Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μεταπτυχιακή Διατριβή
στα Πληροφοριακά Συστήματα

Αποδοτικές τεχνικές ανίχνευσης πελατών με παρόμοια καταναλωτική συμπεριφορά

Στυλιανός Μουλακάκης

Επιβλέπων Καθηγητής
Ματθαίος Δαμίγος

Οκτώβριος 2012

Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Αποδοτικές τεχνικές ανίχνευσης πελατών με παρόμοια καταναλωτική συμπεριφορά

Στυλιανός Μουλακάκης

Επιβλέπων Καθηγητής
Ματθαίος Δαμίγος

Η παρούσα μεταπτυχιακή διατριβή υποβλήθηκε
προς μερική εκπλήρωση των απαιτήσεων για απόκτηση

μεταπτυχιακού τίτλου σπουδών
στα Πληροφοριακά Συστήματα

από τη Σχολή Θετικών και Εφαρμοσμένων Επιστημών
του Ανοικτού Πανεπιστημίου Κύπρου

Οκτώβριος 2012

Περίληψη

Η συγκεκριμένη μεταπτυχιακή διατριβή έχει ως σκοπό τον εντοπισμό αποδοτικών τεχνικών για τον προσδιορισμό χρηστών με παρόμοια συμπεριφορά. Εστιάζοντας σε χρήστες ηλεκτρονικών καταστημάτων, στοχεύουμε μέσω της ανάλυσης που παρουσιάζεται, να εντοπίσουμε χρήστες με παρόμοια καταναλωτική συμπεριφορά, ή διαφορετικά, παρόμοιους καταναλωτές. Η ανάγκη αυτή, εμφανίζεται κατά τη σχεδίαση και δημιουργία εφαρμογών ηλεκτρονικού εμπορίου, αλλά και σε πολλές ηλεκτρονικές εφαρμογές, που παρέχουν ποικίλα είδη ηλεκτρονικής πληροφορίας.

Το γενικότερο πρόβλημα, με το οποίο σχετίζεται το πρόβλημα εύρεσης παρόμοιων χρηστών, είναι αυτό του εντοπισμού και της πρότασης σε χρήστες μίας εφαρμογής, πληροφορίας που "μπορεί να τους ενδιαφέρει", ή διαφορετικά της δημιουργίας συστημάτων συστάσεων (recommendation systems). Στις περισσότερες ηλεκτρονικές εφαρμογές, οι προτάσεις αυτές βασίζονται στην καταγεγραμμένη συμπεριφορά του χρήστη, καθώς και σε στοιχεία που παρέχει ο χρήστης, ανάλογα με τις προτιμήσεις και τα ενδιαφέροντά του.

Καθώς η χρήση των ηλεκτρονικών υπηρεσιών του Web αποτελεί ένα καθιερωμένο τρόπο αναζήτησης πληροφορίας, όπως και ανάπτυξης εμπορίου καταναλωτικών αγαθών, τα δεδομένα που καταγράφονται και διαχειρίζονται από ηλεκτρονικές εφαρμογές, αυξάνουν ραγδαία, τείνοντας σε πολλές περιπτώσεις να καταρρίπτουν καθιερωμένους τρόπους επεξεργασίας τους. Αυτό έχει ως αποτέλεσμα, ο μεγάλος όγκος της διαθέσιμης πληροφορίας που διαχειρίζεται μία ηλεκτρονική εφαρμογή, να δημιουργεί την ανάγκη εύρεσης νέων μεθόδων επεξεργασίας, οι οποίες είναι επεκτάσιμες ως προς τον όγκο των δεδομένων που διαχειρίζονται. Σε αυτό το πλαίσιο, οι τεχνικές που εξετάζονται και προτείνονται από την συγκεκριμένη εργασία, βασίζονται στην διαχείριση μεγάλου όγκου δεδομένων, και ειδικότερα, βασίζονται στην χρήση της παράλληλης υποδομής Map Reduce, που αναπτύχθηκε από την Google.

