi** , επομένως το πεδίο value του collector θα δεχθεί από την τρέχουσα γραμμή που εξετάζει ο mapper το τμήμα αυτό. Κατά αυτό τον τρόπο δημιουργούμε ομάδες ανά site έχοντας όλα τα στοιχεία για να υπολογίσουμε το συνολικό βάρος στον reducer.

Παραθέτουμε ένα εποπτικό παράδειγμα των cluster που δημιουργούνται σε αυτό το βήμα βάσει των πινάκων *2,3*.

Παράδειγμα 2:

Cluster1

http://m.yahoo.com <- Collector Key
[color \t 43] <- Collector Values
[chang \t 1]

Cluster2

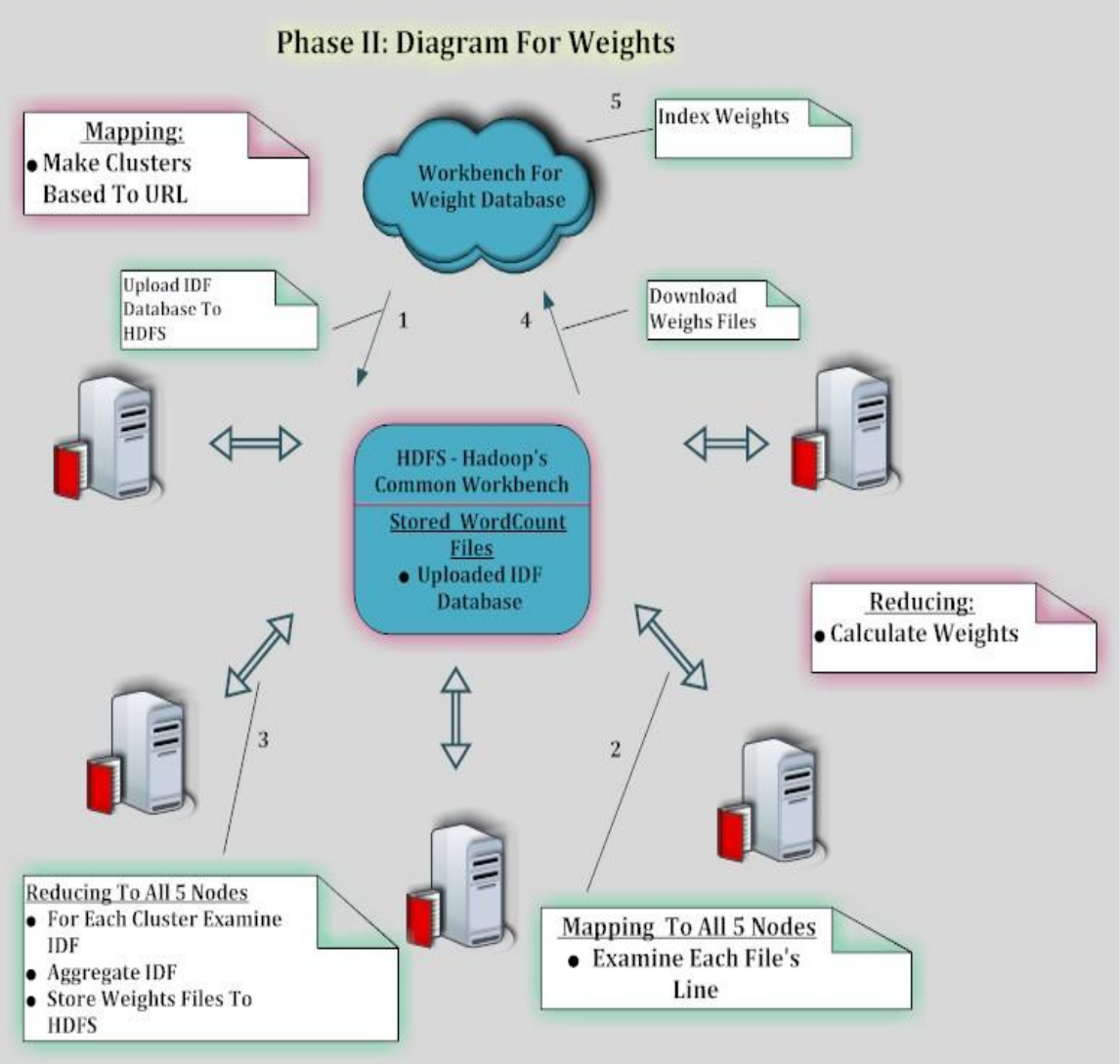
http://www.yahoo.com <- Collector Key
[assist \t 1] <- Collector Values

Παράδειγμα 2.Cluster Που Θα Διοχετευτεί Στον Reducer Για Την Φάση Doc_Weights

4.16.2 Doc_Weights -Reducer

Σε αυτή την φάση θα επεξηγήσουμε τον τρόπο με τον οποίο υπολογίσαμε τα συνολικά βάρη για κάθε site μέσω του αθροίσματος των επιμέρους βαρών ανά όρο που υπάρχουν σε μία διεύθυνση. Από το παραπάνω παράδειγμα αυτό που βλέπουμε είναι πως για το σύνολο των δεδομένων που υπάρχουν σε κάθε cluster τα επιμέρους wi υπολογίζονται από το γινόμενο της συχνότητας εμφάνισης του όρου επί το IDF. Επομένως για να βρούμε το άθροισμα των τετραγώνων των βαρών απλά εξετάζουμε όλες τις τιμές της δεύτερης στήλης των πεδίων *values* και πολλαπλασιάζουμε με τον IDf που έχουμε ήδη καταχωρήσει στην βάση μας *Hash_IDF* ,και αθροίζουμε τα τετράγωνά τους.

Από κει και έπειτα ο υπολογισμός των βαρών πραγματοποιείται αθροίζοντας τα επιμέρους γινόμενα ανά site και στην συνέχεια στην έξοδο του Reducer μεταφέρουμε μονάχα τις τιμές των βαρών ανά site. Επομένως στα τελικά αρχεία έχουμε καταχωρημένα τα στοιχεία των βαρών άρα μπορούμε εύκολα να δεικτοδοτήσουμε αυτές τις τιμές βάσει την κάθε διαδικτυακή διεύθυνση.



2η Φάση: Υπολογισμός Βαρών-Map/Reduce Βήματα

4.17 ΤΟΠΙΚΗ ΔΕΙΚΤΟΔΟΤΗΣΗ ΟΡΩΝ IDF-*Disk_Word_Idf.java*

Ήδη έχουμε αναλύσει τον τρόπο με τον οποίο υπολογίζουμε για κάθε όρο την μεταβλητή IDF. Επίσης , τα αποτελέσματα αυτής της διαδικασίας εναπόκεινται το directory **Out\\Idf** στο Hdfs σύστημα. Η εύλογη λοιπόν απορία που προκύπτει είναι για ποίο λόγο πρέπει να δεικτοδοτήσουμε και μάλιστα τοπικά αυτά τα δεδομένα .Ο λόγος ,όπως θα δούμε και παρακάτω ,είναι για να υπολογίζουμε βάσει της δεικτοδότησης αυτής το βάρος του ερωτήματος του χρήστη. Η αιτιολόγηση θα δοθεί στην επόμενη ενότητα. Προς το παρόν αυτό που θα δούμε είναι την λογική της κατασκευής του συστήματος δεικτοδότησης.

Ο τρόπος με τον οποίο δομήσαμε αυτό το κομμάτι είναι με ένα Btree που αποθηκεύει τα στοιχεία δεικτοδότησης τοπικά με πεδία *key* τον όρο και *value* το IDF. Ο λόγος που επιλέξαμε αυτή την δομή είναι για να έχουμε γρήγορη αναζήτηση των

όρων που μας ενδιαφέρουν και να ανακτούμε το IDF ώστε να υπολογίσουμε το Q_W (βάρος query).

Πώς το υλοποιήσαμε τώρα: Αρχικά να θυμίσουμε την μορφή των κειμένων στο **Out\\Idf.** Τα κείμενα αυτά περιέχουν τα πεδία **Word \t IDF** ενώ μέσα στο σύνολο όλων των κειμένων δεν υπάρχουν διπλοτυπίες των όρων. Κάθε λέξη εμφανίζεται μια φορά. Αυτό μας διευκόλυνε αρκετά γιατί κατά την δεικτοδότηση των δεδομένων δεν χρειαζόταν να ανησυχούμε για επιπρόσθετους ελέγχους για τα στοιχεία που έχουν ήδη καταχωρηθεί μία φορά. Αυτό που απέμενε λοιπόν ήταν να επεξεργαστούμε ανά γραμμή τα δεδομένα και να δεικτοδοτήσουμε την πρώτη στήλη στο *key* και την δεύτερη στο *value.* Η διαδικασία αυτή περιγράφεται στην μέθοδο **CreateIdfHashtable()** που ουσιαστικά είναι και όλο το σώμα της κλάσης.

Γιατί όμως επιλέξαμε μία τέτοια τακτική; Κάποιος θα μπορούσε να πει πως το να μεταφέρουμε την δεικτοδότηση σε επίπεδο δίσκου είναι μια αφελής επιλογή γιατί η αναζήτηση θα γίνεται πολύ αργά. Πράγματι, δεν υπάρχει μέτρο σύγκρισης ανάμεσα στην τοπική πρόσβαση και στο να προσπελαύνουμε τα δεδομένα στην μνήμη αλλά κάτι τέτοιο στην πραγματικότητα δεν μας νοιάζει γιατί η αναζήτηση θα πραγματοποιείται πάνω στους όρους του χρήστη που στην πράξη είναι πολύ λίγοι. Επιπρόσθετα, ο βασικότερος λόγος για αυτή την επιλογή είναι πώς κάνουμε την διαδικασία αυτή μία φορά και τα στοιχεία παραμένουν διαθέσιμα και για μελλοντικές αναζητήσεις ενώ αν επιλέγαμε να τα μεταφέρουμε στην μνήμη πρώτον θα έπρεπε να επαναλαμβάνουμε την διαδικασία αυτή όταν θα αρχικοποιούσαμε ξανά το project και δεύτερον πιθανώς θα είχαμε πρόβλημα εξαιρέσεων (**JavaOutOfMemory Exception**) αν έπρεπε να διαχειριστούμε πολλούς όρους.

Τέλος , ένας άλλος λόγος που μας εξυπηρετεί καλύτερα αυτή η αρχιτεκτονική είναι ότι μπορούμε να την «τρέξουμε» ανεξάρτητα από το υπόλοιπο σώμα της εργασίας από την στιγμή που γίνει η εξαγωγή των κειμένων στο Out\\Idf ,παράλληλα σε ένα άλλο μηχάνημα την στιγμή που τρέχει ένα άλλο τμήμα της εργασίας μας .

4.18 ΣΥΣΤΗΜΑ ΑΝΑΖΗΤΗΣΗΣ ΕΠΕΡΩΤΗΣΕΩΝ ΧΡΗΣΤΗ

Φτάσαμε στο τελικό στάδιο στο οποίο και γίνεται η ανάκτηση των διευθύνσεων βάσει των ερωτημάτων του χρήστη. Σε αυτό το τμήμα θα δούμε πώς από το σύνολο των δεδομένων που έχουμε στην διάθεσή μας ανακτούμε τα αρχεία που προσεγγίζουν καλύτερα το query του χρήστη. Και πάλι σε αυτό το βήμα χωρίσαμε τις διαδικασίες που έπρεπε να γίνουν σε Map/Reduce jobs.

Να σημειώσουμε σε αυτό το σημείο πώς σκοπός της παρούσας εργασίας δεν είναι να κατασκευάσουμε ένα σύστημα που να απαντά σε πραγματικό χρόνο σε

ερωτήματα του χρήστη καθώς για αυτό στον σκοπό πρέπει να κατασκευαστούν διάφορες δομές που θα εκμηδενίζουν τον χρόνο επεξεργασίας (Inverted File, HashTables κλπ) . Σκοπός αυτού του βήματος είναι να ελεγχθεί η ορθότητα του μοντέλου αναπαράστασης που κατασκευάσαμε χρησιμοποιώντας τα δεδομένα που έχουμε αποθηκεύσει στο Hdfs σύστημα από την WordCount Process αλλά και τα δεδομένα IDF, Weights από το σύστημα δεικτοδότησης.

Η διαδικασία που θα περιγράψουμε ξεκινά από το ερώτημα το χρήστη για το οποίο και υπολογίζεται το βάρος Q_W αφού πρώτα γίνει αφαίρεση των Stop words και Stemming. Τα δεδομένα αυτά στην συνέχεια τοποθετούνται στο αρχείο *QueryFile.txt* απ' όπου και στην συνέχεια θα μπορεί κάποιος Mapper να συλλέξει τα απαραίτητα στοιχεία για τον υπολογισμό της ομοιότητας των κειμένων με το ερώτημα του χρήστη.

4.18.1 Vector_Model_Queries-Mapper

Για να καταφέρουμε τώρα να υπολογίσουμε ποια από τα διαθέσιμα sites περιγράφουν καλύτερα το ερώτημα πρέπει σε πρώτη φάση να απομονώσουμε εκείνα που περιέχουν έναν ή περισσότερους όρους από το query. Για να γίνει αυτό το μόνο που έχουμε να κάνουμε είναι να συγκρίνουμε για κάθε όρο του Query την δεύτερη στήλη των δεδομένων που περιέχει τις λέξεις και να διοχετεύσουμε στον Reducer τις διευθύνσεις εκείνες που περιέχουν πάνω από έναν όρους . Έτσι, στα πεδία του Collector θέτουμε ως *key* το Url και ως *value* ολόκληρη τη γραμμή για να καταφέρουμε στο επόμενο βήμα να βρούμε το εσωτερικό γινόμενο του query με το βάρος του κάθε site. Επομένως, για όλο το σύνολο των αρχείων που εναπόκεινται στο **Results\\Weights** του HDFS εξετάζουμε κάθε γραμμή και εκείνες που πληρούν τις προϋποθέσεις μεταφέρονται στο επόμενο επίπεδο.

Παραθέτουμε ένα στιγμιότυπο των clusters που δημιουργούνται κατά την φάση του Map για το Query «I Change Colors Often »:

URL	Word	TFi
http://m.yahoo.com	*color*	43
http://m.yahoo.com	chang	1
http://www.yahoo.com	assistan	1

2.Πίνακας με δεδομένα από Out//Output//

Word	IDF
color	0.383100742
chang	0.108530253
assistan	0.131098346

3.Πίνακας με δεδομένα από Out//IDF//

Παράδειγμα 3:

Cluster1

http://m.yahoo.com/ <- Collector Key

[color \t 43] <- Collector Values

[chang \t 1]

Cluster2

http://www.adobe.com <- Collector Key

[color \t 1] <- Collector Values

Παράδειγμα 3.Cluster Που Θα Διοχετευτεί Στον Reducer Για Το Vector_Model_Queries

Ένα σημείο στην ανάλυσή μας και που δεν έχουμε επεξηγήσει μέχρι τώρα είναι ο λόγος που αποφασίσαμε να δεικτοδοτήσουμε στον δίσκο τους όρους (word,IDF). Η απάντηση μπορεί να δοθεί τώρα κοιτώντας τι διαδικασίες γίνονται κατά τη φάση *Vector_Model_Queries-Map* .Σε αυτό το σημείο είχαμε πει πως εξετάζουμε κάθε γραμμή και αν περιέχει η $2^{η}$ στήλη κάποια από τις λέξεις του query τότε και μεταφέρουμε την γραμμή στον Reducer. Έτσι, αν θέλαμε να υπολογίζουμε το βάρος Q_W θα μπορούσαμε πού εύκολα να αθροίσουμε τα τετράγωνα των όρων IDF αφού στα clusters που δημιουργούνται απλά μεταφέρουμε τα δεδομένα μονάχα των όρων του ερωτήματος του χρήστη και όχι το σύνολο των όρων του κάθε site.

Που είναι όμως το λάθος σε αυτό το σκεπτικό; Το σφάλμα σε αυτή την υπόθεση είναι πως σε ένα site μπορεί να βρούμε από κανένα μέχρι όλους τους όρους του query. Αν βρούμε όλους τους όρους τότε έχουμε στην διάθεση μας όλα τα στοιχεία για να υπολογίσουμε το Q_W αν όμως δεν υπάρχουν όλοι οι όροι τότε δεν έχουμε τα IDF όλων των λέξεων άρα δεν γίνεται να υπολογίσομε και σωστά το Q_W. Επομένως για να μην έχουμε τέτοια προβλήματα απλά μεταφέραμε τα ζητούμενα δεδομένα στον δίσκο και υπολογίζαμε το Q_W πριν ξεκινήσει η φάση του Mapping.

4.18.2 Vector_Model_Queries-Reducer

Φτάνοντας στο τελευταίο βήμα αυτό που έχει απομείνει να κάνουμε είναι να βρούμε το εσωτερικό γινόμενο του query με το κείμενο. Για να επιτευχθεί αυτό ουσιαστικά πρέπει να αθροίσουμε τα γινόμενα του IDF του κάθε όρου που συναντήσαμε στο site και ανήκε στο query, με το καθολικό βάρος του site που βρίσκεται στην τελευταία στήλη κάθε γραμμής. Επομένως, για το παράδειγμα 3 στην περίπτωση *http://m.yahoo.com/* περιέχει τους 2 από τους τρεις όρους του Query άρα το εσωτερικό γινόμενο Q*W θα είναι: **16.4733191*23.8354748 + 0. 0.108530253 *23.835474.** Τελικά, για να βρούμε τον βαθμό ομοιότητας με τα ζητούμενα του χρήστη απλά εφαρμόζουμε τον τύπο $Sim = \frac{Q*W}{|Q||W|}$ αντικαθιστώντας στον παρανομαστή το |W| με το καθολικό βάρος και το |Q| με το βάρος του ερωτήματος που υπολογίσαμε πριν την έναρξη της διαδικασίας του Mapping και βρίσκεται στην τελευταία στήλη του αρχείου *QueryFile.txt.* Από τα αποτελέσματα πιο 'κοντινό' προς το query είναι το site που έχει μεγαλύτερο συντελεστή Sim. Επομένως, στην έξοδο του Collector απλά θα διοχετεύσουμε το (Sim, URL).Τα αποτελέσματα αυτής της διαδικασίας μεταφέρονται στο directory **Results//Similarity.**

Για να υπολογίσουμε τώρα τα ζητούμενα γινόμενα χρησιμοποιήσαμε τις βάσεις που δεικτοδοτούσαν τα IDF των όρων αλλά και τα καθολικά βάρη των κειμένων που υπολογίσαμε στο βήμα Doc_Weights_.

ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

Στο συγκεκριμένο κεφάλαιο θα αναφέρουμε τα αποτελέσματα που λάβαμε από τις πειραματικές διαδικασίες στα δύο βασικά τμήματα της εργασίας μας.

Αρχικά, να πούμε πως η διεξαγωγή των πειραμάτων και στις δύο φάσεις πραγματοποιήθηκε πάνω σε **Cluster** των **πέντε** κόμβων που υποστήριζαν το **Hadoop** , σε περιβάλλον ***Ubuntu- Linux.*** Από το σύνολο των πέντε αυτών κόμβων ορίστηκε ένας ως *Master Node* και οι υπόλοιποι *Slaves,*όπως διαφαίνεται παρακάτω. Αυτό σημαίνει πως η αρχικοποίηση ενός *Hadoop Job* πραγματοποιούνταν από τον Master Node, ενώ η επικοινωνία και ο συγχρονισμός όλων των υπόλοιπων κόμβων πραγματοποιούνταν μέσω του Hadoop. Για την εκκίνηση τώρα κάποιας διαδικασίας για την διεξαγωγή πειραμάτων συνδεόμασταν μέσω **SSH Client(*Putty.exe*)** από περιβάλλον **Windows Xp** στον Master Node και μέσω αυτού ξεκινούσαμε τα πειράματα-(***Fetching Process, Vector Space Representation***).

Hadoop's Nodes	Description
Clu26.softnet.tuc.gr	Master Node
Clu25.softnet.tuc.gr	Slave Node
Clu24.softnet.tuc.gr	Slave Node
Clu20.softnet.tuc.gr	Slave Node
Clu19.softnet.tuc.gr	Slave Node

Πίνακας 1: Περιγραφή Κόμβων

1η ΦΑΣΗ: FETCHING && ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΔΙΕΥΘΥΝΣΕΩΝ

Η πρώτη φάση της πειραματική διαδικασίας έχουμε ήδη πει πως ουσιαστικά απαρτίζεται από **τέσσερις** *Hadoop Jobs :* ***Fetching Process, WordCount Process, Remove_Duplicates Process ,Group_Urls Process.*** Οι παραπάνω διαδικασίες πραγματοποιούνται κατά διαδοχικό τρόπο η μία μετά τη άλλη, επομένως τα σημεία που έπρεπε να εξετάσουμε σε κάθε διαδικασία είναι :**Πρώτον**, τα μεγέθη των αρχείων εισόδου/εξόδου και **δεύτερον** το χρονικό διάστημα από την αρχή μέχρι το τέλος κάθε διαδικασίας . Σε δεύτερη ανάλυση θα δούμε πώς για τις ίδιες διαδικασίες επηρεάζεται ο χρόνος αποπεράτωσης κάθε βήματος ανάλογα τον αριθμό των κόμβων του cluster.

5.1 Fetching Ιστοσελίδων

Πρόκειται για το πρώτο από τα τέσσερα βήματα της πρώτης φάσης των πειραμάτων που διεξήγαμε κατά το οποίο πραγματοποιούνταν η προσέγγιση των διευθύνσεων . Στόχος αυτής της εργασίας σε αυτό το βήμα, ήταν να προσεγγίσουμε και να απομονώσουμε τα δεδομένα από **500.000 sites** συνολικά. Επομένως, δεδομένου πώς σε κάθε επανάληψη του βήματος δίναμε ως είσοδο **10.000 διευθύνσεις** προς επίσκεψη εκτελέσαμε το συγκεκριμένο βήμα **50 φορές** .

Πριν παρουσιάσουμε λοιπόν τα μεγέθη που μας ενδιαφέρουν, ας θυμηθούμε ποιες διαδικασίες πραγματοποιούνταν σε αυτό το βήμα καθώς και την αρχικοποίηση ορισμένων παραμέτρων που σχετίζονταν με αυτές.

Αρχικά, να πούμε πώς σε αυτό το σημείο που γίνεται η συλλογή των δεδομένων πραγματοποιούνται κατά βάση 3 διαφορετικά πράγματα: **Πρώτον**, δημιουργούνται TCP συνδέσεις με τα sites όπου και κατεβάζουμε το περιεχόμενο, **δεύτερον** εξετάζεται το περιεχόμενο των sites για την παραγωγή νέων διευθύνσεων και **τρίτον**, κατασκευάζονται αρχεία με τις νέες διευθύνσεις αλλά και για την ενημέρωση της βάσης. Στην συνέχεια , το εσωκλειόμενο κείμενο διοχετεύεται στον *reducer.* Επομένως, για αυτές τις τρείς βασικές λειτουργίες πρέπει να αρχικοποιηθούν οι εξής παράμετροι:

Fetching Process	Parameters	Values
Tcp Connection	Connection Timeout	5000 msec
	Read Timeout	5000* msec
Url Scheduling	MaxServerListSize	6.000
Mappers Number	NumMaptasks	30
Reducers Number	NumReduceTasks	10

Πίνακας 2: Αρχικοποίηση Μεταβλητών

- ***Tcp Connection:*** Δημιουργία TCP σύνδεσης με την εκάστοτε διεύθυνση.
  - ***Connection Timeout:*** Τερματισμός σύνδεσης μετά το πέρας 5sec.
  - ***Read Timeout:*** Τερματισμός σύνδεσης μετά το πέρας 5 sec για sites που δεν έχουν ορίσει κάποια τιμή για την δεδομένη παράμετρο. Διαφορετικά δεχόμαστε ως τιμή την παράμετρο που ορίζει κάθε Host.

- ***Url Scheduling:*** Εισαγωγή νέων διευθύνσεων σε ουρά.

  - ***MaxServerListSize:*** Μέγιστη τιμή ουράς μέχρι την μεταφορά των νέων διευθύνσεων σε αρχείο.

- ***Mappers Number:*** Ο συνολικός αριθμός των Maps που θα επιδράσουν στο σύνολο των δεδομένων εισόδου.