Η συγκεκριμένη διατριβή παραθέτει μία λεπτομερή ανάλυση αλγορίθμων για τον εντοπισμό όμοιων στοιχείων (finding similar items) σε μία συλλογή δεδομένων, θεωρώντας μεγάλο όγκο διαθέσιμων δεδομένων. Επίσης προτείνεται ο αλγόριθμος MinHashingMR για την εύρεση όμοιων στοιχείων κάνοντας χρήση του περιβάλλοντος Map Reduce. Ειδικότερα, αναλύονται οι σημαντικότεροι αλγόριθμοι εύρεσης όμοιων στοιχείων σε περιβάλλον Map Reduce, παρουσιάζονται τα κύρια χαρακτηριστικά του περιβάλλοντος Map Reduce, διατυπώνεται ο αλγόριθμος MinHashingMR και υλοποιείται με την χρήση του συστήματος Apache Hadoop, το

οποίο αποτελεί το ελεύθερο σύστημα προσομοίωσης του περιβάλλοντος Map Reduce. Η υλοποίηση του αλγορίθμου συνοδεύεται από την παροχή μίας λεπτομερούς ανάλυσης αποτελεσμάτων, σε πραγματικά δεδομένα, και από την παρουσίαση της γενικής αρχιτεκτονικής συστημάτων που κάνουν χρήση της συγκεκριμένης προσέγγισης.

Τέλος, σύμφωνα με τη μελέτη της συγκεκριμένης εργασίας, αποδεικνύεται ότι η χρήση του αλγορίθμου MinHashingMR, που δίνει προσεγγιστικές λύσεις του προβλήματος, βελτιώνει σημαντικά τον χρόνο εντοπισμού όμοιων στοιχείων σε σύγκριση με προηγούμενους αλγόριθμους επίλυσης του προβλήματος, όπως επίσης και σε σχέση με τον σειριακό του πρόγονο, τον αλγόριθμο MinHashing.

Summary

This thesis aims to identify effective techniques in order to find users with similar purchasing behavior, or similarly, similar consumers, especially focusing on e-shop users. This need occurs at the design and creation of e-commerce, but also in many electronic applications, providing various types of electronic information.

The general problem, which is associated with the problem of finding similar users, is to identify and propose information that a user "may be interested", or similarly, create recommendation systems to application users. In most electronic applications, such suggestions are based on the recorded user behavior, and on information supplied by the user, depending on their preferences and interests.

As the use of Web services is an established way to search for information, as well as to develop the trade of consumer goods, the data is recorded and handled by applications which are rapidly increasing. As a result, the large volume of information which has to be managed creates the need to find new treatment methods scalable to amounts of data. The techniques discussed and proposed by the particular job, are based on handling large amounts of data, and in particular, based on the use of Map Reduce, developed by Google.

This thesis gives a detailed analysis of algorithms for identifying similar items on a collection of large volume of available data. MinHashingMR is a proposed algorithm for finding similar items by means of Map Reduce environment. The most important algorithms in finding similar items in Map Reduce environment are being analyzed and the main characteristics of this environment are being presented. The MinHashingMR algorithm has been formulated and implemented by the use of the Apache Hadoop, which is the freeware, well-known, alternative to Google's Map Reduce infrastructure. The implementation of the algorithm comes with a detailed results analysis, on real data, and the presentation of the overall architecture of systems that make use of this approach.

Finally, it is proved that the use of the MinHashingMR algorithm gives solution to the problem of finding elements common in approximation. Besides it improves greatly time tracking, compared with other algorithms, as well as in connection with its ancestor series algorithm MinHashing.

Ευχαριστίες

Ευχαριστώ τον επιβλέποντα καθηγητή μου Ματθαίο Δαμίγο για την καθοδήγηση και το υλικό που μου παρείχε όλο το προηγούμενο διάστημα.

Ευχαριστώ το Νίκο Στασινόπουλο για τις βοήθειες που παρείχε σε θέματα κώδικα και αλγορίθμων.

Ευχαριστώ το συμφοιτητή μου Βασίλη Τριζώνη για την παρακίνηση, τη βοήθεια και τη συνεργασία.

Προπάντων, ευχαριστώ τη σύζυγό μου Χρυσούλα και τον δυόμισι ετών γιο μου Τηλέμαχο, για την υπομονή που έκαναν καθ' όλη τη διάρκεια.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1
ΕΙΣΑΓΩΓΗ

1.1 ΓΕΝΙΚΑ.