  - ***NumMaptasks***: Πειραματικά έχει βρεθεί πως ο συνολικός αριθμός των mappers κυμαίνεται από 10-100 mappers /Node.

- ***Reducers Number:*** Ο συνολικό αριθμός των Reducer που θα επιδράσουν στο σύνολο των δεδομένων εξόδου.

  - ***NumReduceTasks:*** Πειραματικά βέλτιστο θεωρείται : **0.95*Nodes*mapred.tasktracker.tasks.maximum.**[8]

Αφού λοιπόν αρχικοποιήσαμε τις παραμέτρους για το δεδομένο βήμα, επιτελέσαμε 50 επαναλήψεις. Από το σύνολο τον επαναλήψεων αυτών θα δώσουμε τα πειραματικά δεδομένα και στο τέλος θα παρουσιάσουμε τους μέσους όρους των μεγεθών.

Job_Id	Input Size (10.000 URLS)	HDFS Output Size	Execution Time
Job_201002231949_0001	1.77 MB	2.643 GB	43 min, 09 sec
Job_201002231949_0003	1.74 MB	6.106 GB	43 min, 08 sec
Job_201002231949_0005	1.8 MB	6.033 GB	42 min, 49 sec
Job_201002231949_0007	1.7 MB	6.147 GB	39 min, 36 sec
Job_201002231949_0009	1.32 MB	6.174 GB	25 min, 47 sec
Job_201002231949_0011	1.84 MB	4.999 GB	32 min, 54 sec
Job_201002231949_0013	1.59 MB	2.829 GB	31 min, 26sec
Job_201002231949_0015	1.39 MB	0.752 GB	25 min, 27 sec
Job_201002231949_0017	1.59 MB	2.733 GB	31 min, 28 sec
Job_201002231949_0019	1.89 Mb	0.732 GB	25 min, 43 sec
Job_201002231949_0024	1.59 MB	3.715 GB	42 min, 03 sec
Job_201002231949_0026	1.56 MB	3.750 GB	41 min, 52 sec
Job_201002231949_0028	1.64 MB	3.649 GB	40 min ,12 sec
Job_201002231949_0030	1.64 MB	3.652 GB	40 min, 53 sec
Job_201002231949_0032	1.47 MB	3.711 GB	40 min, 44 sec
Job_201002231949_0034	1.74 MB	3.995 GB	34 min, 56 sec

[8] Για το δικό μας Cluster η μέγιστη τιμή έχει οριστεί **2.**

Job_201002231949_0036	1.62 MB	3.860 GB	32 min, 16 sec
Job_201002231949_0038	1.59 MB	3.908 GB	33 min, 23 sec
Job_201002231949_0040	1.46 MB	3.629 GB	35 min, 45 sec
Job_201002231949_0042	1.72 MB	3.658 GB	42 min, 12 sec
Job_201002231949_0044	1.78 MB	2.486 GB	42 min, 06 sec
Job_201002241718_0001	1.89 MB	5.776 GB	45 min, 07 sec
Job_201002241718_0003	1.53 MB	5.816 GB	45 min, 30 sec
Job_201002241718_0005	1.96 MB	5.950 GB	49 min, 03 sec
Job_201002241718_0007	1.75 MB	5.770 GB	47 min, 18 sec
Job_201002241718_0009	1.97 MB	1.233 GB	46 min, 236 sec
Job_201002241718_0011	1.83 MB	5.936 GB	44 min, 46 sec
Job_201002241718_0013	1.95 MB	1.022 GB	35 min, 09 sec
Job_201002241718_0015	1.63 MB	2.528 GB	33 min, 36 sec
Job_201002241718_0019	1.74 MB	5.222 GB	41 min, 15 sec
Job_201002241718_0021	1.57 MB	2.686 GB	33 min, 59 sec
Job_201002241718_0023	1.73 MB	5.219 GB	41 min, 06 sec
Job_201002241718_0025	1.72 MB	5.193 GB	42 min, 057 sec
Job_201002241718_0027	1.76 MB	2.658 GB	33 min, 10 sec
Job_201002241718_0029	1.69 MB	5.310 GB	40 min, 17 sec
Job_201002241718_0031	1.95 MB	5.265 GB	40 min, 43 sec
Job_201002241718_0033	1.69 MB	10.860 GB	49 min, 56 sec
Job_201002241718_0035	1.76 MB	11.404 GB	47 min, 18 sec
Job_201002241718_0037	1.86 MB	4.576 GB	36 min, 22 sec
Job_201002241718_0039	1.84 MB	2.010 GB	32 min ,28 sec
Job_201002241718_0041	1.73 MB	2.009 GB	34 min, 46 sec
Job_201002241718_0043	2.10 MB	2.003 GB	40 min, 244 sec
Job_201002241718_0045	1.85 MB	2.042 GB	41 min, 50 sec
Job_201002190205_0108	1.72 MB	8.300 GB	45 min, 06 sec
Job_201002190205_0120	1.76 MB	8.323 GB	41 min, 10 sec
Job_201002241718_0047	1.75 MB	2.643 GB	41 min, 00 sec
Job_201002231949_0022	1.85 MB	3.446 GB	42 min, 03 sec
Job_201002241718_0055	1.72 MB	3.442 GB	41 min, 58 sec
Job_201002241718_0057	2.43 MB	3.443 GB	41 min, 13 sec
Job_201002231949_0001	1.77 MB	2.643 GB	43 min, 09 sec
Job_201002231949_0003	1.74 MB	6.106 GB	43 min, 08 sec
Job_201002231949_0005	1.8 MB	6.033 GB	42 min, 49 sec
Job_201002231949_0007	1.7 MB	6.147 GB	39 min, 36 sec
Average	1.65 MB	4.52 GB	39 min 06 sec
Sum	94.71 MB	209.226 GB	31h 9 min

Πίνακας 3: Πειραματικά Αποτελέσματα Fetching Process

Από την παραπάνω διαδικασία λοιπόν συνολικά συγκεντρώσαμε **511.566** σελίδες οι οποίες και αποθηκεύτηκαν στο τοπικό directory **/Texts** του κάθε κόμβου με αναλογίες σε κάθε κόμβο όπως φαίνεται παρακάτω:

Κόμβος	Αριθμός Σελίδων	Μέγεθος Δεδομένων
Clu26.softnet.tuc.gr	**101.258**	**12.306 MB**
Clu25.softnet.tuc.gr	**101.667**	**12.235 MB**
Clu24.softnet.tuc.gr	**99.310**	**12.306 MB**
Clu20.softnet.tuc.gr	**106.463**	**12.562 MB**
Clu19.softnet.tuc.gr	**102.808**	**12.668 MB**
Σύνολο	**511.566**	**62.077 MB**

Πίνακας 4: ***Αρχεία Που Συγκεντρώσαμε Ανά Κόμβο***

5.1.1 ΣΧΟΛΙΑΣΜΟΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ : Fetching Process

Από τους παραπάνω πίνακες προκύπτουν δύο βασικά σημεία προς σχολιασμό:

ΣΧΕΣΗ ΔΕΔΟΜΕΝΩΝ ΕΙΣΟΔΟΥ-ΕΞΟΔΟΥ:

Από τα παραπάνω στοιχεία το πρώτο ερώτημα που προκύπτει είναι η μεγάλη διαφορά ανάμεσα στα δεδομένα εξόδου και εισόδου. Κάτι τέτοιο όμως είναι αναμενόμενο αφού έχουμε ήδη αναλύσει στο αντίστοιχο κεφάλαιο πώς σε αυτό το βήμα δίναμε ως είσοδο τις διευθύνσεις που θα επισκεπτόμασταν ενώ στην έξοδο καταχωρούσαμε το επεξεργασμένο εσωκλειόμενο κείμενο που απομονώναμε σε κάθε site.

Από την άλλη αν συγκρίνουμε λίγο καλύτερα τους *Πίνακας 3* και *Πίνακας* 4 πιθανώς να δημιουργηθεί η απορία πως από αρχεία της τάξης των **MB** καταλήξαμε να πάρουμε δεδομένα των **209.226 GB** συνολικά. Και πάλι , σε αυτό το ερώτημα η απάντηση στηρίζεται στον τρόπο με τον οποίο θέλαμε να καταχωρίσουμε τα δεδομένα μας. Για την ακρίβεια, στην αντίστοιχη παράγραφο [48] έχουμε εξηγήσει πώς η μορφή με την οποία επιδιώξαμε να μεταφέρουμε τα στοιχεία στον HDFS ήταν ***URL/t word.*** Επομένως, για να αντιπροσωπεύσουμε όλους τους όρους που συναντάμε σε κάθε site είναι λογικό να δημιουργήσουμε αρχεία τέτοιων μεγεθών ,ιδιαίτερα για διευθύνσεις έχουν μεγάλο εσωτερικό περιεχόμενο.

```
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      ash
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      wednesd
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      shift
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      understand
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      lent
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      rev
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      lawr
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      mick
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      post
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      12904
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      ash
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      wednesd
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      cart
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      titl
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      000
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      shop
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      opt
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      samp
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      book
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      serv
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      book
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      franciscan
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      libr
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      aud
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      dvd
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      vh
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      parish
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      handout
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      espaol
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      gift
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      magaz
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      subscript
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      mar
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      sit
http://Catalog.AmericanCatholic.org/product.asp?prodid=C0204      direct
```

Δεδομένα Εξόδου Φάσης Fetching

ΧΡΟΝΟΣ ΕΚΤΕΛΕΣΗΣ:

Δεύτερο σημείο που πρέπει να θίξουμε είναι ο χρόνος που απαιτείται για να ολοκληρωθεί μία επανάληψη του *Fetching*. Από τα αποτελέσματα, είδαμε πώς ο μέσος χρόνος είναι κοντά στα ***39 min*** οπότε για να δούμε πώς αναλύεται ένα τέτοιο μέγεθος πρέπει να αναλογιστούμε το σύνολο των εργασιών σε αυτό το κομμάτι αλλά και το μέγεθος του cluster.

Οσο αφορά το πρώτο σκέλος, όπως έχουμε αναφέρει κατά την διαδικασία του fetching επιτελείται το άνοιγμα των TCP συνδέσεων, η απομόνωση του περιεχομένου αλλά και η εξαγωγή των νέων διευθύνσεων. Την διαδικασία του παραλληλισμού την αναλαμβάνει το Hadoop δημιουργώντας ταυτόχρονα συνδέσεις με τα sites και επιτελώντας τα παραπάνω βήματα ταυτόχρονα σε πέντε διαφορετικούς κόμβους. Προφανώς , η χρονική πολυπλοκότητα αυτών των βημάτων σε ένα σύνολο *10.000* στοιχείων είναι πολύ μικρότερη από ένα κλασσικό μοντέλο που θα επιτελούσε αυτά τα βήματα σε διαδικασιακό προγραμματισμό. Μάλιστα, από πειράματα που επιτελέσαμε ,για την επικοινωνία και απομόνωση 10.000 διευθύνσεων σε κλασσικό προγραμματισμό χρήστηκαν πάνω από 1.5 ώρες.

Από το παραπάνω στοιχείο και μόνο βλέπουμε την υπεροχή του Hadoop αφού μας προσφέρει την δυνατότητα να παραλληλίσουμε διαδικασίες δίχως να χρειαστεί να υλοποιήσουμε κώδικα πολυνηματικής μορφής.

Είναι όμως το βέλτιστο μοντέλο αυτό που σχεδιάσαμε βάσει του Hadoop; Σε αυτό το ερώτημα πρέπει αν λάβουμε υπ' όψη μας πρώτον το μέγεθος του cluster –

τους διαθέσιμους κόμβους –αλλά και τον συνολικό αριθμό των cluster. Ειδικότερα, σε όλα τα πειράματα που διεξήγαμε καθοριστικό ρόλο για την ταχύτητα των διαδικασιών έπαιξε ο αριθμός των κόμβων αφού όσο μεγαλύτερος είναι αυτός τόσοι πιο πολλοί mappers επιδρούν στα δεδομένα εισόδου άρα μειώνεται ο χρόνος επεξεργασίας.

Επιπρόσθετα, καθοριστικό ρόλο για την αποδοτικότητα του συστήματος έπαιξε και ο αριθμός των χρηστών που έτρεχαν διεργασίες πάνω στον cluster. Την στιγμή λοιπόν που έτρεχαν πάνω από ένας χρήστες στο σύστημα ο χρόνος εκτέλεσης αυξανόταν σημαντικά αφού έπρεπε να διαμοιραστούν οι διεργασίες που επιτελούνταν ανά χρήστη. Επομένως, χρόνοι που ξεπερνούν αρκετά το μέσο όρο αποδίδονται κυρίως σε αυτό τον παράγοντα.

Στο δικό μας cluster τώρα, έχοντας στην κατοχή 5 κόμβους και δεδομένου πως κάθε κόμβος δημιουργεί 2 Mappers, συνολικά κάθε χρονική στιγμή επιδρούσαν 10 Mappers ταυτόχρονα .Καθένας Mapper τώρα φροντίζει και δημιουργεί κάποια νήματα τα οποία επιτελούν τις διαδικασίες που ορίζονται στους Mappers και στους Reducers. Κατ' αυτόν τον τρόπο πραγματοποιούνταν οι παράλληλες συνδέσεις και οι υπόλοιπες διαδικασίες που περιγράψαμε.

Προφανώς λοιπόν για ένα cluster 5 κόμβων οι δυνατότητες που προσφέρονται είναι εξαιρετικά καλύτερες από ένα κλασσικό μοντέλο αλλά ουσιαστικά περιορίζονταν σημαντικά από τον διαθέσιμο αριθμό κόμβων. Στην πράξη ,εταιρείες που έχουν να κάνουν με μεγάλο όγκο δεδομένων χρησιμοποιούν κόμβους της τάξης των εκατοντάδων-χιλιάδων εκμηδενίζοντας τους χρόνους επεξεργασίας. Επομένως, αν στην δική μας περίπτωση θέλαμε να αυξήσουμε τον αριθμό των ταυτόχρονων Tcp συνέσεων θα έπρεπε να αυξήσουμε αντίστοιχα και τον αριθμό τον nodes του διαθέσιμου cluster και έτσι να μειώσουμε τον μέσο χρόνο εκτέλεσης του βήματος Fetching.

Βέβαια αυτό που πρέπει να τονίσουμε είναι πώς η πλειοψηφία του μέσου χρόνου εκτέλεσης δεν ανάγονταν στην δημιουργία των παράλληλων συνδέσεων και αυτό γιατί από δοκιμές που κάναμε δίχως να πραγματοποιούμε Reducing παρατηρήσαμε πώς η επίτευξη συνδέσεων και η απομόνωση του περιεχομένου διαρκούσε λίγα μόλις λεπτά- περίπου **10** για είσοδο 10.000 διευθύνσεων-. Επομένως, αυτό που διαφαίνεται είναι πώς η πλειοψηφία του εκτελέσιμου χρόνου αναλογεί κατά βάση στην μεταφορά των δεδομένων –περιεχόμενο sites-από την μνήμη στον δίσκο και στο sorting μετά την mapping phase.

Και γιατί δίνουμε έμφαση σε αυτό το σημείο; Γιατί η ανάλυση και επεξεργασία του περιεχομένου που γίνεται στην φάση του Reduce αναλογεί κατά 2-3 λεπτά του συνολικού χρόνου αφού κατά τις πειραματικές διαδικασίες παρατηρήθηκε πώς μετά την αποπεράτωση του Map η διαδικασία του Reduce διαρκούσε περίπου τόσο. Από αυτά λοιπόν μαρτυρείται η σημαντικότητα και τα πλεονεκτήματα

που προσφέρονται από την αρχιτεκτονική του Hadoop αφού μπορεί και επεξεργάζεται αρχεία της τάξης των GB μέσα σε λίγα μόλις λεπτά.

5.2 ΚΑΤΑΜΕΤΡΗΣΣΗ ΕΜΦΑΝΙΣΗΣ ΟΡΩΝ –*WordCount*

Δεύτερο βήμα της πρώτης φάσης είναι η καταμέτρηση εμφάνισης των όρων ανά site ώστε να μπορέσουμε στην συνέχεια να υπολογίσουμε τον IDF κάθε όρου σε όλο το σύνολο των διευθύνσεων. Εφόσον, το βήμα αυτό ήταν ακόλουθο της διαδικασίας του Fetching , επιτελέσαμε την προκείμενη διαδικασία **50 φορές** συνολικά με είσοδο κάθε φορά τα αποτελέσματα που εξάγαμε σε κάθε επανάληψη από την διαδικασία του Fetching. Έτσι δίναμε ως είσοδο τα δεδομένα από το directory **Out//Page_Output** και τα αποτελέσματα από αυτήν την φάση τοποθετούνταν στο HDFS directory **Out//Output.**

Παρακάτω θα δώσουμε τα πειραματικά δεδομένα για το σύνολο των επαναλήψεων-χρόνους εκτέλεσης ,μεγέθη εισόδου και εξόδου- αλλά και τους μέσους όρους των μεγεθών αυτών.Να σημειωθεί πώς για όλες τις επαναλήψεις που επιτελέστηκαν ορίστηκε ο αριθμός των **Mappers ,Reducers** ως **30,10** αντίστοιχα. Η λογική που ακολουθήθηκε για την επιλογή των αριθμών ήταν η ίδια όπως περιγράφηκε παραπάνω, βλ σελ67.

Job_Id	Input Size	Output Size	Execution Time
Job_201002231949_0002	2.643 GB	0.543 GB	1min 08 sec
Job_201002231949_0004	6.106 GB	1.275 GB	3min 28 sec
Job_201002231949_0006	6.033 GB	1.242 GB	3 min 13 sec
Job_201002231949_0008	6.148 GB	1.244 GB	2min 53 sec
Job_201002231949_0010	6.174 GB	1.270 GB	2min 48 sec
Job_201002231949_0012	4.999 GB	1.049 GB	2min 17 sec
Job_201002231949_0014	2.829 GB	0.752 GB	1min 13 sec
Job_201002231949_0016	0.752 GB	0.732 GB	1min 01 sec
Job_201002231949_0018	2.733 GB	0.732 GB	1min 12 sec
Job_201002231949_0020	0.732 GB	0.724 GB	1min 10 sec
Job_201002231949_0025	3.715 GB	1.052 GB	1min 46 sec
Job_201002231949_0027	3.750 GB	1.015 GB	1min 46 sec
Job_201002231949_0029	3.649 GB	0.988 GB	1min 43 sec
Job_201002231949_0031	3.652 GB	0.922 GB	1min 43 sec
Job_201002231949_0033	3.711 GB	0.987 GB	1min 41 sec
Job_201002231949_0035	3.995 GB	0.921 GB	1min 33 sec
Job_201002231949_0037	3.860 GB	0.890 GB	1min 37 sec
Job_201002231949_0039	3.908 GB	0.900 GB	1min 40 sec
Job_201002231949_0041	3.629 GB	0.843 GB	2min 06 sec
Job_201002231949_0043	3.658 GB	0.845 GB	1min 41 sec
Job_201002231949_0045	2.486 GB	0.570 GB	1min 22 sec
Job_201002241718_0002	5.776 GB	1.223 GB	1min 46 sec

Job_201002241718_0004	5.816 GB	1.232 GB	2min 07 sec
Job_201002241718_0006	5.950 GB	1.258 GB	1min 56 sec
Job_201002241718_0008	5.770 GB	1.230 GB	2min 02 sec
Job_201002241718_0010	1.233 GB	0.382 GB	2min 36 sec
Job_201002241718_0012	5.936 GB	1.260 GB	1min 45 sec
Job_201002241718_0014	1.022 GB	0.353 GB	1min 43 sec
Job_201002241718_0016	2.528 GB	0.580 GB	3min 29 sec
Job_201002241718_0020	5.222 GB	0.885 GB	2min 05 sec
Job_201002241718_0022	2.686 GB	0.794 GB	1min 41 sec
Job_201002241718_0024	5.219 GB	0.898 GB	2min 59 sec
Job_201002241718_0026	5.193 GB	0.878 GB	2min 53 sec
Job_201002241718_0028	2.658 GB	0.768 GB	1min 31 sec
Job_201002241718_0030	5.310 GB	0.889 GB	2min 42 sec
Job_201002241718_0032	5.265 GB	0.893 GB	3min 03 sec
Job_201002241718_0034	10.860 GB	2.094 GB	5min 29 sec
Job_201002241718_0036	11.404 GB	2.240 GB	6min 38 sec
Job_201002241718_0038	4.576 GB	1.251 GB	2min 32 sec
Job_201002241718_0040	2.010 GB	0.877 GB	1min 09 sec
Job_201002241718_0042	2.009 GB	0.801 GB	1min 21 sec
Job_201002241718_0044	2.003 GB	0.874 GB	2min 28 sec
Job_201002241718_0046	2.042 GB	0.809 GB	2min 13 sec
Job_201002190205_0109	8.301 GB	1.648 GB	4min 44 sec
Job_201002190205_0121	8.323 GB	1.649 GB	4min 35sec
Job_201002241718_0048	2.643 GB	0.786 GB	2min 39 sec
Job_201002231949_0023	3.446 GB	0.923 GB	1min 49 sec
Job_201002241718_0056	3.442 GB	0.856 GB	1min 08 sec
Job_201002241718_0058	3.443 GB	0.73 GB	1min 10 sec
Job_201002231949_0002	2.643 GB	0.543 GB	1min 08 sec
Job_201002231949_0004	6.106 GB	1.275 GB	3min 28 sec
Job_201002231949_0006	6.033 GB	1.242 GB	3min 13 sec
Job_201002231949_0008	6.148 GB	1.244 GB	2min53 sec
Average	4.52 GB	0.99 GB	2min 32 sec
Sum	209.226 GB	48.56 GB	95 min 52 sec

Πίνακας 5: Πειραματικά Δεδομένα WordCount Process

5.2.1 ΣΧΟΛΙΑΣΜΟΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ: *WordCount Process*

Από τα παραπάνω στοιχεία τα σημεία που πρέπει να σχολιάσουμε είναι κυρίως οι μέσοι χρόνος εκτέλεσης της εξεταζόμενης διαδικασίας σε σχέση με το μέγεθος εισόδου καθώς και πώς εξηγείται το μέγεθος των δεδομένων εξόδου στο HDFS σύστημα.

Σχεση Δεδομενων Εισοδου-Εξοδου:

Από μία γρήγορη ματιά στα δεδομένα εισόδου αυτό που παρατηρεί κάποιος είναι πως έχουν τα ίδια μεγέθη με τα δεδομένα εξόδου της πρώτης φάσης του Fetching. Αυτό συμβαίνει γιατί η WordCount διαδικασία είναι συνέχεια τη Fetching παίρνοντας τα δεδομένα εξόδου της δεύτερης ως είσοδο.

Τώρα όσον αφορά τα δεδομένα εξόδου ,η προφανής σχέση με τα δεδομένα εισόδου είναι πώς έχουν μικρότερο μέγεθος. Αυτό οφείλεται στο γεγονός πως σε αυτό το βήμα αφαιρέσαμε τις διπλοτυπίες των όρων ανά site και μετρήσαμε πόσες εμφανίσεις έχει κάθε όρος σε καθένα Url. Επομένως , περιττή πληροφορία που δεν μας ήταν πλέον χρήσιμη την αγνοούσαμε. Κατ' αυτόν τον τρόπο κάναμε το πρώτο βήμα για τον υπολογισμό των παραμέτρων IDf κάθε όρου και ταυτόχρονα μειώσαμε τον όγκο πληροφορίας.

```
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 93    7
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 abil  5
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 abund 1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 accomplishm 4
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 achief      7
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 adorn 1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 aint  3
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 allaround   1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 andor 2
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 announc     5
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 anonym      1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 anthon      1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 anticip     2
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 anytim      1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 appar 2
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 aur   1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 ballot      1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 baltimor    6
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 bargain     1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 bd    1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 bench 1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 bidder      1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 border      3
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 bosoxdiehard      1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 bostonthat  1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 brekfest    1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 brown 1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 bruc  1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 bug   1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 caliber     3
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 calsg 1
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 campaign    3
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 cap   16
http://38pitches.com/2007/10/30/free-agency-weird/#more-110 carl  2
```

Δεδομένα Εξόδου Φάσης WordCount

ΧΡΟΝΟΣ ΕΚΤΕΛΕΣΗΣ:

Δεύτερο και πιο σημαντικό σημείο σε αυτή την φάση είναι ο χρόνος εκτέλεσης συγκριτικά με το μέγεθος εισόδου. Αυτό που παρατηρούμε είναι πώς αν και δίνονται δεδομένα της τάξης των **GB** στην πλειοψηφία των επαναλήψεων ο μέσος χρόνος επεξεργασίας είναι λίγα μόλις λεπτά και μάλιστα για μία διαδικασία υπολογισμού που υπό κλασσικές μεθόδους η πολυπλοκότητα θα ήταν απαγορευτική. Φανταστείτε, να προσπαθούσαμε από ένα σύνολο αρχείων **209.226 GB** να υπολογίσουμε για κάθε όρο σε κάθε site πόσες φορές εμφανίζεται. Στην πράξη ένα τέτοιο εγχείρημα θα ήταν πολύ δύσκολο και χρονοβόρο και μάλιστα όταν έχουμε να κάνουμε μονάχα με δεδομένα καταχωρημένα σε αρχεία.

Πού οφείλεται όμως αυτή η ευελιξία που μας προσφέρει το Hadoop; Ήδη στο αντίστοιχο κεφάλαιο[13] έχουμε πει τί καινοτομίες εισήγαγε και για ποιο λόγο. Η ουσία αυτών των πραγμάτων είναι το ότι ο φόρτος εργασίας διαμοιράζεται σε όλους τους διαθέσιμους κόμβους και οι οποίοι επιδρούν στα δεδομένα. Αυτό το στοιχείο σε συνδυασμό τον παραλληλισμό που προσφέρουν οι Mapper κάνουν το Hadoop ένα μοναδικό εργαλείο που εκμηδενίζει το πρόβλημα διαχείρισης αρχείων. Αυτό το στοιχείο φάνηκε αμέσως ακόμα και σε ένα cluster σαν το δικό μας που διαθέτει 5 μόλις κόμβους. Προφανώς , για cluster με μεγαλύτερο αριθμό nodes οι χρόνοι αυτοί θα ήταν ακόμα πιο μικροί.

5.3 ΑΦΑΙΡΕΣΗ ΔΙΠΛΟΤΥΠΩΝ ΔΙΕΥΘΥΝΣΕΩΝ–*Remove_Duplicates*

Επόμενο βήμα της πρώτης φάσης της εργασίας μας είναι η αφαίρεση διπλοτύπων διευθύνσεων που έχουμε συλλέξει από την φάση του Fetching των ιστοσελίδων. Τα δεδομένα που θα δοθούν ως είσοδο στο βήμα αυτό προέρχονται από το Directory **/Urls** στο HDFS σύστημα, όπου βρίσκονται τοποθετημένα σε αρχεία οι νέες εξαγόμενες διευθύνσεις , ενώ τα δεδομένα εξόδου τοποθετούνται στο HDFS Directory **/ToVisit**.

Και πάλι όπως και στις δύο προηγούμενες περιπτώσεις επιτελέστηκαν συνολικά 50 επαναλήψεις. Παρακάτω θα δώσουμε την πλειοψηφία των αποτελεσμάτων και αυτό γιατί όπως θα δούμε το συγκεκριμένο βήμα δεν είχε μεγάλο υπολογιστικό φόρτο οπότε και οι τιμές στα δεδομένα εισόδου-εξόδου είναι παρόμοιες σε όλες τις επαναλήψεις. Ομοίως και οι χρόνοι εκτέλεσης.

Job_Id	Input Size	Output Size	Execution Time
Job_201002172330_0029	9,60 MB	4,57 MB	1min 20 sec
job_2010021902505_0114	65,07 MB	24,80 MB	1min 19 sec
job_2010021902505_0118	56,40 MB	23,40 MB	1min 24 sec
job_2010021902505_0149	55,70 MB	23,00 MB	1min 33 sec
job_2010021902505_0138	55,10 MB	21,5 MB	1min 24 sec
job_2010021902505_0165	50,39 MB	19,24 MB	1min 24 sec
job_2010021902505_0126	50,10 MB	20,80MB	1min 23 sec
job_2010021902505_0086	40,70 MB	17,60 MB	1min 20 sec
Job_201002172330_0059	30,70 MB	13,90 MB	1min 15 sec
job_2010021902505_0110	26,33 MB	11,20 MB	0min 53 sec
Job_201002172330_0017	25,40 MB	12,09 MB	1min 40 sec
job_2010021902505_0082	22,30 MB	9,30 MB	1min 12 sec
job_2010021902505_0122	21,50 MB	9,06 MB	1min 10 sec
Job_201002172330_0037	20,70 MB	9,85 MB	1min 12 sec
job_2010021902505_0094	20,60 MB	8,95 MB	1min 70 sec
job_2010021902505_0157	19,23 MB	8,15 MB	1min 20 sec
job_2010021902505_0090	17,20 MB	7,47 MB	1min 4 sec
job_2010021902505_0134	17,10 MB	6,85 MB	1min 09 sec
job_2010021902505_0142	15,30 MB	7,28 MB	0min 59 sec
job_2010021902505_0153	14,67 MB	6,15 MB	1min 22 sec
job_2010021902505_0161	14,07 MB	5,85 MB	0min 57 sec
job_2010021902505_0078	12,9 MB	6,14 MB	1min 15 sec
job_2010021902505_0130	12,7 MB	5,18 MB	1min 19 sec
Job_201002172330_0033	11,3 MB	5,38 MB	1min 80 sec
Job_201002172330_0045	10,08 MB	5,04 MB	1min 40 sec
Average	27,16 MB	11,75 MB	1min 18 sec
Sum	1357,97 MB	587,52 MB	59min 2 sec

Πίνακας 6: Πειραματικά Δεδομένα Remove_ Duplicates Process

5.3.1 ΣΧΟΛΙΑΣΜΟΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ: *Remove_Dublicates Process*

Από τα παραπάνω δεδομένα προκύπτουν τα εξής στοιχεία προς σχολιασμό.

ΣΧΕΣΗ ΔΕΔΟΜΕΝΩΝ ΕΙΣΟΔΟΥ-ΕΞΟΔΟΥ:

Σε όλες τις παραπάνω περιπτώσεις αυτό που παρατηρούμε είναι πώς τα δεδομένα εισόδου είναι μεγαλύτερου μεγέθους από τα αντίστοιχα της εξόδου. Αυτό συμβαίνει γιατί σε αυτό το βήμα προσπαθούμε να εξαλείψουμε τις διευθύνσεις

που έχουν καταχωρηθεί πάνω από δύο φορές , πράγμα που με την σειρά του συντελεί την μείωση του όγκου των δεδομένων στην έξοδο.

ΧΡΟΝΟΣ ΕΚΤΕΛΕΣΗΣ:

Σε αυτό το βήμα αυτό που παρατηρείται είναι η μικρή χρονική καθυστέρηση σε όλο το σύνολο των επαναλήψεων .Αυτό αποδίδεται σε δύο παράγοντες. Πρώτον , στο μικρό μέγεθος εισόδου προς επεξεργασία και δεύτερον στο γεγονός πώς η υπολογιστική πολυπλοκότητα του δεδομένου βήματος είναι μικρή. Στο αντίστοιχο κεφάλαιο αυτό που εξηγήσαμε ήταν πώς κατά την φάση του mapping απλά δημιουργούσαμε ομάδες βάσει των Urls και κατά το Reduce απλά μεταφέραμε στην έξοδο μονάχα ένα στοιχείο από κάθε ομάδα Urls. Επομένως, αυτό το στοιχείο σε συνδυασμό με τον μικρό όγκο δεδομένων έδωσαν αυτούς τους χρόνους.

5.4 ΟΜΑΔΟΠΟΙΗΣΗ ΔΙΕΥΘΥΝΣΕΩΝ ΣΕ ΑΡΧΕΙΑ: *Group_Urls*

Πρόκειται για το τελευταίο βήμα της πρώτης φάσης των πειραμάτων κατά την οποία ομαδοποιούνται οι διευθύνσεις σε blocks των 10.000 στοιχείων. Η διαδικασία αυτή όπως και οι προηγούμενες επαναλήφθηκε 50 φορές συνολικά δίνοντας ως είσοδο τα δεδομένα από το HDFS directory **//ToVisit** και μεταφέροντας τα νέα αρχεία στο directory **//Future.**

Παρακάτω θα παραθέσουμε την πλειοψηφία των επαναλήψεων αυτού του βήματος καθώς όπως θα δούμε η χρονική καθυστέρηση αλλά και τα μεγέθη εισόδου εξόδου είναι παραπλήσια σε κάθε επανάληψη.

Job_Id	Input Size	Execution Time
Job_201002172330_0038	9.852 MB	7sec
Job_2010021902505_0083	9.321 MB	7 sec
Job_2010021902505_0123	9.056 MB	7 sec
Job_2010021902505_0095	8.953 MB	6 sec
Job_2010021902505_0158	8.156 MB	6 sec
Job_2010021902505_0091	7.471 MB	4 sec
Job_2010021902505_0143	7.286 MB	5 sec
Job_2010021902505_0135	6.858 MB	8 sec
Job_2010021902505_0154	6.157 MB	10 sec
Job_2010021902505_0079	6.146 MB	7 sec
Job_2010021902505_0162	5.855 MB	11 sec
Job_201002172330_0034	5.387 MB	6 sec
Job_2010021902505_0131	5.188 MB	7 sec

Job_201002172330_0046	5.041 MB	7 sec
Job_201002172330_0046	4.575 MB	3 sec
Job_2010021902505_0115	24.894 MB	5 sec
Job_2010021902505_0119	23.434 MB	8 sec
Job_2010021902505_0150	23.045 MB	6 sec
Job_2010021902505_0139	21.514 MB	6 sec
Job_2010021902505_0127	20.823 MB	6 sec
Job_2010021902505_0166	19.247 MB	7 sec
Job_2010021902505_0087	17.641 MB	6 sec
Job_201002172330_0060	13.947 MB	5 sec
Job_201002172330_0045	12.099 MB	6 sec
Job_2010021902505_0111	11.247 MB	4 sec
Average	11.991 MB	6.4 sec
Sum	599.531 MB	320 sec

Πίνακας 6: Πειραματικά Δεδομένα Group_Urls Process

5.4.1 ΣΧΟΛΙΑΣΜΟΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ: *Group_Urls Process*

Το σημείο αυτό που πρέπει να προσέξουμε σε αυτό το βήμα είναι κυρίως το μικρό ποσοστό του χρόνου που απαιτείται για να γίνει η μεταφορά των δεδομένων στην τελική τους μορφή, πράγμα που μαρτυρά την σημασία του παραλληλισμού που προσφέρουν οι Mappers. Στην πράξη, η μεταφορά δεδομένων από ένα αρχείο σε κάποιο άλλο υπό κλασσικές μεθόδους είναι πολύ χρονοβόρα. Σε πειραματισμούς μάλιστα που κάναμε στο παρελθόν παρατηρήσαμε πώς για την απλή μεταφορά δεδομένων ενός αρχείου 20 MB σε κάποιο άλλο χρειάζονταν κοντά στα 20 λεπτά, την στιγμή που στο Hadoop η ίδια διαδικασία πραγματοποιείται μόλις σε λίγα δευτερόλεπτα.

Για άλλη μια φορά διαφαίνονται τα πλεονεκτήματα που προσφέρονται από την ταυτόχρονη επίδραση πολλαπλών κόμβων και αυτό θα φανεί ακόμα καλυτέρα στην συνέχεια των πειραμάτων όπου θα δούμε την αποτελεσματικότητα του Hadoop σε σύνολο δεδομένων εισόδου της τάξης των **50 GB.**

Μετά το πέρας και αυτής της διαδικασίας ,που είναι και το τελευταίο βήμα της φάσης του προγραμματισμού των νέων διευθύνσεων, καταφέραμε και συγκεντρώσαμε συνολικά **5.188.310** διευθύνσεις οι οποίες και καταχωρήθηκαν σε αρχεία. Από εκεί και έπειτα ακολουθεί το βήμα της ενημέρωσης της βάσης όπου και οι νέες διευθύνσεις καταχωρούνται στο σύστημα δεικτοδότησης.

2η ΦΑΣΗ: ΚΑΤΑΣΚΕΥΗ VECTOR SPACE ΑΝΑΠΑΡΑΣΤΑΣΗΣ

Δεύτερη φάση των πειραμάτων μας είναι η κατασκευή του Vector Space Model. Σε αυτό το σημείο επιτελέσαμε δύο διακριτά βήματα μέχρι να υπολογιστούν τα βάρη κάθε κειμένου-site. Οι βήματα για την κατασκευή αφορούν τις κλάσεις ***Word_IDF, Doc_Weights.*** Η διαδικασία κατασκευής του μοντέλου ξεκινά με την ***Word_IDF*** όπου και δίνουμε ως είσοδο το σύνολο των δεδομένων που εξάγαμε από τις 50 επαναλήψεις της πρώτης φάσης.

5.5 ΥΠΟΛΟΓΙΣΜΟΣ IDF: *Word_IDF*

Στο σημείο αυτό ξεκινάμε την υλοποίηση του Vector Space Model με τον υπολογισμό των IDF κάθε όρου. Ως είσοδο δίνονται το σύνολο των αρχείων που κατασκευάστηκαν από την φάση *WordCount* οπότε ως είσοδος δίνεται το HDFS directory **//Out/Output** ενώ τα αποτελέσματα του προκείμενου βήματος τοποθετούνται στο **//Out//IDF.** Εν συνεχεία, τα αρχεία με το σύνολο των όρων και τα αντίστοιχα IDF τοποθετούνται σε τοπικό directory του Master Node όπου και κατασκευάζεται η βάση για την καταχώρηση αυτών των στοιχείων.

Παρατίθενται τα πειραματικά αποτελέσματα:

Job_Id	Input Size	Output Size	Execution Time
Job_201002251518_0001	45.183 GB	49.462 MB	13 min13 sec
Sum	45.183 GB	49.462 MB	13 min13 sec

Πίνακας 7: Πειραματικά Δεδομένα Word_IDF Process

Επίσης, μετά τον υπολογισμό τον Idf κάθε όρου δεικτοδοτήσαμε τοπικά στον Master Node **2.049.652** όρους με τα αντίστοιχα IDf.

```
conx   5.708902
cooc   5.2317805
cooccur      4.754659
cookbak      5.708902
cookbooker   5.708902
cookdriver   5.708902
cookednoth   5.708902
cookersmoker       5.4078717
cookiecream 5.708902
cookieman    5.1068416
cookieshomer       5.708902
cooksbook    5.4078717
coolbut      4.5047817
coolbuster   5.708902
coolhunt     4.805812
coonnel      5.708902
cooper       1.5835506
copengag     5.708902
```

Εικόνα 2 :Στιγμιότυπο Πειραματικών Αποτελεσμάτων Υπολογισμού IDF

5.5.1 ΣΧΟΛΙΑΣΜΟΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ: *Word_IDF Process*

Από τα παραπάνω στοιχεία το κύριο σημείο σχολιασμού είναι ο εκτελέσιμος χρόνος σε σχέση με το μέγεθος εισόδου.

ΧΡΟΝΟΣ ΕΚΤΕΛΕΣΗΣ:

Σε αυτό το βήμα μπορούμε να δείξουμε καθαρά πλέον την υπολογιστική δύναμη του Hadoop. Αυτό που διαφαίνεται από τα αποτελέσματα είναι πώς αν και δίνεται είσοδος δεδομένων της τάξη των 50 GB συνολικά ο χρόνος αποπεράτωσης είναι αρκετά μικρός.

Και όχι μόνο αυτό. Τα δεδομένα εισόδου αν παρατηρήσουμε έχουν μεγαλύτερα μεγέθη σε σχέση με τα αντίστοιχα της εξόδου. Αυτό οφείλεται στο γεγονός πώς σε αυτό το σημείο γίνεται προσπάθεια να απομονωθούν τα στοιχεία από το **//Out/Output** ώστε στο // **Out/IDF** να μεταφερθούν τα δεδομένα μονάχα των όρων σε συνδυασμό με τα αντίστοιχα IDF. Από εκεί και έπειτα μπορούσαμε πολύ εύκολα να κατασκευάσουμε ένα σύστημα δεικτοδότησης στο οποίο θα εισάγαμε τους όρους με τα αντίστοιχα IDF που υπολογίσαμε από αυτό το βήμα.

Τα παραπάνω αποτελέσματα τώρα αποδίδονται όπως ήδη έχουμε εξηγήσει στο γεγονός πως το Hadoop διαμοιράζει την επεξεργασία των δεδομένων σε κάθε κόμβο και στην παραλληλία που προσφέρει κάθε Mapper. Κατά αυτόν τον τρόπο μετά ο πέρας της διαδικασίας καθενός Mapper διοχετεύονται στον Reducer τα ζευγάρια δεδομένων, *βλ σελ 56* , αφού πρώτα ταξινομηθούν. Έτσι ,μόλις κάποιες ομάδες δεδομένων είναι έτοιμες μεταφέρονται στον Reducer όπου και τα τελικά αποτελέσματα καταγράφονται πάνω στο HDFS σύστημα. Αυτές οι διαδι-

κασίες που περιγράψαμε ,δηλαδή το *Mapping, Sorting,Reducing* πραγματοποιούνται σε παράλληλο χρόνο πράγμα που κάνει την επεξεργασία πολύ γρήγορη ακόμα και για ένα τόσο μεγάλο μέγεθος δεδομένων.

Προφανώς ,αν είχαμε στην κατοχή μας περισσότερους κόμβους η κάθε εργασία πάνω στο Hadoop θα εκτελούνταν ακόμα πιο γρήγορα αφού θα είχαμε μεγαλύτερο καταμερισμό των διεργασιών.

5.6 ΥΠΟΛΟΓΙΣΜΟΣ ΒΑΡΩΝ ΚΕΙΜΕΝΩΝ: *Doc_Weights*

Τελευταίο βήμα στην κατασκευή της Vector Space αναπαράστασης των κειμένων είναι ο υπολογισμός των βαρών κάθε κειμένου. Η διαδικασία αυτή πραγματοποιήθηκε δίνοντας ως είσοδο τα αποτελέσματα από το προηγούμενο βήμα *WordCount*, από το HDFS Directory **/Out//Output.** Τα τελικά δεδομένα εξόδου τοποθετούνται στο **/Results//Weights.**

Παρατίθενται παρακάτω τα πειραματικά αποτελέσματα του βήματος:

Job_Id	Input Size	Output Size	Execution Time
Job_201002262118_0004	45.183 GB	60.132 MB	69 min52 sec
Sum	45.183 GB	60.132 MB	69min 52 sec

Πίνακας 8: Πειραματικά Δεδομένα Doc_Weights

```
http://4sq.com/94CHP1   53.26828333788902
http://800ceoread.com/book/show/9781583333334-Play    42.636050047722236
http://800ceoread.com/book/show/9781604190175-Where_Keynes_Went_Wrong   31.835308659433572
http://800ceoread.com/book/show/9781608320110-The_Business_of_Changing_Lives  31.723409256008704
http://CleanEnergyJobs.mobi   6.3977888102184375
http://Fantasticsmag.com      48.49565057817041
http://HealthFactsAndFears.com      87.43518805020665
http://Pittsburgh.rivals.com  133.28606746862178
http://WWW.striketwo.net      26.221831985127395
http://a2docs.org/doc/146/    22.29982023294975
http://aWorldConnected.org    33.211658829792924
http://aaminahhernandez.wordpress.com/2009/12/03/biting-tongue/   85.46177702359839
http://abagond.wordpress.com/2009/11/13/the-white-lens/     164.89446198629594
http://abclocal.go.com/wls/live     95.83834452477777
```

Εικόνα 3 :Στιγμιότυπο Πειραματικών Αποτελεσμάτων Υπολογισμού Βαρών

5.6.1 ΣΧΟΛΙΑΣΜΟΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ: *Doc_Weights Process*

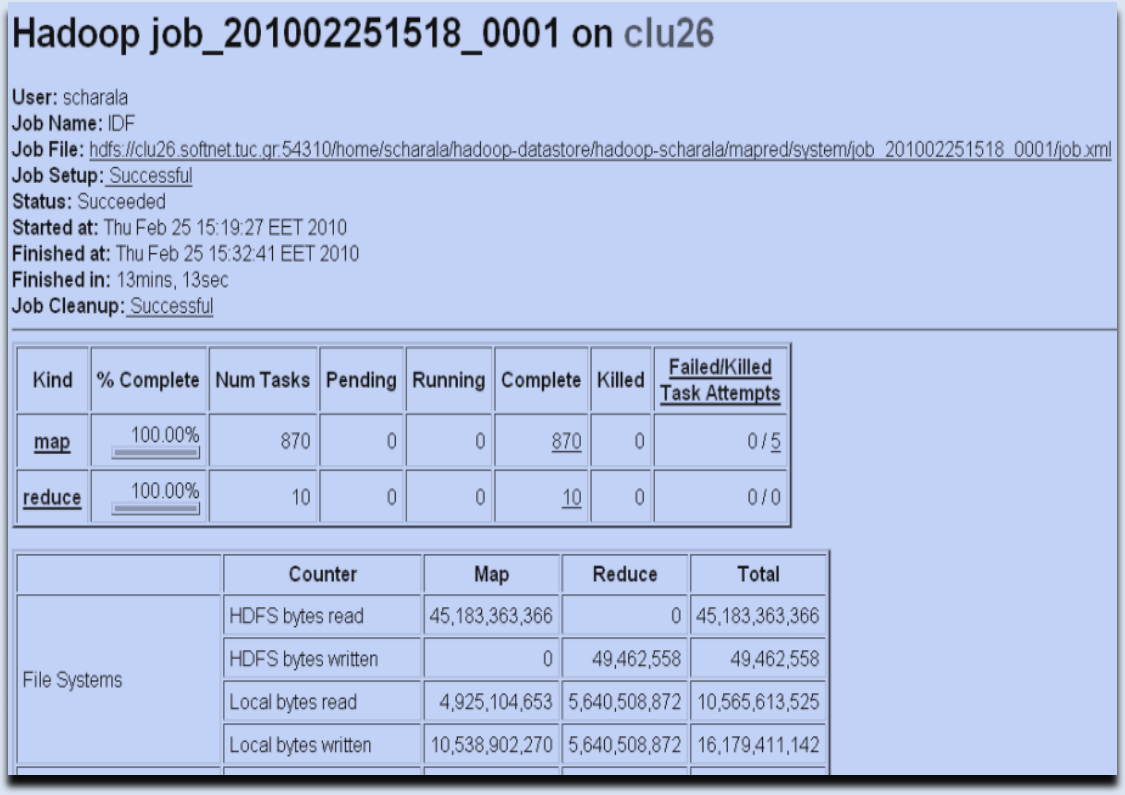
Από τα εξαγόμενα στοιχεία προκύπτουν τα παρακάτω σημεία προς σχολιασμό.

ΧΡΟΝΟΣ ΕΚΤΕΛΕΣΗΣ:

Ως προς τον χρόνο εκτέλεσης, αυτό που πρέπει να αναλογιστούμε είναι πρώτον το μέγεθος εισόδου προς εξέταση αλλά και το μέγεθος των δεδομένων που διοχετεύονται από τον Mapper προς τον Reducer πριν ξεκινήσει η διαδικασία του Reducing για τον υπολογισμό των βαρών.

Ως προς το πρώτο σκέλος, αυτό που παρατηρούμε συγκρίνοντας τους *Πίνακας 7, Πίνακας 8* είναι πως και στα δύο βήματα δίνουμε το ίδιο μέγεθος εισόδου, τα αποτελέσματα από την διαδικασία του *WordCount.* Εντούτοις, παρατηρούμε πως οι χρόνοι εκτέλεσης του βήματος υπολογισμού των βαρών των σελίδων είναι πιο χρονοβόρο σε σχέση με το βήμα *WordCount.* Αυτό αποδίδεται στο δεύτερο σημείο που πρέπει να εξετάσουμε ,δηλαδή το μέγεθος των στοιχείων που διοχετεύουμε από κάθε Mapper προς τον Reducer.