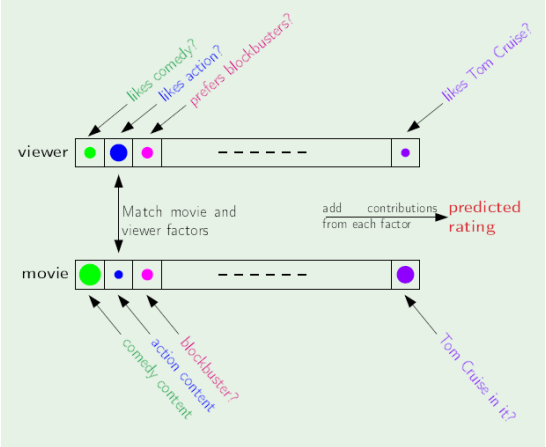
Ολοένα και περισσότερο διαδίδεται τόσο στην παρουσία, όσο και στη χρήση της στο διαδίκτυο, η προσωποποιημένη πλοήγηση. Κάθε χρήστης αποτελεί ξεχωριστή οντότητα, με συγκεκριμένες επιλογές, συγκεκριμένη συμπεριφορά και φυσικά συγκεκριμένες απαιτήσεις. Και με τον όρο «συγκεκριμένο», εννοούμε ακριβώς αυτό που πλέον έχουμε λίγο έως πολύ, όλοι όσοι χρησιμοποιούμε το διαδίκτυο για οποιοδήποτε λόγο, ένα συγκεκριμένο username – ταυτότητα χρήστη, που συνοδεύει - αν όχι όλες – τις περισσότερες κινήσεις, αναζητήσεις και επιλογές μας στο internet.

Το παραπάνω έχει ενταχθεί στην ευρύτερη και ραγδαία αναπτυσσόμενη έννοια του cloud computing, αυτό που στα Ελληνικά ονομάζουμε «σύννεφο». Με τη χρήση του από τους internet χρήστες, ακολουθεί και μια σειρά «παροχών» από διαδικτυακούς τόπους, που διαμορφώνονται τόσο με βάση το «διαδικτυακό παρελθόν» του χρήστη, την προηγούμενη δηλαδή δραστηριότητά του κινούμενος στο διαδίκτυο και εξελίσσεται στην - κατά ένα τρόπο - διαμόρφωση πρόγνωσης για την αντίστοιχη μελλοντική του δραστηριότητα. Για να συμβεί το παραπάνω, απαιτείται η εκπλήρωση μιας σειράς γεγονότων, άμεσα εξαρτώμενων από τον ίδιο το χρήστη.

Αρχικά χρειάζεται η δημιουργία μιας βάσης δεδομένων με προσωποποιημένα χαρακτηριστικά χρήστη, όπως για παράδειγμα, εκτός από το username του, επιλογές σελίδων, επιλογές προϊόντων, ή τα clicks που έχει πραγματοποιήσει. Όσο περισσότερο μεγαλώνει αυτή η βάση, τόσο πιο συγκεκριμένες μπορούν να γίνονται οι προτάσεις από ένα σύστημα που θα μπορεί να «διαβάσει» τη μελλοντική συμπεριφορά του χρήστη.

Ο σκοπός της ανίχνευσης, του «διαβάσματος» δηλαδή, της μελλοντικής συμπεριφοράς του χρήστη, θα χρησιμοποιηθεί προκειμένου να δημιουργηθούν προβλέψεις για τυχόν μελλοντικές διαδικτυακές ανάγκες του. Η δυνατότητα δημιουργίας προβλέψεων, οδηγεί με τη σειρά της στη δυνατότητα πραγματοποίησης συστάσεων (recommendations) στους χρήστες, αντικείμενο που αποτελεί και το ζήτημα της παρούσας εργασίας. Η ουσία δηλαδή της δημιουργίας προβλέψεων, είναι η κατασκευή ενός προτύπου, επάνω στο οποίο θα αναπτυχθεί η επόμενη διαδικασία, αυτή της πραγματοποίησης συστάσεων.