Στην πράξη, το συνολικό μέγεθος των στοιχείων που μεταφέρονται από τον Mapper ,δηλαδή τα clusters που δημιουργούμε με βάσει το *key* κατά το Map, είναι στο σύνολο μεγαλύτερου μεγέθους σε σχέση με τα αντίστοιχα clusters που δημιουργούσαμε κατά την φάση του *Word_IDF.* Αυτό έχει ως αποτέλεσμα τα δεδομένα που μεταφέρονται τοπικά στον δίσκο κάθε κόμβου να είναι μεγαλύτερα σε σχέση με την φάση του *Word_IDF.* Θυμηθείτε, από τα αντίστοιχα κεφάλαια *(σελ 56,σελ 59*) όπου και αναλύσαμε τί κάνουμε σε κάθε βήμα ,στην περίπτωση του Word_IDf τα clusters που φτιάχναμε ήταν βάσει του όρου και ενός αριθμού. Επομένως, βάσει της *Εικόνα 1*, οι τιμές κατά το Mapping που καταγράφονται τοπικά είναι πολύ μικτότερες σε σχέση με την είσοδο.



Hadoop job_201002251518_0001 on clu26

User: scharala
Job Name: IDF
Job File: hdfs://clu26.softnet.tuc.gr:54310/home/scharala/hadoop-datastore/hadoop-scharala/mapred/system/job_201002251518_0001/job.xml
Job Setup: Successful
Status: Succeeded
Started at: Thu Feb 25 15:19:27 EET 2010
Finished at: Thu Feb 25 15:32:41 EET 2010
Finished in: 13mins, 13sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	870	0	0	870	0	0 / 5
reduce	100.00%	10	0	0	10	0	0 / 0

	Counter	Map	Reduce	Total
File Systems	HDFS bytes read	45,183,363,366	0	45,183,363,366
	HDFS bytes written	0	49,462,558	49,462,558
	Local bytes read	4,925,104,653	5,640,508,872	10,565,613,525
	Local bytes written	10,538,902,270	5,640,508,872	16,179,411,142

Εικόνα 1: Αποτελέσματα Word_IDF Process

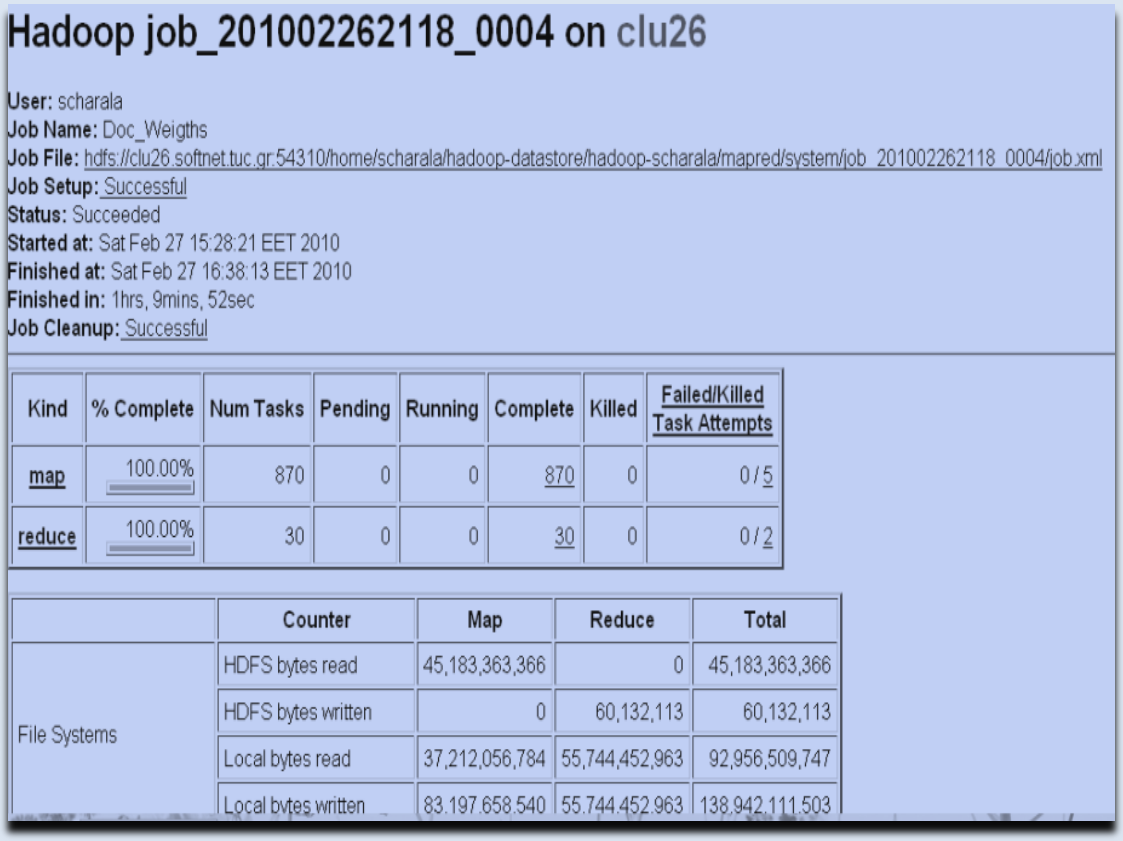
Αντίθετα, σε αυτό το βήμα τα clusters που κατασκευάζονται είναι βάσει του Url, του όρου και της συχνότητας εμφάνισης του όρου στο site. Επομένως ,τα δεδομένα που θα καταγραφούν στον δίσκο αφού πρώτα ταξινομηθούν (πριν το Reducing) θα είναι ποσοτικά πιο πολλά, άρα απαιτείται και περισσότερος χρόνος μέχρι να επιτελεστεί το δεδομένο βήμα.

Να πούμε επίσης πώς key κατά το Map είναι το Url που ποσοστιαία καταναλώνει τον περισσότερο χώρο σε σχέση με τα άλλα δύο στοιχεία ανά γραμμή των αρχείων εισόδου. Αυτό σημαίνει πως θα έχουμε πολλές επαναλήψεις του ίδιου Key στα ζευγάρια (key,values) που κατασκευάζονται ,επομένως είναι λογικό ο αριθμός των δεδομένων που θα καταγραφούν τοπικά να είναι μεγαλύτερος από τα αρχεία εισόδου. Κα' αυτόν τον τρόπο τα δεδομένα τοπικά φτάνουν τα **83.196 GB** και αυτά στην συνέχεια διοχετεύονται προς τους Reducers.

Ακόμα και έτσι όμως και δεδομένου πως ο αριθμός των διαθέσιμων κόμβων είναι μικρός, το Hadoop καταφέρνει αναλογικά με τον όγκο της πληροφορίας εισόδου, να επιτελέσει πολύ γρήγορα την δεδομένη εργασία-(**45.183 GB** εισόδου κατά το Mapping + **83.196 GB** κατά το Reducing). Από τα παρακάτω screenshots που συλλέξαμε και για τις δύο διεργασίες *Word_IDF* και *Doc_Weights* διαφαίνεται το μέγεθος εισόδου αλλά και των τοπικών δεδομένων πριν ξεκινήσει η

εγγραφή των τελικών αποτελεσμάτων κατά το Reduce στο HDFS σύστημα, ό-που και επαληθεύεται η σύγκριση που κάναμε.

Επίσης, καθοριστικό ρόλο στο δεδομένο βήμα δεν παίζει μόνο ο όγκος της επε-ξεργάσιμης πληροφορίας αλλά και οι προσβάσεις στην βάση στην οποία και κρατάμε τα δεδομένα Idf κάθε όρου. Όπως έχουμε πει και κατά την ανάλυση του συγκεκριμένου βήματος , χρησιμοποιήσαμε για τον υπολογισμό των βαρών τα δεδομένα Idf κάθε όρου. Κατά το Reducing όμως του προκείμενου υπολογισμού πρέπει να επιτελέσουμε τόσες προσβάσεις όσος είναι ο συνολικός αριθμός όρων που έχουμε συναντήσει σε όλα τα sites ώστε να βρούμε για καθεμία διεύθυνση την αντιστοιχούμενη βαρύτητα. Προφανώς, η πολυπλοκότητα που εισάγεται σε αυτό το σημείο είναι αρκετά μεγάλη για αυτόν τον λόγο και παρουσιάζεται αυ-τός ο χρόνος εκτέλεσης.



Hadoop job_201002262118_0004 on clu26

User: scharala
Job Name: Doc_Weigths
Job File: hdfs://clu26.softnet.tuc.gr:54310/home/scharala/hadoop-datastore/hadoop-scharala/mapred/system/job_201002262118_0004/job.xml
Job Setup: Successful
Status: Succeeded
Started at: Sat Feb 27 15:28:21 EET 2010
Finished at: Sat Feb 27 16:38:13 EET 2010
Finished in: 1hrs, 9mins, 52sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	870	0	0	870	0	0 / 5
reduce	100.00%	30	0	0	30	0	0 / 2

	Counter	Map	Reduce	Total
File Systems	HDFS bytes read	45,183,363,366	0	45,183,363,366
	HDFS bytes written	0	60,132,113	60,132,113
	Local bytes read	37,212,056,784	55,744,452,963	92,956,509,747
	Local bytes written	83,197,658,540	55,744,452,963	138,942,111,503

Εικόνα 2: Αποτελέσματα Doc_Weights Process

5.7 2η ΦΑΣΗ: ΕΠΕΡΩΡΗΜΑΤΑ ΧΡΗΣΤΗ

Τελευταίο βήμα αυτής της φάσης των πειραμάτων μας είναι ο έλεγχος της ορθότητας της αναπαράστασης κειμένων βάση διανυσμάτων. Γι' αυτόν τον λόγο θέσαμε ορισμένα Queries βάσει ιστοσελίδων που γνωρίζαμε πώς είχαμε προσεγγίσει.

Για να δείξουμε την ορθότητα του μοντέλου που κατασκευάσαμε θέσαμε δύο κατηγορίες ερωτημάτων: Πρώτον, ερωτήματα δίχως την χρήση timestamps αλλά και ερωτήματα βάσει χρονικών περιορισμών του χρήστη. Παρακάτω διαφαίνονται τα ερωτήματα που θέσαμε αλλά και τα αποτελέσματα της αναζήτησης. Στο τέλος ,θα σχολιαστούν συνολικά τα αποτελέσματα των διαδικασιών.

ΠΕΡΙΠΤΩΣΗ 1: ΕΡΩΤΗΜΑ ΔΙΧΩΣ ΧΡΗΣΗ TIMESTAMP

- ***Query: "Vermont Pay Deployment":***

Job Name	Job_Id	Input Size	Output Size	Execution Time
Query	Job_201002262118_0052	45.183 GB	4.331 MB	878 sec
Sorting	Job_201002262118_0053	4.331 MB	4.331 MB	16 sec
Sum				14 min 54 sec

Πίνακας 9: Αποτελέσματα Ερωτήσεων Χρήστη

```
0.18268725  http://www.amazon.com/tag/armed%20forces/lists-guides
0.18777604  http://www.amazon.com/Separated-Duty-United-Shellie-Vandevoorde/product-reviews/0806527277
0.19571424  http://www.amazon.com/365-Deployment-Days-Wifes-Survival/product-reviews/1933538945
0.22548115  http://blog.tomevslin.com/2009/12/why-we-can-succeed.html
0.22764252  http://blog.tomevslin.com/2010/01/challenges-for-change.html
0.24330805  http://blog.tomevslin.com/2010/01/should-vermont-pay-for-broadband-deployment.html
0.2465234   http://blog.tomevslin.com/2009/05/index.html
0.24791043  http://blog.tomevslin.com/2009/10/what-the-69-million-smart-grid-grant-will-mean-to-vermont.html
0.24957281  http://rodonline.typepad.com/rodonline/vermont/
0.2499217   http://blog.tomevslin.com/current_affairs/
0.24994339  http://blog.tomevslin.com/2009/12/vermont-yankee---is-the-price-right.html
0.2545467   http://www.bilerico.com/2009/04/vermont_rules.php
0.26019815  http://blog.tomevslin.com/telecommunications/
0.27278054  http://badevan.com/2009/03/vermont-interrupts-my-vacation/
0.27324572  http://badevan.com/2009/03/vermont-interrupts-my-vacation/#comments
0.27902478  http://blog.tomevslin.com/2010/01/index.html
0.28564644  http://blog.tomevslin.com/2009/08/index.html
0.28895092  http://blog.tomevslin.com/2009/10/index.html
0.29295564  http://blog.tomevslin.com/2009/11/index.html
0.29585344  http://blog.tomevslin.com/2009/06/index.html
0.30204442  http://blog.tomevslin.com/energy/
0.30294552  http://blog.tomevslin.com/estate/
0.3130376   http://feedproxy.google.com/FractalsOfChange
0.3358718   http://shelterbox.org/deployments.php
0.34007788  http://blog.tomevslin.com/economic-stimulus/
```

Εικόνα 6: Στιγμιότυπο Αποτελέσματων Query «Vermont Pay Deployment»

- ***Query: "Amazon Canon Camera":***

Job Name	Job_Id	Input Size	Output Size	Execution Time
Query	Job_201002262118_0050	45.183 GB	32.225MB	1208 sec
Sorting	Job_201002262118_0051	32.225MB	32.225MB	18 sec
Sum				20 min,26 sec

```
0.16292234  http://www.bizrate.com/digital-cameras/stores--datavision-computer-video/products__att296935--28517-.html
0.16554183  http://www.amazon.de/Druckpatronen-IP4200-Original-Canon/lm/R1PJUOP3IYSPOA
0.16591927  http://www.amazon.com/tag/camera%20lenses/lists-guides
0.16636498  http://www.amazon.com/tag/canon%205d
0.16804315  http://www.amazon.com/gp/pdp/profile/AUB1AN3QYEBN7
0.16993165  http://digitalcameras.bizrate.com/products__keyword--canon+digital+slr+camera.html
0.17106266  http://www.amazon.com/Get-Started-Rebel-450D-Kiss/lm/R3V8ZS6QMA3QN1
0.1746219   http://www.amazon.com/gp/pdp/profile/A3OSO5A2SCTRJY
0.17980248  http://www.bizrate.com/printers/canon-printers/
0.18196447  http://www.amazon.com/gp/pdp/profile/A1O4PRTK97ELDE
0.18379323  http://amapedia.amazon.com/edit/asin=B0007WK8LM
0.18420136  http://www.bizrate.com/digital-cameras/powershot-canon/
0.18968615  http://www.bizrate.com/digital-cameras/canon-rebel-xs/
0.19000228  http://www.bizrate.com/printers/canon/
0.19108202  http://www.amazon.de/tag/is/products
0.19327775  http://www.amazon.com/tag/50d/products
0.19907562  http://www.amazon.com/tag/40d/products
0.20035857  http://www.amazon.com/gp/pdp/profile/A2ZK3DVSXFH3VA
0.20242235  http://www.amazon.com/rss/tag/photography/popular
0.20421761  http://www.bizrate.com/digital-cameras/canon-a470/
0.21796173  http://www.amazon.com/tag/canon%20slr/lists-guides
```

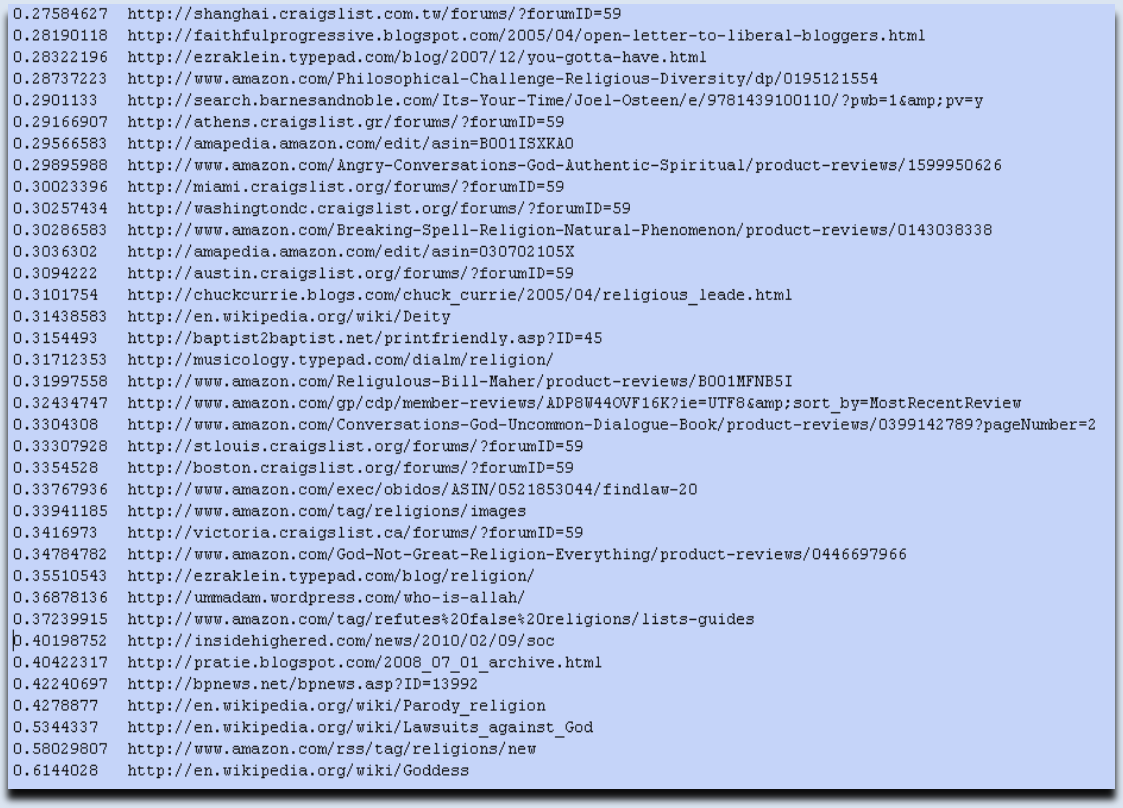
Εικόνα 7: Στιγμιότυπο Αποτελέσματων Query «Amazon Canon Cammera»

ΠΕΡΙΠΤΩΣΗ 2: ΕΡΩΤΗΜΑ ΜΕ ΧΡΗΣΗ TIMESTAMP

- ***Query: "God Religion" Retrieve After Date: 1/2/2010:***

Job Name	Job_Id	Input Size	Output Size	Execution Time
Query	Job_201002262118_0039	45.183 GB	2.881 MB	738 sec
Sorting	Job_201002262118_0040	2.881 MB	2.881 MB	16 sec
Sum				12min, 24 sec

Πίνακας 10: Αποτελέσματα Ερωτήσεων Χρήστη



```
0.27584627  http://shanghai.craigslist.com.tw/forums/?forumID=59
0.28190118  http://faithfulprogressive.blogspot.com/2005/04/open-letter-to-liberal-bloggers.html
0.28322196  http://ezraklein.typepad.com/blog/2007/12/you-gotta-have.html
0.28737223  http://www.amazon.com/Philosophical-Challenge-Religious-Diversity/dp/0195121554
0.2901133   http://search.barnesandnoble.com/Its-Your-Time/Joel-Osteen/e/9781439100110/?pwb=1&amp;pv=y
0.29166907  http://athens.craigslist.gr/forums/?forumID=59
0.29566583  http://amapedia.amazon.com/edit/asin=B001ISXKA0
0.29895988  http://www.amazon.com/Angry-Conversations-God-Authentic-Spiritual/product-reviews/1599950626
0.30023396  http://miami.craigslist.org/forums/?forumID=59
0.30257434  http://washingtondc.craigslist.org/forums/?forumID=59
0.30286583  http://www.amazon.com/Breaking-Spell-Religion-Natural-Phenomenon/product-reviews/0143038338
0.3036302   http://amapedia.amazon.com/edit/asin=030702105X
0.3094222   http://austin.craigslist.org/forums/?forumID=59
0.3101754   http://chuckcurrie.blogs.com/chuck_currie/2005/04/religious_leade.html
0.31438583  http://en.wikipedia.org/wiki/Deity
0.3154493   http://baptist2baptist.net/printfriendly.asp?ID=45
0.31712353  http://musicology.typepad.com/dialm/religion/
0.31997558  http://www.amazon.com/Religulous-Bill-Maher/product-reviews/B001MFNB5I
0.32434747  http://www.amazon.com/gp/cdp/member-reviews/ADP8W44OVF16K?ie=UTF8&amp;sort_by=MostRecentReview
0.3304308   http://www.amazon.com/Conversations-God-Uncommon-Dialogue-Book/product-reviews/0399142789?pageNumber=2
0.33307928  http://stlouis.craigslist.org/forums/?forumID=59
0.3354528   http://boston.craigslist.org/forums/?forumID=59
0.33767936  http://www.amazon.com/exec/obidos/ASIN/0521853044/findlaw-20
0.33941185  http://www.amazon.com/tag/religions/images
0.3416973   http://victoria.craigslist.ca/forums/?forumID=59
0.34784782  http://www.amazon.com/God-Not-Great-Religion-Everything/product-reviews/0446697966
0.35510543  http://ezraklein.typepad.com/blog/religion/
0.36878136  http://ummadam.wordpress.com/who-is-allah/
0.37239915  http://www.amazon.com/tag/refutes%20false%20religions/lists-guides
0.40198752  http://insidehighered.com/news/2010/02/09/soc
0.40422317  http://pratie.blogspot.com/2008_07_01_archive.html
0.42240697  http://bpnews.net/bpnews.asp?ID=13992
0.4278877   http://en.wikipedia.org/wiki/Parody_religion
0.5344337   http://en.wikipedia.org/wiki/Lawsuits_against_God
0.58029807  http://www.amazon.com/rss/tag/religions/new
0.6144028   http://en.wikipedia.org/wiki/Goddess
```

Εικόνα 8: Στιγμιότυπο Αποτελέσματων Query «God Religion»

- ***Query: "Yankees Game 2009" Retrieve Before Date: 29/2/2010:***

Job Name	Job_Id	Input Size	Output Size	Execution Time
Query	Job_201002262118_0041	45.183 GB	8.780 MB	945 sec
Sorting	Job_201002262118_0042	8.780 MB	8.780 MB	16 sec
Sum				14 min ,24 sec

Πίνακας 11: Αποτελέσματα Ερωτήσεων Χρήστη

```
0.36889252  http://eephus.blogspot.com
0.375431    http://bats.blogs.nytimes.com/tag/yankees/
0.3770372   http://ballhype.com/mlb/george_steinbrenner_iii/
0.37859404  http://mlb.mlb.com/stats/individual_player_splits.jsp?c_id=nyy&playerID=113028&statType=1
0.38048658  http://www.amazon.com/rss/tag/world%20series/new
0.38926515  http://newyork.yankees.mlb.com/mlb/sweepstakes/y2010/nyy/taxday_sweeps_rules.jsp
0.38986835  http://shop.mlb.com/shop/index.jsp?categoryId=1452360
0.3911144   http://latimesblogs.latimes.com/sports_blog/world-series/
0.3937543   http://bats.blogs.nytimes.com/tag/angels/
0.39610925  http://web.yesnetwork.com/schedule/programs/index.jsp
0.40190428  http://www.behindthebombers.com/horror.htm
0.4071544   http://ballhype.com/mlb/nick_johnson/
0.40774843  http://shop.mlb.com/entry.point?entry=1452360&amp;source=NYY_Gameday&affiliateId=Gameday
0.40776917  http://mlb.mlb.com/stats/individual_player_gamebygamelog.jsp?playerID=282332&statType=2
0.41305465  http://bats.blogs.nytimes.com/tag/phil-hughes/
0.42863145  http://latimesblogs.latimes.com/sports_blog/yankees/
0.4313859   http://mlb.mlb.com/stats/individual_stats_player.jsp?c_id=nyy&playerID=150359
0.4442966   http://ballhype.com/mlb/new_york_yankees/
0.4484463   http://mlb.mlb.com/stats/individual_stats_player.jsp?c_id=nyy&playerID=282332
0.45662975  http://web.yesnetwork.com/teams/player_of_the_game.jsp?oid=36019
0.45978662  http://newyork.yankees.mlb.com/fan_forum/all_time_nine/index.jsp?c_id=nyy
0.4607745   http://mlb.mlb.com/stats/individual_stats_player.jsp?c_id=nyy&playerID=407893
0.4617956   http://nybaseballdigest.com/
0.48338047  http://newyork.yankees.mlb.com/news/article.jsp?ymd=20090911&content_id=6909874&vkey=news_nyy&fext=.jsp&c_id=nyy
0.4978612   http://web.yesnetwork.com/media/video.jsp?topic_id=6504248
0.51623225  http://slidingintohome.blogspot.com/
0.51936895  http://ballhype.com/mlb/yankee_stadium/
0.52127796  http://www.behindthebombers.com/memorials.htm
0.523825    http://riveraveblues.com
0.5250274   http://bats.blogs.nytimes.com/tag/yankee-stadium/
0.5283128   http://mlb.mlb.com/stats/individual_stats_player.jsp?c_id=nyy&playerID=121347
0.53342813  http://www.abctickets.com/sports/baseball/mlb/new-york-yankees/0/city.html
0.57011884  http://newyork.yankees.mlb.com/photogallery/archive.jsp?c_id=nyy
0.575452    http://newyork.yankees.mlb.com/ticketing/index.jsp?c_id=nyy
0.5794144   http://newyork.yankees.mlb.com/news/article.jsp?ymd=20100218&content_id=8092648&vkey=news_nyy&fext=.jsp&c_id=nyy
0.5869683   http://newyork.yankees.mlb.com/media/video.jsp?topic_id=nyy&c_id=nyy
0.5989596   http://newyork.yankees.mlb.com/news/article.jsp?ymd=20100216&content_id=8081562&vkey=news_nyy&fext=.jsp&c_id=nyy
0.602073    http://sports.yahoo.com/mlb/teams/nyy;_ylt=AOwNdbaulYBLc1MBrSuFCLcF
0.6132658   http://sports.yahoo.com/mlb/teams/nyy;_ylt=AOwNdcvC8n9Lq3oARwCFCLcF
```