Ας δούμε ένα παράδειγμα με θέμα την πρόβλεψη του τρόπου που οι χρήστες θα αξιολογήσουν μία προτεινόμενη ταινία[19]. Έχουμε στη διάθεσή μας, από τη μία πλευρά, το προφίλ ενός χρήστη, με διάφορα χαρακτηριστικά του, όπως για παράδειγμα εάν του αρέσουν οι κωμωδίες, ή εάν του αρέσουν οι ταινίες δράσης, αντίστοιχα εάν του αρέσουν ταινίες από τα blockbusters, ή και πιο εξειδικευμένα, αν του αρέσει ο Tom Cruise και από την άλλη πλευρά, το προφίλ μιας ταινίας με αντίστοιχα χαρακτηριστικά, κατά πόσο είναι κωμωδία, κατά πόσο είναι ταινία δράσης, κατά πόσο ανήκει στο blockbuster και αν παίζει σε αυτήν ο Tom Cruise και εντάξουμε στο όλο αυτό σχέδιο και ένα τρόπο βαθμονόμησης, όπως αυτός που εικονίζεται στο σχήμα παρακάτω (σχήμα 1.1), η μηχανή θα μπορεί να εξετάσει και αντίστοιχα να κρίνει, εάν η συγκεκριμένη ταινία μπορεί να προταθεί στο συγκεκριμένο χρήστη.



Σχήμα 1.1. Αξιολόγηση χαφακτηριστικών ταινίας και χρήστη. Όσο πιο έντονο είναι ένα χαφακτηριστικό, τόσο πιο μεγάλος είναι ο έγχρωμος κύκλος.

Σε ένα δεύτερο παράδειγμα, θα μπορούσαμε να χρησιμοποιήσουμε, ένα σύστημα για να βρούμε ποια είναι τα χαρακτηριστικά που θα πρέπει να εκπληρώνει ένας χρήστης, προκειμένου να λάβει πίστωση ποσού και τι αξίας θα είναι αναλόγως το ποσό αυτό.

Έστω ότι στο σύστημα θα βρίσκονται καταχωρισμένοι χρήστες – πελάτες με την μορφή που εικονίζεται στο παρακάτω σχήμα 1.2.

age	23 years
gender	male
annual salary	$30,000
years in residence	1 year
years in job	1 year
current debt	$15,000
...	...

Σχήμα 1.2. Τα χαρακτηριστικά της καρτέλας του πελάτη.

Και το ερώτημα που έχει να απαντήσει η μηχανή, είναι: «Θα πρέπει να του χορηγηθεί ποσό; Και αν ναι, πόσο;».

1.2 Recommendation Systems.

Στην καθημερινή χρήση του internet, σίγουρα έχει τύχει να δεχτούμε «διαδικτυακή εισήγηση», σχετικά με κάποιο θέμα, ή κάποια σελίδα, ή κάποιο προϊόν που εκτιμάται από το σύστημα ότι μπορεί να μας ενδιαφέρει.

Για να δημιουργηθεί ένα σύστημα που θα κάνει εισηγήσεις σε χρήστες, απαιτείται, τίθεται ως προϋπόθεση, να έχει δοθεί στη διάθεση του, «υλικό» που θα το καταστήσει ικανό να γνωρίζει, ή να υποθέτει, τι είναι κατάλληλο και τι όχι στον εκάστοτε χρήστη, ώστε να το προτείνει, να το εισηγηθεί σε αυτόν. Τέτοια συστήματα καλούνται Recommendation Systems και σε μία ελληνική μετάφραση θα τα ονομάζουμε Συστήματα Εισηγήσεων [2, 11].

1.3 ΤΟ ΠΡΟΒΛΗΜΑ.

Αυτό που επιχειρούμε να κάνουμε σε αυτή την εργασία, είναι να περιγράψουμε, να αναλύσουμε και να δημιουργήσουμε ένα πειραματικό μοντέλο ενός Συστήματος Εισηγήσεων, το οποίο θα αναζητά ομοιότητες μεταξύ χρηστών ως προς την καταναλωτική τους συμπεριφορά κατά προσέγγιση. Θα εντοπίζει δηλαδή χρήστες – καταναλωτές, οι οποίοι έχουν προμηθευτεί προϊόντα και ανάλογα με τον αριθμό των προϊόντων που έχουν από κοινού προμηθευτεί θα τους κατατάσσει ως όμοιους ή όχι, χωρίς όμως να απαιτείται ακρίβεια στον προσδιορισμό αυτό.

Εάν για παράδειγμα έχουμε τους καταναλωτές και τα προϊόντα που φαίνονται στον παρακάτω πίνακα 1, χρησιμοποιώντας το «1» για κάθε προϊόν που έχει προμηθευτεί ο κάθε καταναλωτής, έχουμε πλήρη εικόνα όλων των αγορών που έχει πραγματοποιήσει. Εάν στη συνέχεια θέσουμε ως προϋπόθεση ομοιότητας ότι: όμοιοι είναι δύο καταναλωτές οι οποίοι έχουν προμηθευτεί τουλάχιστον δύο προϊόντα από κοινού, τότε σαν αποτέλεσμα της αναζήτησης θα μας εμφανιστούν οι καταναλωτές Α, Β και Ε, οι οποίοι έχουν από κοινού προμηθευτεί τα προϊόντα α και γ. Αυτό που έχει να κάνει στη συνέχεια ένα σύστημα εισηγήσεων, είναι να προτείνει τα υπόλοιπα προϊόντα που δεν είναι κοινά στου όμοιους καταναλωτές. Αυτό σημαίνει ότι θα προτείνει στον Α το β και το δ, το ίδιο και στον Ε.

	Προϊόντα					
Καταναλωτής	α	β	γ	δ	ε	ζ
Α	1		1			
Β	1	1	1	1		
Γ		1				
Δ				1		1
Ε	1		1		1	

Πίνακας 1.

Το μοντέλο που θα μελετήσουμε, ζητούμε να έχει τα ακόλουθα χαρακτηριστικά:

1. Θα χρησιμοποιεί υπολογιστικό σύστημα, με περισσότερους από ένα κόμβους (ηλεκτρονικούς υπολογιστές), για να πετύχουμε παράλληλη επεξεργασία δεδομένων. Ο λόγος που θα το επιδιώξουμε αυτό, είναι για την επίτευξη ταχύτερης, ταυτόχρονης επεξεργασίας δεδομένων και επομένως ταχύτερης εξαγωγής αποτελεσμάτων.

2. Το αποτέλεσμα της αναζήτησης να είναι προσεγγιστικό. Μας ενδιαφέρει να εντοπίζουμε κάθε φορά καταναλωτές, με όχι πλήρως όμοιες αγορές, αλλά με κάποιο ποσοστό ομοιότητας

(παρακάτω το ποσοστό αυτό ομοιότητας θα το ορίσουμε ως Threshold). Με τον τρόπο αυτό υπάρχει απώλεια της καθολικής ομοιότητας, ίσως χαθούν και κάποιοι όμοιοι καταναλωτές κατά την προσπέλαση, υπάρχει όμως το περιθώριο να έχουμε πιο ανοιχτά αποτελέσματα και επομένως οι συστάσεις να αφορούν περισσότερους χρήστες, ή και να τους δίνουν μια νέα οπτική. Αν για παράδειγμα ένας χρήστης που ενδιαφέρεται κατά κύριο λόγο για αγορά ρούχων εντοπιστεί ότι ομοιάζει με ένα χρήστη που αντίστοιχα κατά κύριο λόγο ενδιαφέρεται για ταξίδια, είναι πιθανό και οι δύο να ενδιαφέρονται για βαλίτσες. Επομένως η προσεγγιστική ομοιότητα μας ενδιαφέρει προκειμένου να διευρυνθεί η γκάμα ενδιαφέροντος του χρήστη.

3. Χρησιμοποιώντας την τεχνική Map Reduce. Το Map Reduce είναι μια τεχνική η οποία έχει προσεγγιστεί ιδιαίτερα, αλλά μέχρι τώρα δεν υπάρχει κάποια πλήρης υλοποίηση. Η τεχνική αναπτύσσεται πλήρως στο κεφάλαιο 2 του παρόντος.

4. Χρησιμοποιώντας ως πλατφόρμα υλοποίησης της Map Reduce διαδικασίας, το Hadoop. Αντίστοιχα και η πλατφόρμα του Hadoop αναπτύσσεται πλήρως στο κεφάλαιο 2.

1.4 Η ΕΡΓΑΣΙΑ.