Εικόνα9:Στιγμιότυπο Αποτελέσματων Query «Yankees Game 2009»

- ***Query: "Murdoch Blocks Google Searchers" Retrieve Between Dates: 1/2/2010-29/02/2010:***

Job Name	Job_Id	Input Size	Output Size	Execution Time
Query	Job_201002262118_0048	45.183 GB	26.243 MB	1321 min
Sorting	Job_201002262118_0049	26.243MB	26.243 MB	18 sec
Sum				22 min, 19 sec

Πίνακας 12: Αποτελέσματα Ερωτήσεων Χρήστη

```
0.38779584  http://wikipedia.org/wiki/Google
0.5399168   http://bit.ly/3dlalG?showallcomments=true#CommentKey:0f209dea-9361-45bf-97d1-eb70ea6ee86b
0.5399168   http://bit.ly/3dlalG?showallcomments=true#CommentKey:ad34e631-d087-4b4b-a325-4575033a0701
0.5399168   http://bit.ly/3dlalG?showallcomments=true#CommentKey:b5f5910b-95cf-4905-9735-e1345981fe09
0.5399169   http://bit.ly/3dlalG#
0.53991693  http://bit.ly/3dlalG?showallcomments=true#CommentKey:34f97dfa-2f68-495a-8c1f-f38a336235f9
0.53991693  http://bit.ly/3dlalG?showallcomments=true#CommentKey:77ca5425-cb0e-4db0-9d30-56f285669dc0
0.53991693  http://bit.ly/3dlalG?showallcomments=true#CommentKey:4fe06322-b9c6-447e-a8e0-4e031160d4af
0.53991705  http://bit.ly/3dlalG?showallcomments=true#CommentKey:19aea720-2d9f-48ff-8588-699b987d0501
0.5399171   http://bit.ly/3dlalG?showallcomments=true#CommentKey:16636210-c025-40b5-99a7-350462cd9ef1
0.5399171   http://bit.ly/3dlalG#global-jobs-1
0.5399202   http://bit.ly/3dlalG?showallcomments=true#end-of-comments
0.5399202   http://bit.ly/3dlalG?showallcomments=true#CommentKey:f35f3e84-dd3a-4b59-b0dd-0b2d12eb640e
0.53992033  http://bit.ly/3dlalG?showallcomments=true#CommentKey:f3a0fdd6-ec15-4dd8-875d-3e81b03103bd
0.54566866  http://bit.ly/3dlalG?showallcomments=true#CommentKey:f1a68aff-fc50-41b6-9182-8980f72c2d18
0.5456688   http://bit.ly/3dlalG?showallcomments=true#CommentKey:7419e722-e4d8-454c-b74b-9f691b0eb2ab
0.5456688   http://bit.ly/3dlalG#web-search-field
0.5456688   http://bit.ly/3dlalG#start-of-comments
0.5456688   http://bit.ly/3dlalG?showallcomments=true#CommentKey:c9362025-c2dd-4d06-b7bd-fc2867a42125
0.54598546  http://bit.ly/3dlalG?showallcomments=true#CommentKey:58a45a31-b309-47c6-8870-5e92180d0e42
0.54598546  http://bit.ly/3dlalG?showallcomments=true#CommentKey:8c8b5c82-d8d3-4c49-b82b-7e3cfb15eab3
0.54598546  http://bit.ly/3dlalG?showallcomments=true#CommentKey:b59da7fc-9dbe-45b4-81d1-5c9e06d3fd7e
0.5459855   http://bit.ly/3dlalG#box
0.5459855   http://bit.ly/3dlalG?showallcomments=true#CommentKey:3e7013b5-7361-4297-9034-f9d95a27502d
0.5459855   http://bit.ly/3dlalG?showallcomments=true#CommentKey:c6bdf7cb-c7b3-49cc-a095-f6da367fa770
0.5459856   http://bit.ly/3dlalG?showallcomments=true#CommentKey:9ca21896-c7b0-4042-a729-f328b5d474db
0.5459857   http://bit.ly/3dlalG?showallcomments=true#CommentKey:f9037fdd-fb02-4436-9755-703244b116d9
0.5459867   http://bit.ly/3dlalG#most-2
0.54598707  http://bit.ly/3dlalG?showallcomments=true#CommentKey:064d9896-838f-4f0e-9a44-3c5ab51a9daa
0.54600114  http://bit.ly/3dlalG?showallcomments=true#CommentKey:c1817aae-a338-45cc-a13b-b517daabf839
0.54600114  http://bit.ly/3dlalG?showallcomments=true#CommentKey:9ae5095b-ee0c-4b71-bdb6-3d102b2f7623
0.54600126  http://bit.ly/3dlalG?showallcomments=true#CommentKey:f202b4ec-6a0d-47d2-98f1-17b48db19045
0.5460013   http://bit.ly/3dlalG?showallcomments=true#CommentKey:a6594654-1eef-4d5b-9343-884c92195ba2
0.5460013   http://bit.ly/3dlalG?showallcomments=true#CommentKey:91077079-eee6-4013-90b5-8ceb36bb0b34
0.5460013   http://bit.ly/3dlalG?showallcomments=true#CommentKey:3b4e0b40-c254-4cc1-af98-2eecc9219c6f
0.5460013   http://bit.ly/3dlalG?showallcomments=true#CommentKey:34b9fdc0-265d-4502-8100-41b1a50f10b5
0.5460014   http://bit.ly/3dlalG?showallcomments=true#CommentKey:72dd5385-8137-46ec-ad5a-c489046c745b
0.58035445  http://www.bilerico.com/2009/11/on_murdoch_and_google_or_hey_rupert_wheres_my_chec.php
0.5920783   http://www.bilerico.com/2009/11/on_murdoch_and_google_or_hey_rupert_wheres_my_chec.php#social
0.5920785   http://www.bilerico.com/2009/11/on_murdoch_and_google_or_hey_rupert_wheres_my_chec.php#comments
```

Εικόνα 10: *Στιγμιότυπο Αποτελεσμάτων Query «Murdoch Blocks Google Searchers»*

5.7.1 ΣΧΟΛΙΑΣΜΟΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ: *Query Process*

Στο τελικό βήμα αυτό που απομένει να σχολιάσουμε είναι ο χρόνος εκτέλεσης ενός Query αλλά και την ακρίβεια του μοντέλου αναπαράστασης του περιεχόμενου.

ΧΡΟΝΟΣ ΕΚΤΕΛΕΣΗΣ:

Πρώτο σημείο που πρέπει να εξετάσουμε είναι ο χρόνος στον οποίο απαντάει η το σύστημα σε ένα τυχαίο query του χρήστη. Από τα πειραματικά δεδομένα παρατηρούμε πως αυτό ανάγεται σε κάποια λεπτά την στιγμή που μεγάλες μηχανές αναζήτησης μπορούν και δίνουν απάντηση μόλις σε msec.

Βασικός παράγοντας για μία τέτοια διαφορά είναι ο αριθμός των διαθέσιμων κόμβων που έχει το σύστημα για να επιτελέσει την δεδομένη εργασία του χρήστη. Στην πράξη μεγάλες εταιρείες όπως *Yahoo* και η *Google* για να ανταποκριθούν στις απαιτήσεις τόσο μεγάλου όγκου δεδομένων έχουν παγκοσμίως χιλιάδες κόμβους που αναλαμβάνουν να επεξεργαστούν τέτοια ερωτήματα.

Και όχι μόνο. Πραγματικά συστήματα που αναλύουν ερωτήματα χρηστών δεν αρκούνται μονάχα σε δεδομένα που έχουν καταχωρημένα τοπικά σε δίσκους αλλά μεταφέρουν στοιχεία και στην μνήμη βάσει διάφορων στρατηγικών(πχ συχνότητα υποβολής Query) ώστε να ανταποκριθούν άμεσα στα διάφορα ερωτήματα. Επιπρόσθετα, πραγματικές μηχανές αναζήτησης για να ανταποκριθούν στις απαιτήσεις του χρήστη κατασκευάζουν δομές για απομόνωση της πληροφορίας σχετικής με το ερώτημα του χρήστη(Inverted Files,HashTables κλπ).

Στην περίπτωσή μας τώρα , σε όλα τα βήματα που έχουμε παρουσιάσει μέχρι στιγμής δεν χρησιμοποιούμε δομές για να μεταφέρουμε στοιχεία από τα αρχεία που έχουμε συλλέξει, αλλά εκμεταλλευόμενοι τα πλεονεκτήματα που μας παρέχει το Hadoop προσομοιώσαμε ένα σύστημα αναζήτησης που έχει τα βασικά χαρακτηριστικά με ένα αντίστοιχο που επεξεργάζεται μαζική πληροφορία. Για να απαντήσουμε λοιπόν σε ένα ερώτημα δεν στηρίζαμε την αναζήτηση σε δομές όπως Inverted Files ή HashTables, αλλά απομονώναμε τους ζητούμενους όρους που είχε ένα query από το σύνολο των δεδομένων κατά το Map. Αυτός είναι ο λόγος που παρουσιάζονται τέτοιοι χρόνοι κατά την υποβολή ερωτημάτων.

Άλλωστε, σκοπός της παρούσας εργασίας δεν ήταν να κατασκευάσουμε ένα σύστημα που θα ανταποκρίνεται σε παραγοντικούς χρόνους κατά τα ερωτήματα. Σκοπός ήταν να κατασκευαστεί ο βασικός κορμός ενός συστήματος αναζήτησης. Για την βελτίωση της αποδοτικότητας του προκείμενου βήματος απαιτείται μονάχα η κατασκευή ενός Inverted Index πάνω στα δεδομένα που έχουμε δημιουργήσει από την WordCount Process.

VECTOR SPACE ΑΝΑΠΑΡΑΣΤΑΣΗ:

Ως προς την αποδοτικότητα του μοντέλου που επιλέξαμε να υλοποιήσουμε, από τα αποτελέσματα των ερωτημάτων που θέσαμε φαίνεται η υπεροχή έναντι του Boolean που υποστηρίζουν όλα τα μεγάλα συστήματα αναζήτησης.

Στα ερωτήματα που θέσαμε παρατηρούμε εκτός από τον βαθμό ομοιότητας που παρέχεται σε σχέση με το υποβαλλόμενο ερώτημα, τα αποτελέσματα τα οποία εξάγονται έχουν άμεση σχέση με το νόημα του ερωτήματος που θέτει ο χρήστης. Αυτό αποδίδεται στο ότι αυτό το μοντέλο καταφέρνει και μοντελοποιεί το καθένα κείμενο στηριζόμενο στον αριθμό εμφάνισης του κάθε όρου μέσα σε αυτό αλλά και στο σύνολο των εξεταζόμενων κειμένων. Αντίθετα, το Boolean μοντέλο για να εξάγει το νόημα και να κάνει την συσχέτιση με το εκάστοτε ερώτημα στηρίζεται μονάχα στην εμφάνιση του κάθε όρου του Query μέσα σε κάποιο κείμενο οπότε και σε ερωτήματα που θέτονται και υπάρχουν πολλοί όροι το συγκεκριμένο μοντέλο είναι αρκετά αδύναμο.

Γιατί όμως δεν προτιμάται η Vector Space αναπαράσταση σε μεγάλες μηχανές αναζήτησης την στιγμή που το Boolean μοντέλο υστερεί σε σχέση με το πρώτο; Ο προφανής λόγος είναι η πολυπλοκότητα της υλοποίησης τόσο κατά τον υπολογισμό των IDF όρων όσο και κατά τον υπολογισμό των βαρών. Και ενώ η διαδικασία υπολογισμού των IDf μπορεί αν πραγματοποιηθεί σε ένα εύλογο χρονικό διάστημα ,ο υπολογισμός των βαρών έχει μεγάλη πολυπλοκότητα αφού πρέπει να εξετάσουμε όλους τους όρους που συναντάμε σε κάθε κείμενο. Αυτό άλλωστε αποδεικνύεται και από τους χρόνους που εξάγαμε κατά την συγκεκριμένη διαδικασία.

5.8 2η ΦΑΣΗ ΠΕΙΡΑΜΑΤΩΝ

Σε αυτό το σημείο θα μετρήσουμε τους χρόνους εκτέλεσης κάθε βήματος μειώνοντας τον διαθέσιμο αριθμό κόμβων. Παρακάτω παρατίθενται τα πειραματικά αποτελέσματα κάθε βήματος δίνοντας κάθε φορά τα ίδια δεδομένα,**10.000 Urls** εισόδου.

Node Num	Fetching	WordCount	Remove_Dublicates	Group_Urls
5	2353 sec	50 sec	69 sec	10 sec
4	2973 sec	68 sec	178 sec	13 sec
3	4501 sec	91 sec	180 sec	16 sec
2	5807 sec	229 sec	452 sec	24 sec
1	8267 sec	395 sec	1057 sec	35 sec

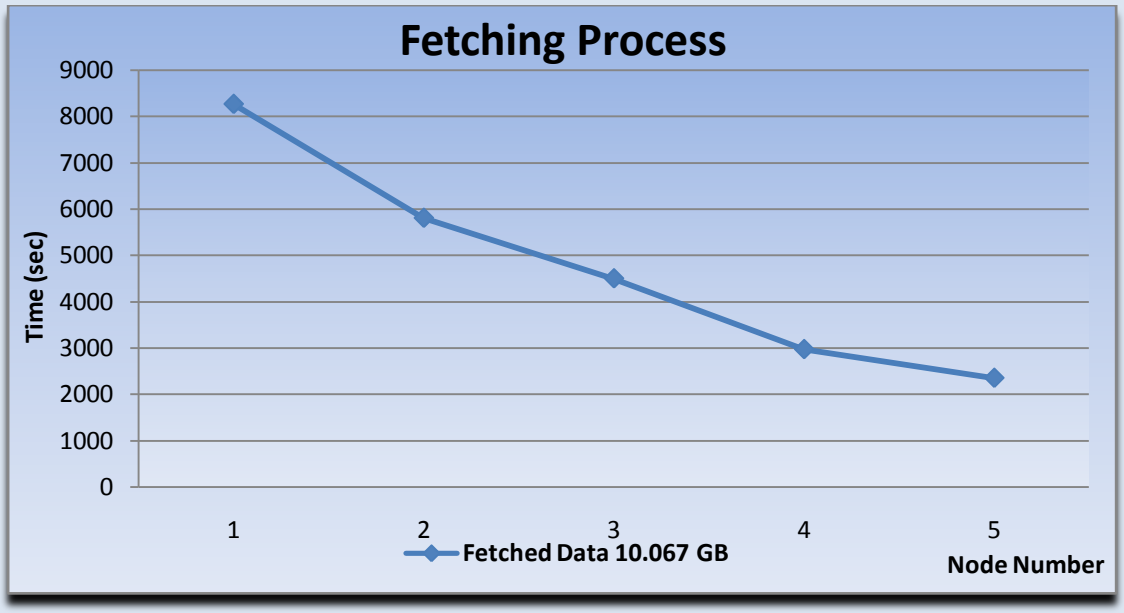
Πίνακας 13: Execution Time Vs Node Number

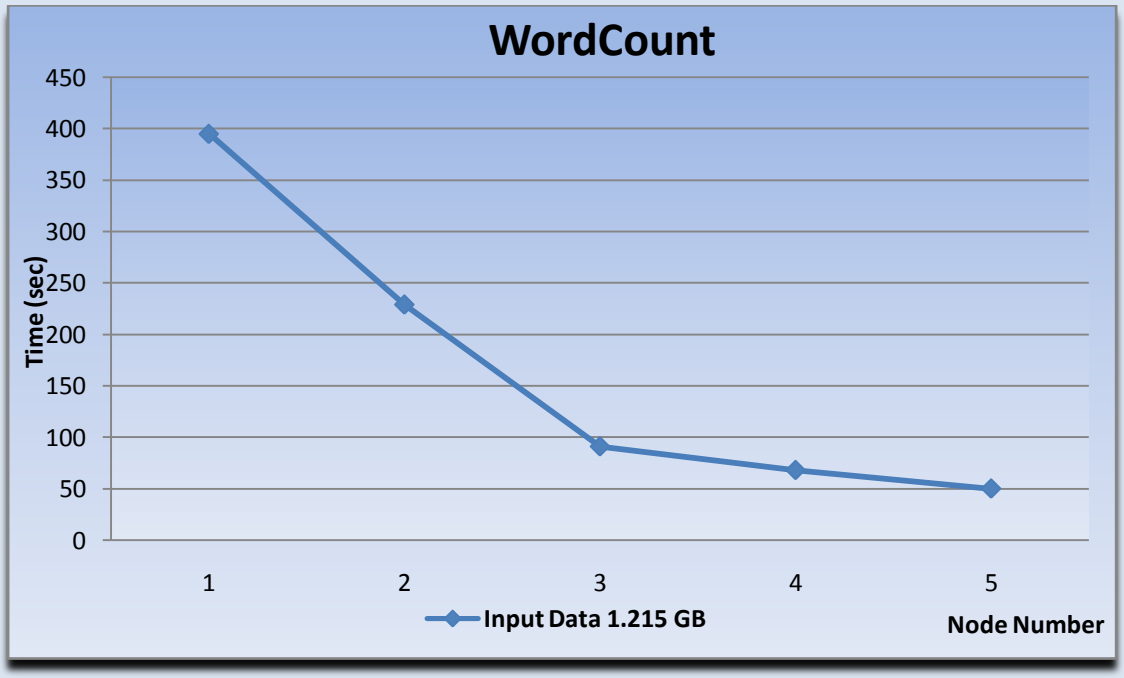
Node Num	IDF	Doc_Weights	Queries(Results &Sort)
5	34 sec	37 sec	33 sec+23 sec =56 sec
4	53 sec	45 sec	50 sec +17 sec =67sec
3	57 sec	52 sec	67 sec+22 sec =89 sec
2	104 sec	99 sec	138 +30 =168 sec
1	241 sec	201 sec	495+54=549 sec

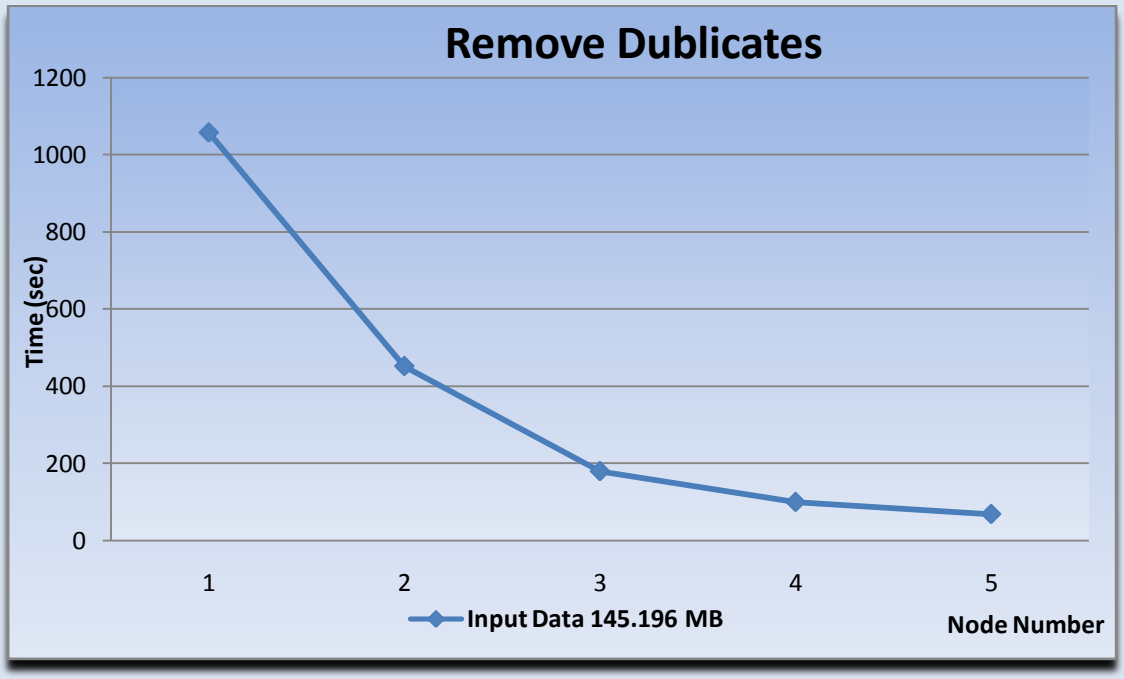
Πίνακας 14: Execution Time Vs Node Number

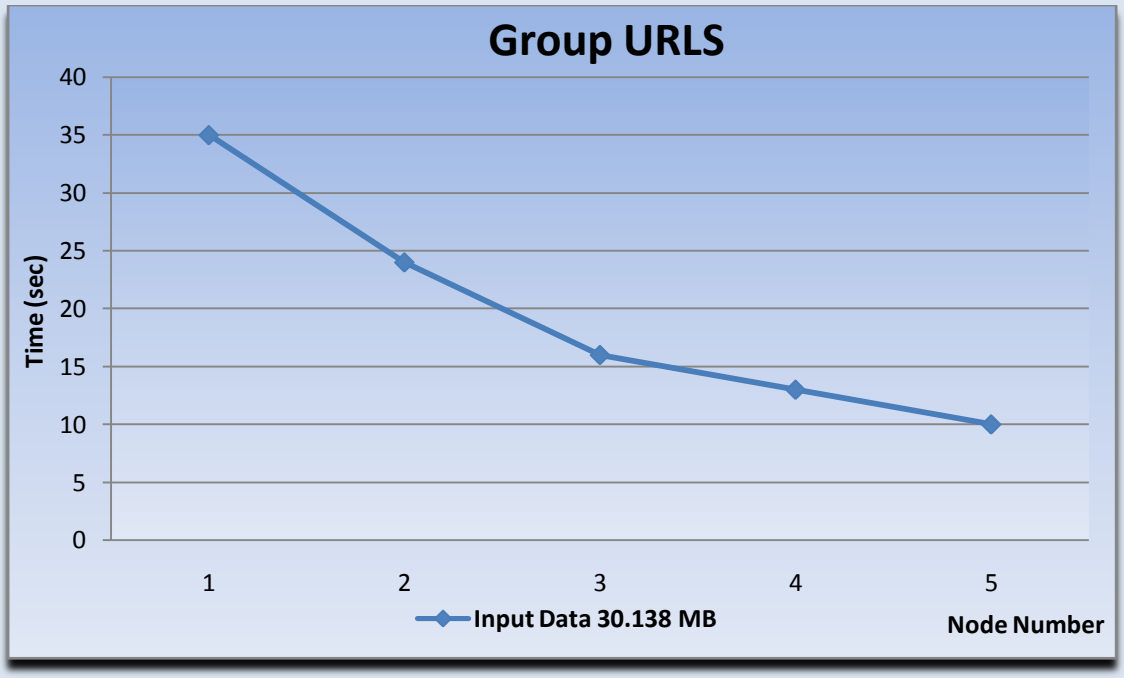
5.8.1 Διαγραμματικές Απεικονίσεις Κάθε Φάσης