Για την υλοποίηση της εργασίας, μελετάται και αναλύεται η τεχνική του Map Reduce. Αντίστοιχα μελετώνται και αναλύονται επιμέρους τεχνικές, όπως για παράδειγμα, η εύρεση της ομοιότητας με απόσταση, ή η διασπορά των δεδομένων σε όλους τους κόμβους που εργάζονται για να εκπληρώσουν τις εργασίες Map Reduce. Επίσης, μελετάται το εργαλείο Hadoop, αφού μέσω αυτού θα γίνει και η υλοποίηση.

Στην κατηγορία αλγορίθμων που θα χρησιμοποιήσουμε προκειμένου να επιτύχουμε αυτή την προσεγγιστική ομοιότητα και στο επίπεδο που απαιτείται, χρησιμοποιούμε ως βάση, ένα τύπο ομαδοποίησης το MinHash. Η μέθοδος MinHashing είναι μια μέθοδος ομαδοποίησης πιθανοτήτων. Η βασική ιδέα στο MinHashing είναι να μετατίθενται στιγμιαία τα set των αντικειμένων και για κάθε χρήστη u_i να υπολογίζεται μια τιμή που θα ονομάζουμε «τιμή hash». Πρόκειται ουσιαστικά για τη μέθοδο και τον αλγόριθμο που χρησιμοποιήσαμε προκειμένου να αναπτύξουμε τον αλγόριθμο αυτής της εργασίας, τον **MinHashingMR**, για να εξαγάγουμε τα

συμπεράσματα για την καταλληλότητα χρήσης του σε συνθήκες αναζήτησης ομοιότητας κατά προσέγγιση.

Μελετώνται, επίσης, σχετικοί αλγόριθμοι, προκειμένου να καταλήξουμε στον Min Hashing, ως τον πιο κατάλληλο για το συγκεκριμένο είδος υλοποίησης που επιδιώκουμε, αφού με την κατάλληλη παραμετροποίησή του αποτελεί τον καλύτερο τρόπο εξαγωγής αποτελεσμάτων ομοιότητας κατά προσέγγιση. Όπως φαίνεται και παρακάτω στα αποτελέσματα, η βελτίωση σε χρόνο είναι πολύ μεγάλη, και έχουμε και το περιθώριο εφαρμογής ενός τέτοιου συστήματος, όταν υπάρχει η ανάγκη άμεσης εξαγωγής αποτελεσμάτων, σε δεδομένα που μεταβάλλονται γρήγορα, ενδεχομένως γρηγορότερα από το χρόνο που θα απαιτούταν να επεξεργαστούν αυτά, σειριακά, από μία και μόνο μηχανή (δυναμικά μεταβαλλόμενα). Η καταλληλότερη τεχνική γι' αυτό, είναι το collaborative filtering, αλλά σπάνια έχουν αναφερθεί μελέτες για πραγματικά μεγάλο αριθμό δεδομένων (αρκετά εκατομμύρια χρηστών και αντικειμένων) και δυναμικά μεταβαλλόμενο (το set δεδομένων που υπόκειται σε επεξεργασία, διαρκώς μεταβάλλεται). Συνήθως οι χρήστες αναζητούν κάποιο θέμα, με κλειδί αναζήτησης τη φράση «δείξε μου κάτι που να με ενδιαφέρει». Σε τέτοιες ακριβώς περιπτώσεις η παρουσίαση συστάσεων στους χρήστες, που να βασίζονται στα ενδιαφέροντά τους, όπως αυτά έχουν καταγραφεί στο παρελθόν, τόσο από τη δραστηριότητά τους, κυρίως από sites κοντινού ενδιαφέροντος που έχουν ανατρέξει, αποτελεί το αντικείμενο της έρευνάς μας. Περισσότερα όμως για την τεχνική του collaborative filtering, αναλύονται παρακάτω, στην παράγραφο 2.3.

Το βέλτιστο αποτέλεσμα επιτυγχάνεται με τη χρήση μιας ομάδας αλγορίθμων που έχουν κατηγοριοποιηθεί ως Model-based αλγόριθμοι. Πρακτικά βασίζονται στην δημιουργία προβλέψεων και συστάσεων, με το να χρησιμοποιούν τις προηγούμενες προβλέψεις των χρηστών, για να προβλέψουν τις αξιολογήσεις σε αντικείμενα που οι χρήστες δεν έχουν ακόμα δει. Περισσότερα για την κατηγορία αυτή αλγορίθμων, αναφέρονται στην παράγραφο 2.3.3.