- 1η ΦΑΣΗ ΕΡΓΑΣΙΑΣ

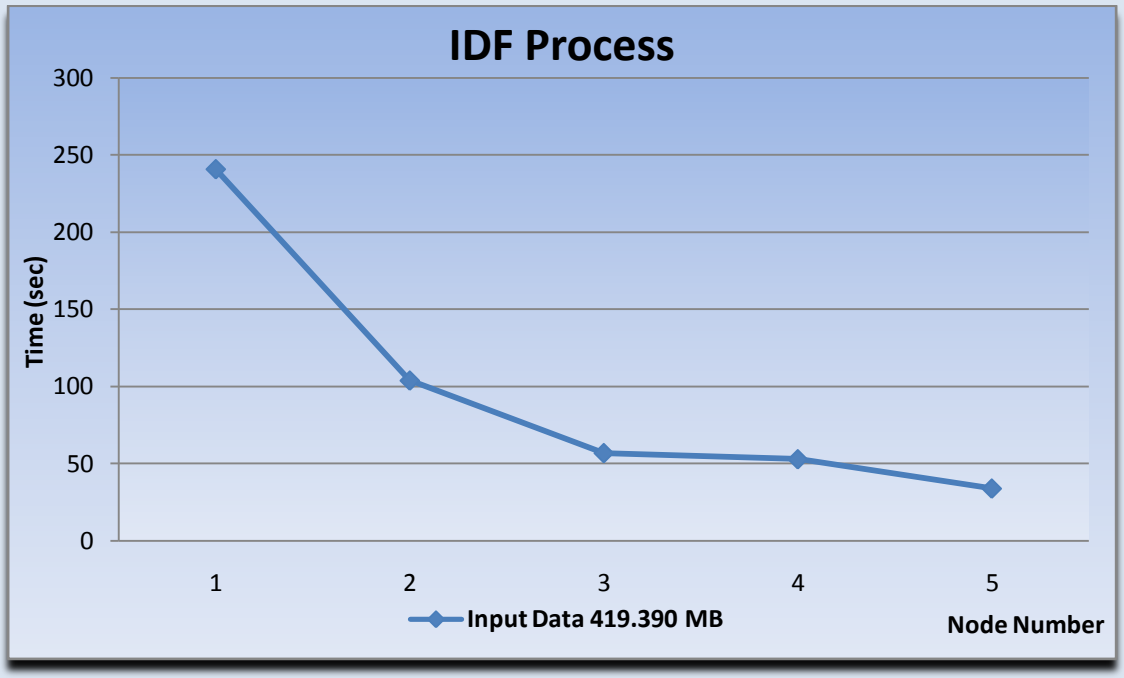


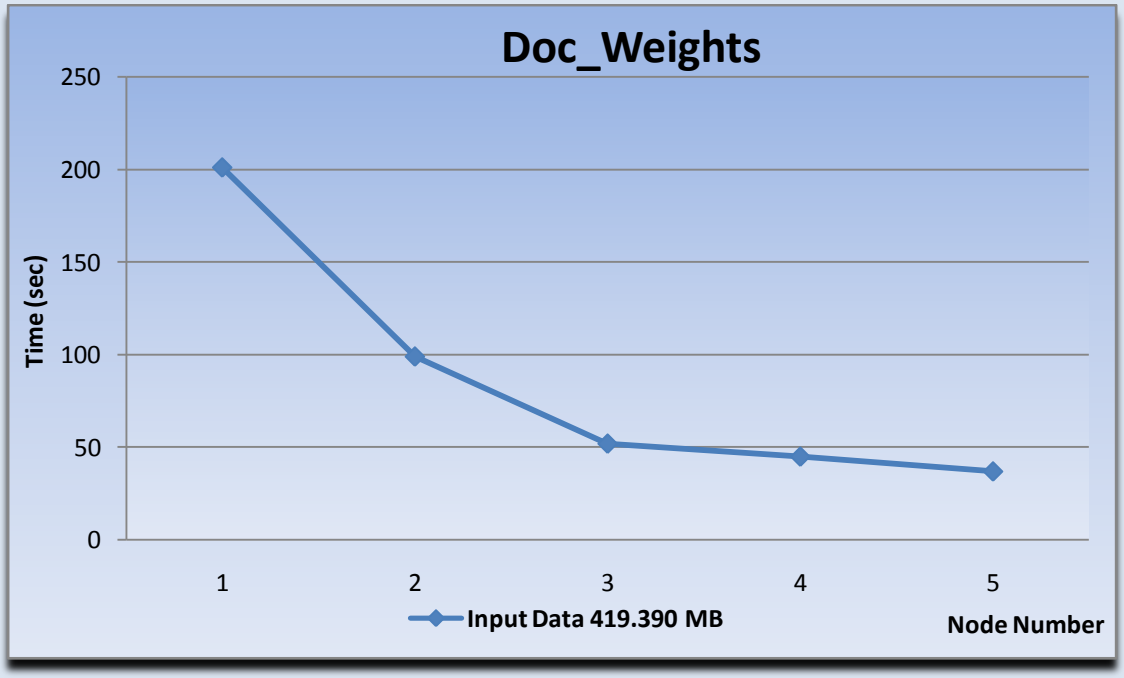


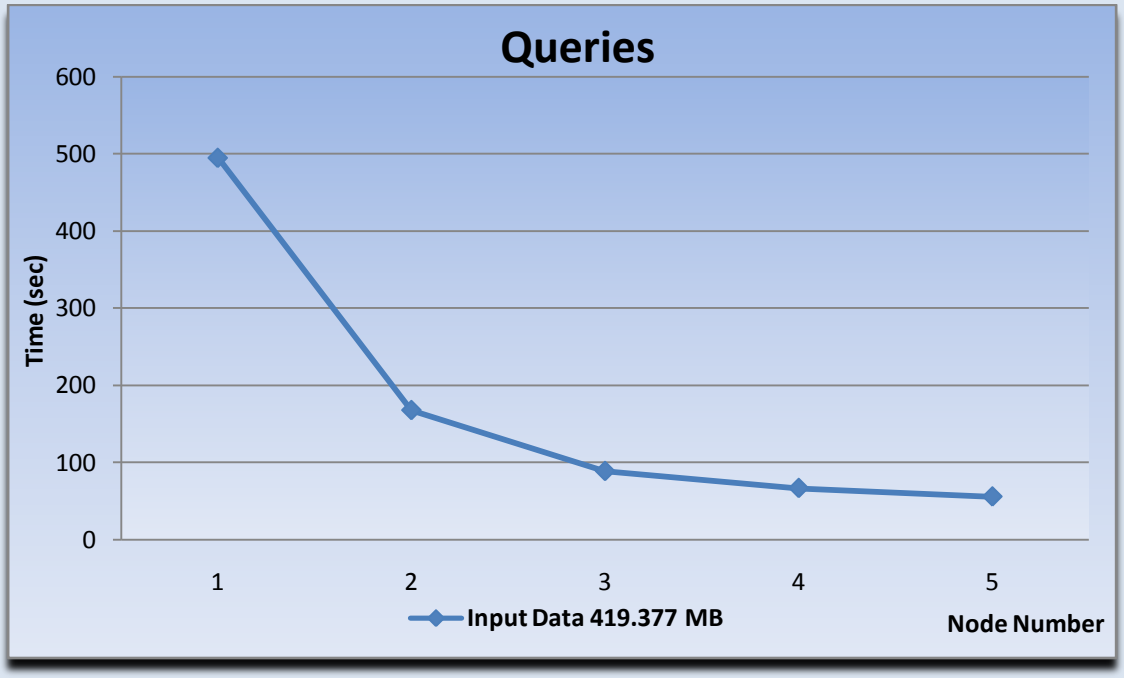

Remove Dublicates
1200
1000
800
600
400
200
0
Time (sec)
1
2
3
4
5
Input Data 145.196 MB
Node Number


Group URLS
40
35
30
25
20
15
10
5
0
Time (sec)
1
2
3
4
5
Input Data 30.138 MB
Node Number

❖ 2η ΦΑΣΗ ΕΡΓΑΣΙΑΣ







5.8.2 Σχολιασμός Αποτελεσμάτων –Γενικά Συμπεράσματα

Από τα παραπάνω πειράματα διαφαίνεται πλέον καθαρά η δυναμική του Hadoop συναρτήσει του μεγέθους του Cluster .Αυτό που παρατηρούμε είναι αυτό που έχουμε επισημάνεις πολλές φορές , πως οι παράλληλοι σταθμοί επεξεργασίας συμβάλλουν σημαντικά στην απόδοση του συστήματος.

Σε όλες τις περιπτώσεις αυτό που εξάγεται είναι μια κοινή συμπεριφορά κατά την μείωση των διαθέσιμων κόμβων -η αύξηση του χρόνου εκτέλεση και μια σχεδόν γραμμική συμπεριφορά κατά την αύξηση του αριθμού των κόμβων. Αυτό αποδίδεται στο γεγονός πώς ο καταμερισμός του όγκου της πληροφορίας γίνεται πιο μικρός οπότε και κάθε κόμβος αναλαμβάνει να επεξεργαστεί συνολικά περισσότερη πληροφορία. Αυτό με την σειρά του οδηγεί στην αύξηση του χρόνου αποπεράτωσης μίας εργασίας άρα και σε μείωση της αποδοτικότητας.

Και όχι μόνο αυτό. Σε όλα τα πειράματα που κάναμε οι αστοχίες που συνολικά είχε το σύστημά μας ήταν μηδενικές. Ακόμα και κατά την φάση του Crawling που έπρεπε να «τρέξει» το τμήμα αυτό για αρκετές ώρες μέχρι να συλλέξει τα δεδομένα δεν είχαμε καμία αστοχία ή απώλεια πληροφορίας.

Αυτό το χαρακτηριστικό αποδίδεται σε δύο παράγοντες. Πρώτον, στην αντιμετώπιση πιθανών προβλημάτων με την χρήση εξαιρέσεων (Πχ *Socket Exceptions*) αλλά κυρίως στο γεγονός ότι το Hadoop εξ' ορισμού χρησιμοποιεί το λεγόμενο *Speculative Execution* και την επανάληψη μίας διαδικασίας αν εκείνη αποτύχει.

Τι εννοούμε με αυτά τα δύο χαρακτηριστικά; Ως προς το πρώτο, αυτό που ουσιαστικά επιτελείται είναι η επικοινωνία των κόμβων μεταξύ τους οπότε αν για κάποιο λόγο ένας σταθμός επεξεργασίας εντοπίσει πως κάποιος άλλος δεν δίνει σήμα αναφοράς για ένα χρονικό διάστημα , αναλαμβάνει και επεξεργάζεται τα δεδομένα του κόμβου από το σημείο και μετά που είχε ενημερώσει για τελευταία φορά ο σταθμός που δεν έχει ενημερώσει τους υπόλοιπους. Κατ ΄ αυτόν τον τρόπο αν ένας σταθμός τεθεί εκτός λειτουργίας ή απλά καθυστερεί να ενημερώσει τους υπόλοιπους κόμβους συνολικά το σύστημα καταφέρνει ανανήπτει οπότε και δεν έχουμε απώλεια της πληροφορίας.

Ως προς το δεύτερο χαρακτηριστικό, αυτό που γίνεται είναι η επανάληψη μίας διαδικασίας που έχουν υποπέσει σε αστοχία πχ λόγω φύσεως του κώδικα ,τόσες φορές όσες τις ορίζει ο χρήστης(default 4). Επομένως, αν για κάποιο λόγο μία διεργασία υποπέσει σε σφάλμα (πχ αστοχία βάσης) συνολικά το πρόγραμμα δεν κινδυνεύει να τερματίσει βίαια. Στην περίπτωση όμως που και κατά τις τέσσερις επαναλήψεις παρατηρηθεί το ίδιο πρόβλημα τότε τερματίζονται όλες οι διαδικασίες.

ΣΥΓΚΡΙΣΗ ΜΕ ΑΛΛΑ ΣΥΣΤΗΜΑΤΑ

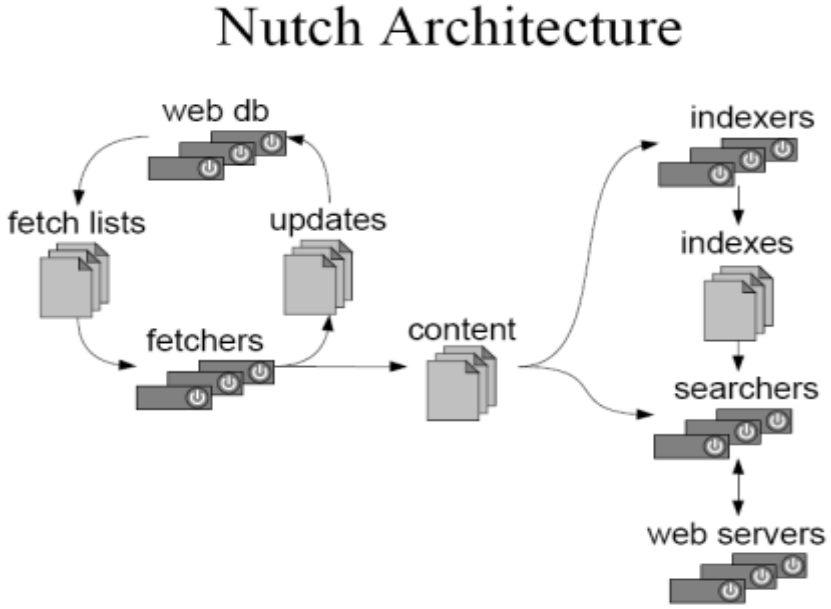
Στο παρόν κεφάλαιο θα παρουσιάσουμε και άλλες αρχιτεκτονικές που έχουν σχεδιαστεί για την κατασκευή συστημάτων αναζήτησης. Χαρακτηριστικό παράδειγμα εφαρμογής που χρησιμοποιεί το Hadoop για αυτόν τον σκοπό αποτελεί το **Nutch.**

Το *Nutch* ,όπως είπαμε και στο αντίστοιχο κεφάλαιο παρουσίασης του Hadoop έκανε την εμφάνισή του το 2004 και σε συνδυασμό με το project Lucene που επεξεργάζεται το περιεχόμενο κειμένων κατασκευάστηκε ένα σύστημα αναζήτησης που στηρίζεται στην επεξεργασία δεδομένων με βάση το μοτίβο Map/Reduce.

Το Nutch διαχωρίζεται σε δύο μεγάλα τμήματα στο κομμάτι του Crawler και στο κομμάτι της επεξεργασίας του περιεχομένου ,για την υποβολή ερωτήσεων, που έχουν συλλεχθεί από την φάση του Crawling. Ο Crawler απλά προσεγγίζει ιστοσελίδες και δημιουργεί ένα Inverted Index πάνω στο περιεχόμενο που έχει προσεγγίσει. Από κει και έπειτα το Inverted Index χρησιμοποιείται από το Lucene το οποίο και αναλαμβάνει να κατασκευάσει το μοντέλο για την αναπαράσταση του περιεχομένου και να απαντήσει τα ερωτήματα του χρήστη.

Τα βασικά σημεία στα οποία συνοψίζεται η διαδικασία του Crawling είναι τα εξής:

1. ***Δημιουργία Βάσης Για Δεικτοδότηση Διευθύνσεων***
2. ***Εισαγωγή Διευθύνσεων Στην Βάση***
3. ***Δημιουργία Λίστας Για Προσέγγιση Διευθύνσεων***
4. ***Προσέγγιση Περιεχομένου Ιστοσελίδων***
5. ***Ενημέρωση Βάσης Από Τα Εξαγόμενα Links***
6. ***Επανάληψη βημάτων 3-5 Μέχρι Το Επιθυμητό Βάθος***
7. ***Ενημέρωση Τμημάτων Με Scores –links Από την Βάση***
8. ***Δεικτοδότηση Περιεχομένου***
9. ***Αφαίρεση Διπλοτύπων***
10. ***Ενοποίηση των Indexes Σε Ένα Για Την Αναζήτηση***



Εικόνα 3: Διαγραμματική Παρουσίαση Αρχιτεκτονικής Nutch

Από τα παραπάνω στοιχεία αυτό που παρατηρούμε μέσα σε γενικά πλαίσια είναι πως η βασικός κορμός της αρχιτεκτονικής του Nutch είναι αρκετά κοντά με τον δικό μας αλλά υπάρχουν δύο βασικές διαφοροποιήσεις. Πρώτον, η διαδικασία επεξεργασίας του περιεχομένου και η πρώτη φάση για την κατασκευή του Inverted Indexing πραγματοποιείται σε εμάς κατά το Fetching στο κομμάτι του Reducing, την στιγμή που το Nutch απομονώνει κατά το Fetching το περιεχόμενο και επιδρά πάνω σε αυτό για την κατασκευή του Index σε δεύτερο βήμα.

Δεύτερο χαρακτηριστικό που διαφοροποιεί την εργασία μας είναι η φάση επεξεργασίας του περιεχομένου για την κατασκευή του Index αναζήτησης. Το Nutch για αυτόν τον σκοπό χρησιμοποιεί το Lucene οπότε και δύναται να μοντελοποιηθεί το περιεχόμενο που απομονώθηκε βάσει πολλών διαφορετικών μοντέλων - *Boolean, Vector Space* – την στιγμή που εμείς παρέχουμε την επιλογή μονάχα της Vector Space αναπαράστασης. Αυτό βέβαια που ξεχωρίζει την εργασία μας σε αυτήν την περίπτωση είναι ότι την διαδικασία της ανάλυσης του περιεχομένου γίνεται καθαρά σε Map/Reduce βήματα.

Επιπλέον διαφορές που εντοπίζονται ανάμεσα στις δύο αρχιτεκτονικές αποτελούν οι έλεγχοι που επιβάλλονται από το μοντέλο μας για την προσέγγιση σελίδων θέτοντας ένα χρονικό παράθυρο για την απομόνωση του περιεχομένου. Αντίθετα, στο Nutch δεν υφίσταται ένα τέτοιο χαρακτηριστικό που να απορρίπτει σελίδες που έχει επισκεφτεί στο παρελθόν όπως επίσης δεν κρατιέται πληροφορία στο σύστημα δεικτοδότησης για το περιεχόμενο κάθε σελίδας που γίνεται προσεγγίζεται-page signature.

Επιπρόσθετα, στο Nutch δεν αποθηκεύεται χρονική πληροφορία συσχετισμένη με το περιεχόμενο των σελίδων πράγμα που σε εμάς δίνει το πλεονέκτημα στον χρήστη να μπορεί αν αναζητά πληροφορία βάσει timestamps οπότε και να αποδίδεται περιεχόμενο που να είναι πιο κοντά στα στοιχεία που τον ενδιαφέρουν.

Αντίθετα, το Nutch δίνει έμφαση περισσότερο στην καταχώριση πληροφορίας για τα εσωτερικά Links κάθε διεύθυνσης ώστε στην πορεία να μπορέσει να υπολογίσει hits&scores και βάσει αυτών να κατατάξει τις διαθέσιμες σελίδες με κάποιο κριτήριο σημαντικότητας. Κάτι τέτοιο δεν χρειάστηκε να υλοποιηθεί από εμάς αφού κατασκευάσαμε ένα μοντέλο που εξ' ορισμού κατηγοριοποιεί τα διαθέσιμα κείμενα με κριτήριο την ομοιότητα του Query με τον χρήστη.

ΕΠΙΛΟΓΟΣ

Εν κατακλείδι, στην παρούσα εργασία έγινε μία προσπάθεια για την κατασκευή ενός συστήματος που προσεγγίζει αρκετά καλά την αρχιτεκτονική και πραγματικών συστημάτων αναζήτησης δεδομένου πως επεξεργαζόμαστε μεγάλο όγκου πληροφορίας. Προφανώς για να επεξεργαστούμε τέτοια μεγέθη υπήρξε η ανάγκη καταμερισμού των εργασιών αλλά κυρίως να στραφούμε σε εναλλακτικές μεθόδους διαχείρισης των δεδομένων σε επίπεδο δίσκου.

Την λύση σε τέτοιου είδους θέματα έδωσε η αρχιτεκτονική του Hadoop προσφέροντας παραλληλισμό εργασιών μέσω του μοτίβου Map/Reduce αλλά και μέσω της τμηματοποίησης εργασιών και ανάθεσης του φόρτου εργασίας στους διαθεσίμου κόμβους.

Καθοριστικό ρόλο λοιπόν στην αποδοτικότητα του μοντέλου στάθηκε το μέγεθος του cluster αφού από την δεύτερη φάση των πειραμάτων βλέπουμε πως με την περαιτέρω μείωση των κόμβων ο χρόνος εκτέλεσης αυξάνεται αισθητά σε όλα τα βήματα μέχρι και την υποβολή ερωτήσεων από τον χρήστη. Προφανώς αν θέλαμε να αυξήσουμε την αποδοτικότητα του συστήματος πρώτο μέλημα θα ήταν να αυξήσουμε και τον αριθμό των παράλληλων σταθμών επεξεργασίας αλλά και να υλοποιήσουμε ένα μοντέλο που θα μοντελοποιεί δεδομένα σε επίπεδο μνήμης για την άμεση ανταπόκριση στα ερωτήματα των χρηστών.

Ανεξάρτητα πάντως από το πλήθος των κόμβων αυτό που εξάγεται είναι πώς η δυναμική του Hadoop είναι μοναδική ως προς τον χειρισμό και την ανάλυση μεγάλου όγκου πληροφορίας αφού και για όγκο πληροφορίας πολλών GB καταφέρνει και επεξεργάζεται δεδομένα όπως κανένα άλλο σύστημα μέχρι σήμερα. Αυτός είναι και ο λόγος που μεγάλες εταιρείες σήμερα υιοθετούν το Hadoop για να επεξεργαστούν το σύνολο της πληροφορίας που διαθέτουν (*Yahoo, Amazon ,New York Times)* είτε ως μεμονωμένο σύστημα είτε ως κομμάτι μεγάλων Projects.(Nutch, Hive-Cloud Computing).

ΠΑΡΑΡΤΗΜΑ

Α.1 ΑΛΓΟΡΙΘΜΟΣ ΔΕΙΚΤΔΟΤΗΣΗΣ:B-Tree

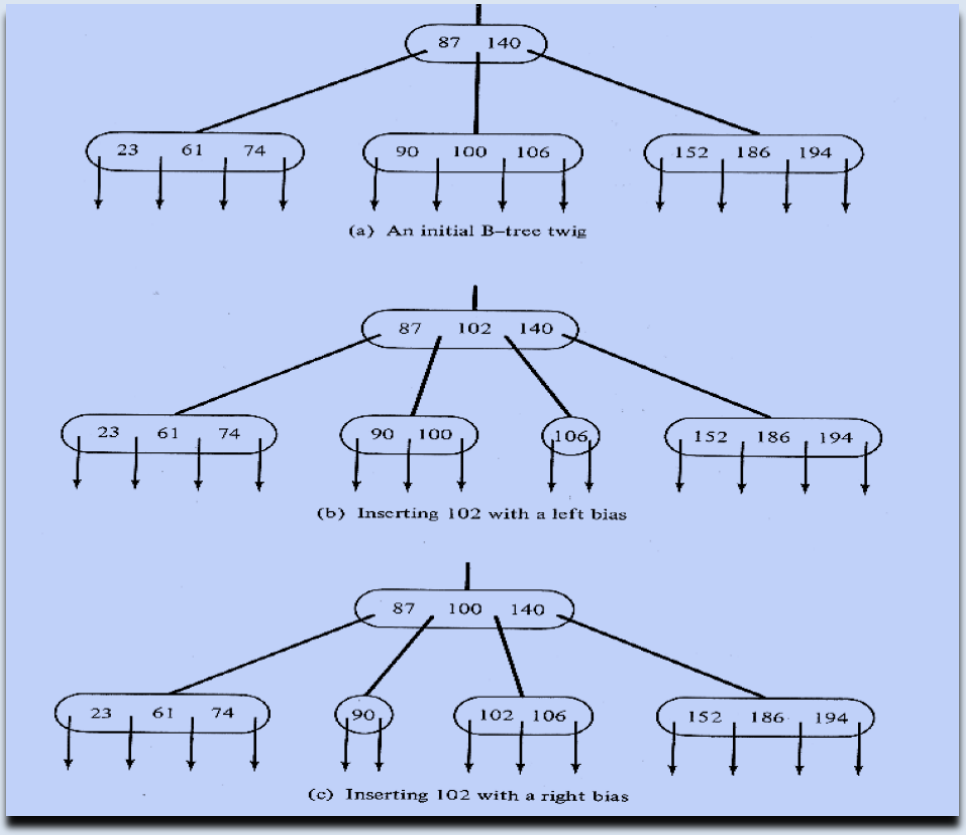
Συχνά όταν διαπραγματευόμαστε δεδομένα μεγάλου όγκου προκύπτει το πρόβλημα της δεικτοδότησης κάποιων δομών. Το θέμα που αναφύεται λοιπόν είναι πώς θα καταφέρουμε να καταχωρήσουμε ένα μεγάλο αριθμό στοιχείων αλλά συνάμα να έχουμε καλή κατανομή της μνήμης-δίσκου και γρήγορη πρόσβαση σε αυτά. Η απάντηση σε αυτούς τους προβληματισμούς δίνεται από τον πολύ γνωστό σε όλους αλγόριθμο B-TREE.

Η ιδιότητα που κάνει ξεχωριστό αυτό τον αλγόριθμο είναι το ότι το δέντρο που δημιουργείται είναι ισορροπημένο ,το ύψος του δέντρου διατηρείται σχετικά μικρό και ταυτόχρονα όλα τα φύλλα βρίσκονται σε ίδιο ύψος. Σε δεύτερη ανάλυση αυτό υποδηλώνει πως για να κάνουμε μία αναζήτηση- στην χειρότερη περίπτωση το στοιχείο που θέλουμε μπορεί να είναι σε ένα φύλλο- η πολυπλοκότητα θα είναι logN και θα είναι ίδια για όλα τα φύλλα.

Για να κάνουμε όμως πιο σαφή τα πράγματα ας δούμε την λογική του αλγορίθμου. Σε κάθε Btree κάθε μονοπάτι από την κορυφή μέχρι τα φύλλα έχει το ίδιο μήκος h,όσο το ύψος του δέντρου. Σε κάθε κόμβο η αποθήκευση γίνεται βάσει του μοτίβου (key, value) ,δηλαδή για σε κάθε κλειδί αποδίδεται μία τιμή-value . Και τα δύο στοιχεία τελικά θα καταχωρηθούν σε ένα κόμβο ενώ σε καθένα εκ αυτών ο μέγιστος αριθμός κλειδιών είναι n-1, όπου n είναι ο βαθμός του δέντρου.

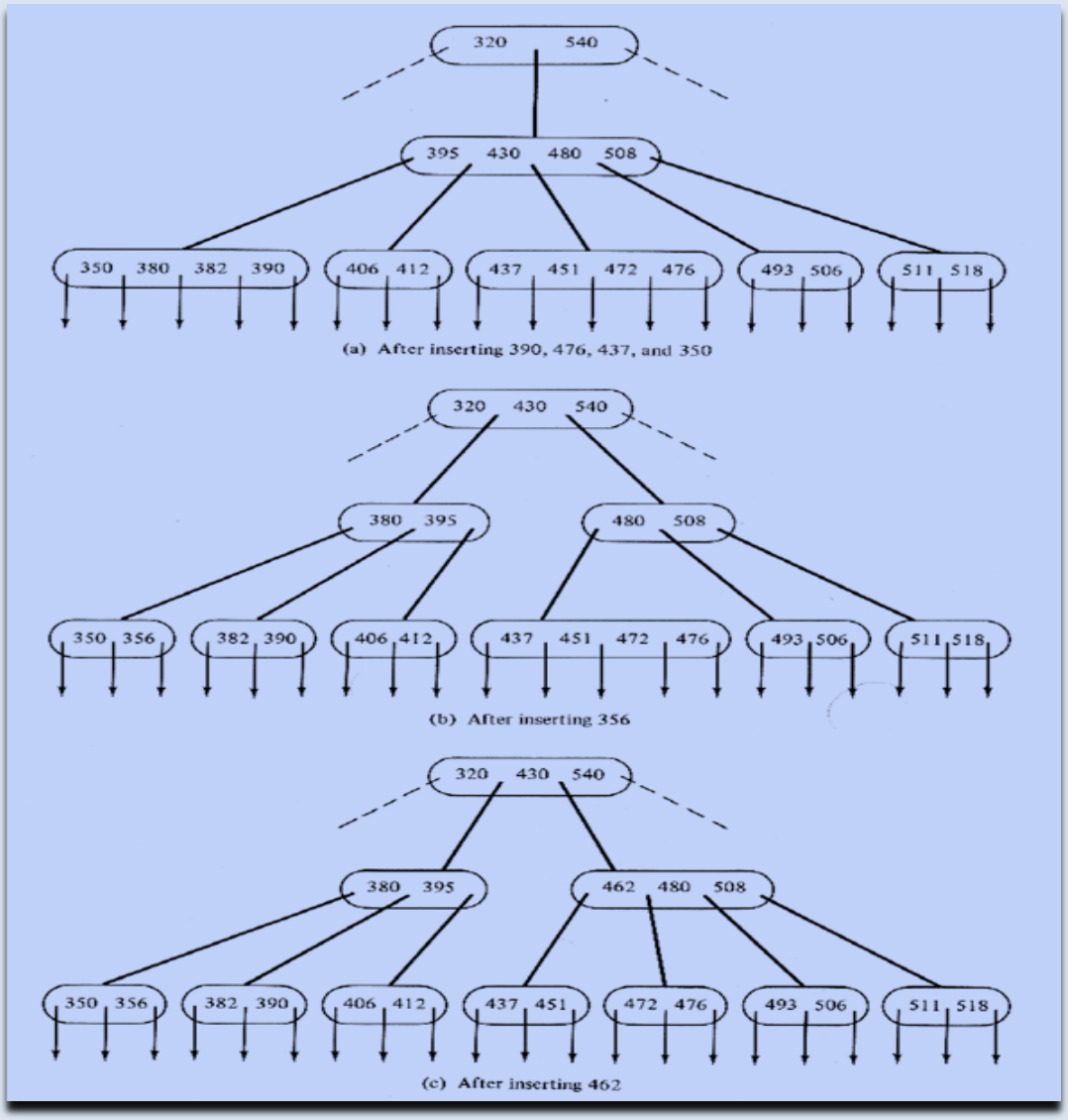
Για να αποκτήσει τώρα αυτή την δομή το δέντρο κατά τη εισαγωγή των στοιχείων ακολουθείται η εξής τακτική: Ακολουθούμε το κατάλληλο μονοπάτι για να εισάγουμε ένα στοιχείο βάσει του κλειδιού ξεκινώντας από την ρίζα μέχρι στο επίπεδο των φύλλων. Αν ο δεδομένος κόμβος δεν είναι γεμάτος τότε εισάγεται το στοιχείο στον κόμβο αυτό διαφορετικά δημιουργούμε ένα νέο και σπάμε τον παλιό. Κατά αυτό τον τρόπο δημιουργείται ένας αριστερός και ένας δεξιός κόμβος κατανέμοντας τα n/2 πιο μικρά στοιχεία στον αριστερό και τα n/2 μεγαλύτερα στον δεξί. Το «μεσαίο» στοιχείο θα μεταφερθεί στον αμέσως προηγούμενο κόμβο πατέρα. Στην περίπτωση τώρα που το n είναι ζυγός αριθμός τότε ένα στοιχείο θα καταχωρηθεί είτε στον δεξιό είτε στον αριστερό κόμβο.

Για να κάνουμε πιο κατανοητά τα παραπάνω ας δούμε αυτή την διαδικασία μέσα από ένα παράδειγμα για ένα δέντρο βαθμού **n=5**.



Εικόνα 4: Btree Insertion N=5

Από τα παραπάνω βλέπουμε πώς αν ένας κόμβος πατέρας είναι γεμάτος θα χρειαστεί μετά από την διάσπαση σε ένα κατώτερο κόμβο-παιδί να διασπαστεί και ο ίδιος. Αυτή η διαδικασία μπορεί να συνεχιστεί διαδοχικά και σε επόμενους κόμβους προς τα πάνω πιθανώς μέχρι και την ρίζα όπως φαίνεται και παρακάτω.



Εικόνα 5:*Btree Insertion Splitting Propagation*

Τί πετυχαίνουμε κατά αυτό τον τρόπο; Όπως φαίνεται από τα παραπάνω παραδείγματα ένα Btree δέντρο παραμένει ισορροπημένο δηλαδή όλα τα φύλλα βρίσκονται σε ίδιο ύψος .Επομένως, για όλα τα φύλλα χρειάζεται ίδιος αριθμός προσβάσεων ο οποίος και διατηρείται χαμηλός, παρά τον μεγάλο αριθμό εισαγόμενων στοιχείων, λόγω του μικρού ύψους του δέντρου.

n \ N	10^3	10^4	10^5	10^6	10^7
10	3	4	5	6	7
50	2	3	3	4	4
100	2	2	3	3	4
150	2	2	3	3	4

disc accesses

Εικόνα 6: *Απόδοση Βάσει Βαθμό Δέντρου και Αριθμό Κόμβων*

Όπως διαφαίνεται και από την εικόνα όσο αυξάνεται το *n* ,δηλαδή ο μέγιστος αριθμός κλειδιών ανά κόμβο, τόσο πιο λίγες προσβάσεις έχουμε στον δίσκο και αυτό γιατί ουσιαστικά ομαδοποιούμε πιο πολλά δεδομένα πριν τα διοχετεύσουμε στον δίσκο. Αυτομάτως αυτό υποδηλώνει πώς αυξάνοντας τον αριθμό των στοιχείων ανά κόμβο πρέπει να αυξηθεί και το paging άρα και το μέγεθος των δεδομένων που θα μεταφερθούν από τον την μνήμη στον δίσκο.

Εν κατακλείδι, μία τέτοια δομή προσφέρει το σημαντικό πλεονέκτημα της δεικτοδότησης πολλαπλών στοιχείων εξασφαλίζοντας μικρή πολυπλοκότατα κατά την αναζήτηση. Αυτός είναι και ο λόγος που μας κάνει να ξεχωρίζουμε αυτόν τον αλγόριθμο ανάμεσα σε όλους τους άλλους γι' αυτό όπως θα δούμε και παρακάτω[0] χρησιμοποιήσαμε αυτό το μέσο δεικτοδότησης για να εξυπηρετήσουμε διάφορους σκοπούς της εργασίας μας .

Α.2 ΜΕΤΕΩΡΟΛΟΓΙΚΑ ΔΕΔΟΜΕΝΑ MAP/REDUCE- JAVA

Παρατίθενται παρακάτω μία σύντομη υλοποίηση του παραδείγματος στην παράγραφο, *σελ 13*.

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
```

```
public class MaxTemperatureMapper extends MapReduceBase implements Map-
per<LongWritable, Text, Text, IntWritable>
{
	private static final int MISSING = 9999;

	public void map(LongWritable key, Text value,OutputCollector<Text, IntWritable>
	output, Reporter reporter)throws IOException
	{
		String line = value.toString();
		String year = line.substring(15, 19);
		int airTemperature;

		if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs
			airTemperature = Integer.parseInt(line.substring(88, 92));
		} else {
			airTemperature = Integer.parseInt(line.substring(87, 92));
		 }
		String quality = line.substring(92, 93);
		if (airTemperature != MISSING && quality.matches("[01459]")) {
		output.collect(new Text(year), new IntWritable(airTemperature));
		}
	}
}
```

Mapper Implementation For Weather Data

Κοιτάζοντας καλύτερα την παραπάνω υλοποίηση αυτό που ουσιαστικά πρέπει να επεξηγήσουμε είναι τα ορίσματα του mapper. Το key αντιπροσωπεύει για τον mapper ένα κωδικό βάσει του οποίου κωδικοποιείται κάθε γραμμή του κειμένου εισόδου, η value μεταβλητή είναι η γραμμή του κάθε κειμένου που κάνει fetch ο mapper ,οπότε με την βοήθεια αυτής μπορούμε να επεξεργαστούμε κάθε φορά τα δεδομένα εισόδου. Εδώ λοιπόν βάσει αυτής παίρνουμε κάθε φορά το String μίας γραμμής που περιέχει το σύνολο της πληροφορίας και απομονώνουμε το έτος και την θερμοκρασία.

Η μεταβλητή output είναι εκείνη που μας βοηθά να διοχετεύσουμε στο επόμενο επίπεδο τις ομάδες (key,data_to_reducers) ,εδώ (έτος, θερμοκρασία). Τέλος, η μεταβλητή reporter χρησιμοποιείται για να δίνει αναφορά στο σύστημα για την εξέλιξη της διαδικασίας του mapping.

Ας δούμε και τον Reducer:

```
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
```

```
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class MaxTemperatureReducer extends MapReduceBase implements
Reducer<Text, IntWritable, Text, IntWritable>
{

public void reduce (Text key, Iterator<IntWritable> values,OutputCollector
        <Text, IntWritable> output, Reporter reporter) throws I OException
        {
                int maxValue = Integer.MIN_VALUE;
                while (values.hasNext ())
                {
                        maxValue = Math.max (maxValue, values. next().get());
                }
        output.collect(key, new IntWritable(maxValue));
        }
}
```

Reducer Implementation For Weather Data

Τα βασικά σημεία που πρέπει να προσέξουμε είναι τα ορίσματα του Reducer. Το όρισμα Key όπως περιμέναμε είναι το key που διοχετεύσαμε μέσω του Output Collector στον Reducer ενώ η values είναι τα ομαδοποιημένα δεδομένα που πρέπει να τροποποιήσουμε για να λάβουμε τα τελικά αποτελέσματα. Στην περίπτωση μας το πρώτο όρισμα είναι τα έτη και το δεύτερο οι ομάδες με την θερμοκρασία κάθε έτους. Ετσι μέσα από τον βρόχο στο σώμα της Reduce καταφέρνουμε και απομονώνουμε την μέγιστη θερμοκρασία οπότε τελικά ενημερώνουμε τον Reducer πως στην έξοδο θέλουμε να μας εμφανίζονται στα κείμενα το ζεύγος (year,max_temperature). Τέλος, ο reporter έχει τον ίδιο ρόλο όπως στο mapping.

Α.3 ΠΑΡΑΔΕΙΓΜΑ VECTOR SPACE ΑΝΑΠΑΡΑΣΤΑΣΗΣ

Ας υποθέσουμε πώς έχουμε τρία διαφορετικά κείμενα:

D1:"Shipment of gold damaged in a fire"
D2: "Delivery of silver arrived in a silver truck"
D3: "Shipment of gold arrived in a truck"

Εφαρμόζοντας το μοντέλο παίρνουμε τον παρακάτω πίνακα :

TERM VECTOR MODEL BASED ON $w_i = tf_i * IDF_i$

Query, Q: "gold silver truck"
D_1: "Shipment of gold damaged in a fire"
D_2: "Delivery of silver arrived in a silver truck"
D_3: "Shipment of gold arrived in a truck"
$D = 3$; $IDF = \log(D/df_i)$

		Counts, tf_i						Weights, $w_i = tf_i * IDF_i$			
Terms	Q	D_1	D_2	D_3	df_i	D/df_i	IDF_i	Q	D_1	D_2	D_3
a	0	1	1	1	3	3/3 = 1	0	0	0	0	0
arrived	0	0	1	1	2	3/2 = 1.5	0.1761	0	0	0.1761	0.1761
damaged	0	1	0	0	1	3/1 = 3	0.4771	0	0.4771	0	0
delivery	0	0	1	0	1	3/1 = 3	0.4771	0	0	0.4771	0
fire	0	1	0	0	1	3/1 = 3	0.4771	0	0.4771	0	0
gold	1	1	0	1	2	3/2 = 1.5	0.1761	0.1761	0.1761	0	0.1761
in	0	1	1	1	3	3/3 = 1	0	0	0	0	0
of	0	1	1	1	3	3/3 = 1	0	0	0	0	0
silver	1	0	2	0	1	3/1 = 3	0.4771	0.4771	0	0.9542	0
shipment	0	1	0	1	2	3/2 = 1.5	0.1761	0	0.1761	0	0.1761
truck	1	0	1	1	2	3/2 = 1.5	0.1761	0.1761	0	0.1761	0.1761

Πίνακας Με Τους Συντελεστές Του *Vector Space Model.*

Για να βρούμε τώρα τα βάρη των κειμένων εργαζόμαστε ως εξής: Αρχικά υπολογίζουμε τα συνολικά βάρη των κειμένων συμπεριλαμβάνοντας όλους τους όρους κάθε D_i και το Q του χρήστη.

$$|D_1| = \sqrt{0.4771^2 + 0.4771^2 + 0.1761^2 + 0.1761^2} = \sqrt{0.5173} = 0.7192$$

$$|D_2| = \sqrt{0.1761^2 + 0.4771^2 + 0.9542^2 + 0.1761^2} = \sqrt{1.2001} = 1.0955$$

$$|D_3| = \sqrt{0.1761^2 + 0.1761^2 + 0.1761^2 + 0.1761^2} = \sqrt{0.1240} = 0.3522$$

$$\therefore |D_i| = \sqrt{\sum_i w_{i,j}^2}$$

$$|Q| = \sqrt{0.1761^2 + 0.4771^2 + 0.1761^2} = \sqrt{0.2896} = 0.5382$$

Υπολογισμός Βαρών D_j ,Q.

Στην συνέχεια υπολογίζουμε το εσωτερικό γινόμενο του Q,D_i λαμβάνοντας :

$$Q \bullet D_1 = 0.1761 * 0.1761 = 0.0310$$
$$Q \bullet D_2 = 0.4771 * 0.9542 + 0.1761 * 0.1761 = 0.4862$$
$$Q \bullet D_3 = 0.1761 * 0.1761 + 0.1761 * 0.1761 = 0.0620$$

Υπολογισμός Εσωτερικού Γινομένου QD_j

Τελικά υπολογίζουμε την ομοιότητα βρίσκοντας την γωνία:

$$\text{Cosine } \theta_{D_1} = \frac{Q \bullet D_1}{|Q| * |D_1|} = \frac{0.0310}{0.5382 * 0.7192} = 0.0801$$

$$\text{Cosine } \theta_{D_2} = \frac{Q \bullet D_2}{|Q| * |D_2|} = \frac{0.4862}{0.5382 * 1.0955} = 0.8246$$

$$\text{Cosine } \theta_{D_3} = \frac{Q \bullet D_3}{|Q| * |D_3|} = \frac{0.0620}{0.5382 * 0.3522} = 0.3271$$

Υπολογισμός Ομοιότητας Q, D_j

Άρα τελικά έχουμε: *Rank 1:* ***Doc 2*** *= 0.8246*
Rank 2: ***Doc 3*** *= 0.3271*
Rank 3: ***Doc 1*** *= 0.0801*

Βλέπουμε λοιπόν από τα παραπάνω πως για να μπορέσουμε να εφαρμόσουμε τον αλγόριθμο αρκεί να υπολογίσουμε για κάθε κείμενο το συνολικό βάρος και το μόνο που απομένει κατά τη υποβολή του ερωτήματος του χρήστη είναι να βρούμε την γωνία *cos(D,Q).* Κάτι τέτοιο μας διευκολύνει αρκετά καθώς μετά τον υπολογισμό των βαρών το μόνο που απαιτείται είναι να απομονώσουμε τα κείμενα που εμπεριέχουν τους όρους -ή κάποιους-του ερωτήματος του χρήστη και για αυτά τα κείμενα να βρούμε τον βαθμό ομοιότητας. Θα δούμε λοιπόν παρακάτω στο αντίστοιχο κεφάλαιο[0] που υλοποιείται το Vector Space Model πώς θα έχουμε υπολογίσει τα βάρη που αντιστοιχούν σε κάθε κείμενο που μας ενδιαφέρει οπότε στο τελικό βήμα που υποβάλλεται από τον χρήστη ένα επερώτημα θα αρκεί μονάχα η αναζήτηση ων αρχείων που εμπεριέχουν έναν ή περισσότερους όρους από το αρχικό Query.

Α.4 ΑΝΑΛΥΣΗ ΚΛΑΣΗΣ *Indexing.java*

Μέσω αυτής της κλάσης προφανώς υλοποιούμε την δεικτοδότηση των Urls. Για αυτό τον λόγο και έχουμε ορίσει private μεταβλητές –δείκτες σε B-tree και σε *RecordManager* ο μέσω του οποίου διαχειριζόμαστε την βάση με τα δεδομένα

που έχουμε αποθηκεύσει σε αυτή. Τα σημεία που πρέπει να επεξηγήσουμε σε αυτή την κλάση είναι ***Constructor*** και η μέθοδος ***Retrieve_UrlPages_Based_Date(String url,Date date,int action,Date EndDate)***.

Για τον Constructor αυτό που απλά πρέπει να αναφέρουμε είναι πως μέσω αυτού αρχικοποιούμε ένα νέο αντικείμενο B-tree αν δεν έχουμε καταχωρίσει κάτι στην βάση ή απλά φορτώνουμε τα δεδομένα που έχουμε ήδη μέσα σε αυτή φτιάχνοντας ένα B-tree με αυτά τα στοιχεία.

Όσον αφορά την μέθοδο ***Retrieve_UrlPages_Based_Date*** επιστρέφονται στον χρήστη τα ονόματα με τα οποία έχουν αποθηκευτεί τοπικά οι σελίδες που πληρούν τις προϋποθέσεις που δίνει ο χρήστης . Οι προϋποθέσεις σε αυτήν την περίπτωση είναι η ημερομηνία-ες βάσει τον όποιων ζητά ο χρήστης να του επιστραφούν οι πιθανές εκδόσεις μιας συγκεκριμένης ιστοσελίδας για το χρονικό περιθώριο που έχει δώσει ο χρήστης.

Οι επιλογές που παρέχονται από το σύστημα είναι αναζήτηση βάσει Url (***String url***) πριν από μία ημερομηνία(**Date date, action=1**) ,μετά από μία ημερομηνία(**Date date, action=2**),ίση με μία ημερομηνία(**Date date ,action=0**) και ανάμεσα σε 2 ημερομηνίες(**Date date ,Date EndDate ,action=3**). Πχ αν ο χρήστης επιθυμεί να του επιστραφούν για το www.yahoo.com όλες οι σελίδες για αυτό το Url από 1/12/2009-3/12/2009, τότε το όρισμα **date<-**1/12/2009 και το **EndDate<-**3/12/2009.

Ο τρόπος με τον οποίο γίνεται η αναζήτηση για να πάρουμε το αποτέλεσμα είναι απλός. Μετατρέποντας το Url σε signature και κάνουμε αναζήτηση στο B-tree για το αν υπάρχει τέτοιο κλειδί. Αν ναι, εξετάζουμε το Vector με τα timestamps και αν κάποιο στοιχείο ικανοποιεί την συνθήκη που δίνει ο χρήστης τότε και επιστρέφεται το όνομα της σελίδας-ες που έχουμε καταχωρίσει τοπικά.

Α.5 ΑΝΑΛΥΣΗ ΚΛΑΣΗΣ *Scheduling.java*

Ας δούμε πρώτα τους ελέγχους που πραγματοποιούμε (μέθοδος **IsUrlVisited(BitSet bt,URL_Entity urlEntity,int datewindow)**). Αρχικός έλεγχος πού γίνεται είναι φυσικά αν υπάρχει ήδη καταχωρημένο το τρέχον Url.Αν δεν είναι τότε χωρίς περαιτέρω έλεγχο εισάγουμε την διεύθυνση προς μελλοντική επεξεργασία. Στην περίπτωση που έχει ήδη δεικτοδοτηθεί τότε πρέπει να δούμε αν η ημερομηνία στην οποία την επισπευτήκαμε είναι τόσες μέρες πιο πριν όσο και η μεταβλητή *datewindow*. Αν έχει παρέλθει χρονικό διάστημα μεγαλύτερο ή ίσο με αυτό τότε και την εντάσσουμε στην ουρά .

Όσον αφορά την διαχείριση της ουράς, αρχικά να πούμε πως ορίσαμε μία μεταβλητή την ***MaxServerListSize*** η όποια και καθορίζει ποιο θα είναι το μέγιστο μήκος της ουράς που δεχόμαστε να έχει πριν διοχετεύσουμε τις νέες διευθύνσεις

στο Hdfs σύστημα. Έτσι , μπορούμε να ελέγχουμε το μέγεθος των δεδομένων που στέλνουμε προς επεξεργασία. Υπό άλλες συνθήκες αν δεν βάζαμε αυτό τον περιορισμό τότε η διαδικασία της εισαγωγής νέων διευθύνσεων θα γινόταν αέναη(θεωρητικά μέχρι να τελειώσει κάποιος Mapper)οπότε και θα κινδυνεύαμε να χάσουμε τις νέες διευθύνσεις αν συνέβαινε κάποια αστοχία. Επιπρόσθετα ,με την μεταβλητή αυτή μπορούμε και διαχειριζόμαστε την μνήμη καλύτερα, αφού όπως θα δούμε καθώς αυξάνει το μέγεθος του Btree τόσο πιο καθοριστικό γίνεται το να αποδεσμεύουμε δεδομένα.

Τέλος, απλά να πούμε πως οι private μεταβλητές ***Queue<URL_Entity> ServerList , Indexing Btree*** αντιπροσωπεύουν την ουρά για τον προγραμματισμό των διευθύνσεων και ένα δείκτη στην δομή δεικτοδότησης της εργασίας μας.

Α.6 ΑΝΑΛΥΣΗ ΚΛΑΣΗΣ UpdateHdfsInputeFile.java

Η μέθοδος ***UpdateUrlFile(Scheduling sc)*** έχει ρόλο να μεταφέρει τα εξαγόμενα Urls από την ουρά της κλάσης Scheduling σε ένα αρχείο στο Hdfs σύστημα .Ομοίως για τα νέα αυτά Urls δημιουργεί ένα άλλο αρχείο που αυτή την φορά θα τοποθετήσει στο **//BtreeFiles** του HDFS μεταφέροντας τα Bit Sets Signatures των προκειμένων Urls . Έτσι, στο τέλος της διαδικασίας του fetching τα αρχεία με τα νέα signatures θα ενημερώσουμε την βάση για τα νέα δεδομένα που έχουμε βρει ώστε να φτιάξουμε ένα κεντρικοποιημένο σύστημα δεικτοδότησης .βλ σελ [40]

Η μέθοδος ***UpdateBtreeFile (Queue qu)*** έχει παρόμοια αρμοδιότητα με την προηγούμενη. Ειδικότερα, μέσω αυτής καταχωρούμε τα δεδομένα που έχουν προκύψει από τους ελέγχους τους δεύτερου επιπέδου-Crawldb- (Url Signatures, Timestamps) σε αρχεία τα οποία και αυτά καταχωρούμε στο **//BtreeFiles** του HDFS. Τα στοιχεία αυτά βρίσκονται ήδη αποθηκευμένα σε μία ουρά η όποια όταν ξεπεράσει το όριο της μεταβλητής *MaxEntriesNum* τότε αδειάζει μεταφέροντας τα στοιχεία σε αρχείο. Έτσι, τα δεδομένα αυτά σε συνδυασμό με τα προηγούμενα θα καταχωρηθούν τελικά στην βάση μας ενημερώνοντας το σύστημα τόσο για τα νέα Urls που έχουμε συλλέξει όσο και για τα πιο παλιά που επισπευτήκαμε.

Τελευταία βασική μέθοδος αυτής της κλάσης είναι η ***UpadteLocalnputFile()*** κατά την οποία και πραγματοποιείται η μεταφορά των αρχείων με τα νέα Urls από το τοπικό directory *//Future του Master Node* στο αρχείο εισόδου *//In* του HDFS. Αυτή μέθοδος καλείται μετά το πέρας των *διαδικασιών Fetching, WordCount, RemoveDublicates, Group_Urls, Database* Commition , οπότε και ανεβάζει στο Hdfs νέο block δεδομένων για να ξεκινήσουν εκ νέου οι προαναφερθέντες διαδικασίες.

Α.7 ΑΝΑΛΥΣΗ ΚΛΑΣΗΣ *Crawler.java*

Τα βασικά σημεία της κλάσης συνοψίζονται παρακάτω:

Πρώτο σημείο είναι η μέθοδος **CkeckUrlInsertion ()** όπου και πραγματοποιούνται ουσιαστικά δύο βήματα. Πρώτον, μέσα σε ένα βρόχο επανάληψης ξεκινάμε αφαιρώντας την κεφαλή της ουράς και εξετάζουμε για το συγκεκριμένο Url τα εσωτερικά links και εισάγουμε μέσα σε μία λίστα την τρέχουσες διευθύνσεις . Μέσω αυτής στο τέλος της διαδικασίας θα μεταφέρουμε τα δεδομένα στο αρχείο. Η διαδικασία αυτή επαναλαμβάνεται μέχρι ότου η ουρά να αποκτήσει αριθμό στοιχειών μεγαλύτερο ή ίσο της *MaxServerListSize.* Κατά το δεύτερο βήμα απλά μεταφέρουμε τα δεδομένα από την ουρά στο αρχείο το οποίο και τελικά μεταφέρουμε στο directory εισόδου του HDFS, **/In.**

Μέσω της **OpenUrlConnection (URL_Entity Url,int extract_code)** ανοίγουμε συνδέσεις με τις σελίδες για ένα συγκεκριμένο URL_Entity. Να πούμε βέβαια πως έχουμε ορίσει ένα χρονικό περιθώριο για το οποίο διατηρούμε την επικοινωνία με αυτό το site και αυτό καθορίζεται βάσει της μεταβλητής Timeout – εκφρασμένη σε msec. Αυτό το κάνουμε γιατί ορισμένα sites μπορεί να έχουν ορίσει την παράμετρο **java.net.URLConnection.setReadTimeout** στο άπειρο, οπότε μπορεί να εγκλωβιστούμε σε κάποιο site περιμένοντας να λάβουμε το περιεχόμενο. Με τον τρόπο αυτό λοιπόν αποφεύγουμε τέτοιου είδους προβλήματα.

Η μέθοδος **RetrieveLinks(Url,String text)** αφορά την εξαγωγή των εσωτερικών links από το περιεχόμενο μίας ιστοσελίδας ,το οποίο και δίνεται και σε μορφή String . Για να καταφέρουμε τώρα να απομονώσουμε το ζητούμενο εξετάζουμε μέσα στο σύνολο του κώδικα τα tags "**<a href="** τα οποία και αντιστοιχούν σε links που εισάγονται στο σώμα της ιστοσελίδας. Για κάθε Url που εντοπίζεται γίνεται μέσω της μεθόδου *UrlVisited* της κλάσης *Scheduling* έλεγχος αν πρέπει να τοποθετηθεί στην ουρά των μελλοντικών διευθύνσεων. Έτσι, όταν τελειώσει αυτή η διαδικασία έχουν πραγματοποιηθεί οι έλεγχοι πρώτου επίπεδου.

Τέλος , να επεξηγήσουμε και τον ρολό της μεταβλητής **NumFetchedUrl.** Αυτή η μεταβλητή χρησιμοποιήθηκε με σκοπό να κρατάμε το σύνολο των διευθύνσεων που στέλνουμε για επεξεργασία .Αυτή η μεταβλητή θα μας χρησιμεύσει όπως θα δούμε στην κλάση *Initiate.java* για να ελέγχουμε πόσα sites έχουμε μεταφέρει στο Hdfs σύστημα. Έτσι, αν ξεπεραστεί ένα όριο τότε κάνουμε commit στην βάση του B-tree και αποθηκεύουμε τα δεδομένα σε αυτήν.

Α.7.1 ΨΕΥΔΟΚΩΔΙΚΑΣ :Crawler- CkeckUrlInsertion()

Ψευδοκώδικας:

```
Until( URL_Queue_Size < MaxFetchedUrlNum)
{
        Url =Url_Queue_Poll;
        Open_Url_connection;
        Extract_Urls_From_Url;

}
counter= 0;
Until ( counter<URL_Queue_Size)
{
        Url=Url_Queue_Poll;
         Put_Url_to_HDFS;
       counter++;
}
```

A.8 ΑΡΧΙΚΟΠΟΙΗΣΗ CRAWLER –*INITIATE.JAVA*

Σε αυτή την κλάση έχουμε βάλει ως private μεταβλητές τα πεδία **CrawlDB Crawldb** , **Indexing Btree** .Η καθεμία από αυτές τις μεταβλητές είναι καθολικές για όλη την εργασία μας επομένως δημιουργείται μόνο ένα instance για την καθεμία αφού μέσω αυτών κρατάμε στην μνήμη την δομή δεικτοδότησης . Από την άλλη η **MaxNonCommitedUrl** είναι μια final μεταβλητή και αφορά το σύνολο των Urls που προγραμματίζουμε να επισκεφτούμε μέσω του Hadoop πριν κάνουμε commit στην βάση. Όταν υπερβούμε αυτόν τον αριθμό τότε και μεταφέρουμε τα στοιχεία στην βάση. Να πούμε βέβαια πως αυτός ο αριθμός είναι αρκετά μεγάλος γιατί δε θέλουμε πολύ συχνά να κάνουμε commit καθυστερώντας το σύστημα.

Η διαδικασία ξεκινά από την συνάρτηση main() που καλεί ένα menu όπου και κατασκευάζουμε ένα αντικείμενο της Initiate. Από εκεί δημιουργείται ένα καινούριο instance του B-tree το οποίο και θα διοχετεύσουμε και σε αντικείμενα τύπου Scheduling και όπου άλλου χρειαστεί. Στην συνέχεια ,αρχικοποιούμε τον Crawler εισάγοντας τα πρώτα Urls που θα τοποθετηθούν στην ουρά της κλάσης Scheduling. Από εκεί θα καλέσουμε την συνάρτηση CkeckUrlInsertion() για να ξεκινήσει η διαδικασία εξαγωγής καινούριων ιστοσελίδων από τις διευθύνσεις που εισήγαμε κατά την αρχικοποίηση. Όταν λοιπόν τελειώσει και αυτό , εκείνο που απομένει είναι μεταφέρουμε τα δεδομένα της λίστας στο αρχείο UrlsToUpload.txt απ' όπου θα διοχετευτεί το αρχείο στο HDFS για να ξεκινήσει το fetching των ιστοσελίδων.

Αυτή η διαδικασία λοιπόν με εξαίρεση την αρχικοποίηση του Crawler μπορεί είτε να επαναληφτεί τόσες φορές όσο επιθυμεί ο χρήστης είτε να ορίσει τον χρόνο κατά τον οποίο θέλει να τρέχει. Με λίγα λόγια, το μενού προσφέρει δύο

δυνατότητες: θέτοντας ένα χρονικό διάστημα ή ορίζοντας πόσες φορές επιθυμεί να πραγματοποιηθεί η διαδικασία.

Τι κερδίζουμε επιλέγοντας μια τέτοια αρχιτεκτονική; Εξασφαλίζουμε πως αυτό τμήμα είναι εντελώς ανεξάρτητο από τα επόμενα που θα δούμε γιατί για να ξεκινήσει η επόμενη φάση το μόνο που απαιτείται είναι το αρχείο UrlsToUpload.txt . Έτσι , θέτοντας τον Crawler σε μία μηχανή και σε άλλη το επόμενο τμήμα του fetching και επεξεργασίας των ιστοσελίδων καταφέραμε να λειτουργήσουν ταυτόχρονα τα δύο συστήματα. Επομένως, τον χρόνο που χάσαμε από το να υλοποιήσουμε την πρώτη φάση σε διαδικασιακό προγραμματισμό τον ανακτούμε από την παραλληλία των δύο συστημάτων.

Α.8.1 ΑΛΓΟΡΙΘΜΟΣ ΠΡΩΤΗΣ ΦΑΣΗΣ *Crawling*

```
Insert_Urls_URL_Queue;                          //Initiate Crawler
Until (user_defines_stop)
{     Extract_URLS_Until_Queue <MaxServerListSize    //CkeckUrlInsertion
Update_ UrlsToUpload;
If ( NumFetchedUrl  >MaxNonCommitedUrls )
Comit_Btree
}
```

Α.9 Ανάλυση κλάσης TextAnalyser.java

Στην συγκεκριμένη κλάση κάνουμε τον έλεγχο για τα StopWordrds. Για να μπορέσουμε λοιπόν να κάνουμε αυτήν την διαδικασία γρήγορα εντάξαμε το σύνολο των StopWords σε ένα Hash Table ώστε η αναζήτηση για το κλειδί –StopWords να γίνεται γρήγορα χωρίς την σειριακή αναζήτηση μέσα σε πίνακα –λίστα κλπ. Αυτό τον ρόλο έχει και η Private μεταβλητή **StopHash**.

Για να αρχικοποιήσουμε τώρα το HashTable έπρεπε να μεταφέρουμε τα δεδομένα από το αρχείο ***stoplist.txt*** που βρίσκεται στο directory του project μας . Αυτή η διαδικασία γίνεται μέσα στον constructor της κλάσης καλώντας την **InitiateStopWordList() .**

Για να ελέγξουμε τώρα αν μία λέξη κατατάσσεται σε Stop Word χρησιμοποιήσαμε την **Removestopwords(String word).** Σε αυτή την μέθοδο αυτό που κάναμε είναι να μετατρέψουμε αρχικά την δεδομένη λέξη σε Lowercase, δηλαδή τα κεφαλαία σε πεζά , και ως δεύτερο βήμα να εξαλείψουμε τα σημεία στίξης που πιθανών να έχουν μείνει κατά την εξαγωγή του κειμένου από τον parser. Στην συ-

νέχεια, ελέγχουμε αν ο δεδομένος όρος υπάρχει καταχωρημένος μέσα στο HashTable και αναλόγως επιστρέφουμε τον όρο ή “” για να υποδείξουμε να μην χρησιμοποιηθεί για περεταίρω επεξεργασία.

Α.10 ΨΕΥΔΟΚΩΔΙΚΑΣ Fetching

Πριν συνοψίσουμε τα βήματα να δούμε την μέθοδο run() της Map_Reduce_Crawler απ’ όπου και ξεκινά η διαδικασία Map/Reduce.

Παραθέτουμε τα βασικά σημεία για να δούμε πως αρχικοποιείται μία Hadoop Job.

```
1.  JobConf conf=new JobConf();
2.  conf = new JobConf(Map_Reduce_Crawler.class);
3.  conf.setJobName("MapReduce");
4.  conf.setInputFormat(TextInputFormat.class);
5.  conf.setOutputFormat(TextOutputFormat.class);
6.  //set output formats
7.  conf.setOutputKeyClass(Text.class);
8.  conf.setOutputValueClass(Text.class);
9.  conf.setMapperClass(Map_Reduce_Crawler.class);
10. conf.setCombinerClass(Reduce_Page_Store.class);
11. conf.setReducerClass(Reduce_Page_Store.class);
12. FileInputFormat.setInputPaths(conf, new Path("In\\"));
13. FileOutputFormat.setOutputPath(conf, new Path("Out\\Page_Output"));
```

Από τα παραπάνω βλέπουμε πως για να δώσουμε τις αρχικές παραμέτρους χρησιμοποιούμε την μεταβλητή τύπου **JobConf .** Έτσι, στην 3 δίνομε όνομα στην εργασία που θα ξεκινήσει, στην 4-5 ορίζουμε τι τύπο θα έχουν οι είσοδοι και έξοδοι του προγράμματος μας ,στις 7-8 ορίζουμε τι τύπο θα έχουν τα πεδία Key,Value στους Mapper , Reducer ενώ στις 9-11 ορίζουμε ποιες κλάσεις του προγράμματος μας υλοποιούν αντίστοιχα αυτές τις μεθόδους του Hadoop αντίστοιχα. Τέλος, στις 12-13 δίνουμε το Path για τα αρχεία εισόδου και εξόδου στο Hdfs.

Ψευδοκώδικας;

Mapper

```
For_Each_Input_Line {
```

```
Url=value;
Open_Url_Connection;
S=Take_Page_Content;
Control=Check_Crawldb(Url);
If(control==ok)
{
   Store_S_Locally;
  Text=Extract_Text_From_S;
   Extract_Internal_Urls();
  Collect(Url, Text);
}
}
                                                                    Reducer
For_Each_Cluster_with_Key{

While_Cluster_Has_Values{
    Words=Tokenize_Value;
    While(Words!=null)
          {
                stop=Check_StopWord(current_word);
                if(stop!=null)
                   Collect(key,Stem(current_word));
        }

}
```

Α.11 ΨΕΥΔΟΚΩΔΙΚΑΣ Κλάσης *WordCount.java*

Ψευδοκώδικας:

```
                                                                    Mapper

For_Each_Input_Line {

Url_Word=value;
Collect(Url_Word, new IntWritable(1));
}
                                                                    Reducer
For_Each_Cluster_with_Key{
int sum=0;
While_Cluster_Has_Values{
       sum = sum+values.hasNext ();
  }
Int tfi=sum;
 Collect ( key, new IntWritable(  tfi ));

}
```

A.11.1 WordCount- Run ()

Παραθέτουμε την υλοποίηση της μεθόδου WordCount.run() για να επεξηγήσουμε κάποια σημεία που έχουν διαφοροποιηθεί σε σχέση με την προηγούμενη Map/Reduce Job.

```
1.  JobConf conf = new JobConf(WordCount.class);
2.  conf.setJobName("wordcount");
3.  conf.setOutputKeyClass(Text.class);
4.  conf.setOutputValueClass(IntWritable.class);
5.  conf.setMapperClass(Map.class);
6.  conf.setCombinerClass(Reduce.class);
7.  conf.setReducerClass(Reduce.class);
8.  conf.setInputFormat(TextInputFormat.class);
9.  conf.setOutputFormat(TextOutputFormat.class);
10. FileInputFormat.setInputPaths(conf, new Path("Out\\Page_Output\\"));
11. FileOutputFormat.setOutputPath(conf, new Path("Out\\Output"));
12. JobClient.runJob(conf);
```

Σε αυτή την περίπτωση αυτό που παρατηρούμε πως αλλάζει είναι πως δίνουμε 2 φορές τί τύπου θα είναι τα πεδία Key-Value. Αυτό το κάνουμε γιατί ο Mapper και ο Reducer δεν εξάγουν στην έξοδό τους τα στοιχεία με τον ίδιο τύπο δεδομένων. Ειδικότερα, κατά την φάση του Mapping θέλουμε στην έξοδο να μεταφέρουμε το key ως *Text* και το Value ως *IntWritable*- (την τιμή *1*). Αυτό ορίζουμε στις γραμμές 3-4. Αντίθετα, στην φάση του Reduce κατά την έξοδο θέλουμε τα πεδία key,value να είναι τύπου *Text(*γραμμές 8-9*).* Τέλος, στις γραμμές 5-7 ορίζουμε τις κλάσεις των Map/Reduce και οι οποίες είναι εσωτερικές κλάσεις της *WordCount.*

A.13 ΨΕΥΔΟΚΩΔΙΚΑΣ Κλάσης *Word_IDF.java*

Ψευδοκώδικας:

Mapper

```
For_Each_Input_Line {

line =value;
parts[]=line. split(/t);

if(  parts. length==3)
        Collect(parts[1], new FloatWritable(1));
```

```
}
                                                                    Reducer
For_Each_Cluster_with_Key{
int sum=0;
While_Cluster_Has_Values{
        sum = sum+values.hasNext ();
  }
Float  IDF=log D/sum;
 Collect ( key, new FloatWritable(  IDF ));

}
```

A.14 ΨΕΥΔΟΚΩΔΙΚΑΣ *Κλάσης* Doc_Weights *.java*

Ψευδοκώδικας:

```
                                                                    Mapper

For_Each_Input_Line {

line =value;
parts[]=line. split(/t);

if (  parts. length==3)
   Collect(parts[0],parts[1]+"\t"+parts[2]);  //For Each Site Collect Input Lines
}
                                                                    Reducer
For_Each_Cluster_with_Key{
float w=0;
While_Cluster_Has_Values
{
        line= values.hasNext ();                    //Calculate Url Weight
        part[]=line. split(\t);
         w = w+ (part[2]*Btree(part[1]))² ;
 }
   collect ( key, √w);                              //Export key Concatenated To Weight

}
```

A.15 ΨΕΥΔΟΚΩΔΙΚΑΣ *Κλάσης Vector-Model_Queries .java*

Ψευδοκώδικας:

Mapper

```
Mapper_Constructor (

  QueryTable[]=User_Query;

)

For_Each_Input_Line {

parts[]=value.split(\t);
for(int i=0;i<Querytable.length;i++)
 {
          If(parts[1]==QueryTable[i])
            Collect(parts[0], parts[1]+"\t"+parts[2]);
 }
}
```

Reducer

```
Reducer_Constructor(
          Qw= QueryTable[QueryTable.length-1];

)
For_Each_Cluster_with_Key{
float QW=0;
While_Cluster_Has_Values{
        line= values. hasNext();
        parts[]= line. split(\t);
        QW = QW+ IDfTree(parts[0)]2 *parts[1];
  }
float sim=QW/Qw*WeightTree(key);
Collect ( key,sim );

}
```

Βιβλιογραφία

ACM.org. (n.d.). *MapReduce: A Flexible Data Processing Tool.* Retrieved from http://cacm.acm.org/magazines/2010/1/55744-mapreduce-a-flexible-data-processing-tool/fulltext

Apache Software Foundation. (n.d.). *TroubleShooting*. Retrieved from http://wiki.apache.org/hadoop/TroubleShooting

Apache-Software Foundation. (2008). Retrieved from http://hadoop.apache.org/common/docs/current/mapred_tutorial.pdf

Apache-Software Foundation. (n.d.). *Hadoop Wiki.* Retrieved from http://wiki.apache.org/hadoop/WordCount

Coharian,Grossman, Frieder. (2002,2009). Retrieved from http://www.ir.iit.edu/~dagr/cs529/files/handouts/03VectorSpaceImplementation-6per.PDF

Dr. E. Garcia. (2006 , 10 27). *The Classic Vector Space Model.* Retrieved from http://www.miislita.com/term-vector/term-vector-3.html

Dustin Boswell. (2003, 12). *Distributed High-Performance Web Crawlers.* Retrieved from http://cseweb.ucsd.edu/~dboswell/PastWork/WebCrawlingSurvey.pdf

E.G.M Petrakis. (n.d.). Retrieved from http://www.intelligence.tuc.gr/~petrakis/courses/multimedia/classic.pdf

E.G.M Petrakis. (n.d.). Retrieved 12 2009, from http://www.intelligence.tuc.gr/~petrakis/courses/datastructures/btrees.pdf

Feygym Cao, D. J. (n.d.). *Scheduling Web Crawl for Better Performance and Quality.* Retrieved 2009, from ftp://ftp.cs.princeton.edu/techreports/2003/682.pdf

G.Noll, M. (2009). *Running Hadoop On Ubuntu Linux*. Retrieved 12 1, 2009, from http://www.michael-noll.com/wiki/Running_Hadoop_On_Ubuntu_Linux_(Multi-Node_Cluster)

Java.net-Introduction To Nutch. (n.d.). *The Source For java Technology Collaboration.* Retrieved from http://today.java.net/pub/a/today/2006/01/10/introduction-to-nutch-1.html

Jdbm Project. (n.d.). Retrieved from http://jdbm.sourceforge.net/

Stemming. (n.d.). *Wikipedia*. Retrieved from http://en.wikipedia.org/wiki/Stemming

Suel, V. S. (n.d.). *Design and Implementation of a High-Performance Distributed Crawler.* Retrieved 2009, from http://cis.poly.edu/suel/papers/crawl.pdf

The Java Tutorials. (n.d.). *JDBC Introduction.* Retrieved from http://java.sun.com/docs/books/tutorial/jdbc/overview/index.html

The Java Tutorials-Custom Networking. (n.d.). *Reading from and Writing to a URLConnection*. Retrieved from http://java.sun.com/docs/books/tutorial/networking/urls/readingWriting.html

Timothy W Macinta's Website Menu . (n.d.). *Fast MD5 Implementation In Java.* Retrieved from http://www.twmacinta.com/myjava/fast_md5.php

White, T. *The Definitive Guide.*

Wikepedia. (n.d.). *B-Tree.* Retrieved 12 2009, from http://en.wikipedia.org/wiki/B-tree

Wikipedia. (n.d.). *MD5 Algorithm.* Retrieved from http://en.wikipedia.org/wiki/MD5

Yahoo Developer Network. (n.d.). *Hadoop Tutorial YDN.* Retrieved from http://developer.yahoo.com/hadoop/tutorial/module4.htmlTimothy W Macinta's Website Menu