Απαιτήθηκε η εγκατάσταση και δημιουργία κατάλληλου περιβάλλοντος προκειμένου να διεξαχθούν οι μετρήσεις. Προκειμένου να πάρουμε ένα δείγμα λειτουργίας σε επίπεδο ενός υπολογιστή ώστε να μπορούμε στη συνέχεια να το συγκρίνουμε με αντίστοιχες μετρήσεις σε ένα υπολογιστικό σύστημα, με περισσότερους από ένα κόμβους, δημιουργήσαμε εικονικό περιβάλλον για να λειτουργήσει το Hadoop, λεπτομέρειες του οποίου παρατίθενται παρακάτω.

Η αξιοποίηση χαρακτηριστικών του Hadoop που το κάνουν ξεχωριστό και το καταλληλότερο για τέτοιου είδους εργασίες, είναι αναγκαία. Το σπουδαιότερο ίσως από αυτά τα χαρακτηριστικά

είναι η χρήση και διαχείριση της Distributed Cache. Πρόκειται για τη μέθοδο διασποράς του υπό επεξεργασία αρχείου δεδομένων (datafile) σε όλους τους κόμβους, προκειμένου να γίνεται ταχύτερα και με τη λιγότερη δυνατή χρήση υπολογιστικής ισχύος, η προσπέλασή του. Η μέθοδος της Distributed Cache, αναλύεται στην παράγραφο 4.2.

Προκειμένου να καταλήξουμε στον παραπάνω τρόπο προσαγούσης του προβλήματος, χρειάστηκε να μελετήσουμε αλγορίθμους οι οποίοι εκπροσωπούν κατά κάποιο τρόπο κατηγορίες και μεθόδων που πιθανά θα μπορούσαν να χρησιμοποιηθούν, πλην όμως δεν επελέγησαν για λόγους που περιγράφονται και στην ανάλυσή τους και κυρίως αφορούν στο γεγονός ότι το αποτέλεσμα που θα προέκυπτε από τη χρήση τους ήταν περισσότερο «αυστηρό» από αυτό που επιτάσσει η ανάγκη της ομοιότητας κατά προσέγγιση.

ΚΕΦΑΛΑΙΟ 2

ΤΟ Map Reduce ΚΑΙ Η ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΜΕ ΤΟ Hadoop

2.1 ΑΝΑΛΥΣΗ ΣΥΣΤΗΜΑΤΩΝ ΕΙΣΗΓΗΣΕΩΝ.

Τα συστήματα προτάσεων σε χρήστες, διαχωρίζονται σε δύο μεγάλες κατηγορίες. Τα **Content based** συστήματα και τα **Collaborative filtering** συστήματα [2]. Στα **Content based** συστήματα, η ομοιότητα των αντικειμένων, καθορίζεται ανάλογα με το περιεχόμενο. Σε κάποιες κατηγορίες παρόλα αυτά, η προτίμηση του χρήστη, δε μπορεί να καθοριστεί από το περιεχόμενο, πχ τα άρθρα, δεν είναι πάντοτε δυνατό να χαρακτηριστούν από τον τίτλο τους, ή το θέμα τους. Ο ευρύτερος σκοπός εξάλλου, είναι να δημιουργηθεί ένα σύστημα που θα βρει εφαρμογή σε ένα μεγαλύτερο εύρος αντικειμένων, όπως για παράδειγμα εικόνες, μουσική και βίντεο, όπου είναι πολύ δύσκολο να αναλυθεί το περιεχόμενό τους. Από την άλλη πλευρά, τα **Collaborative filtering** συστήματα, χρησιμοποιούν την αξιολόγηση των αντικειμένων από χρήστες και επιστρέφουν προτάσεις σχετικού ενδιαφέροντος. Στην περίπτωση του Google News, η

αξιολόγηση είναι δυαδικού τύπου, δηλαδή κάθε κλικ σε ένα άρθρο αντιστοιχεί σε «1», ενώ όταν δεν υπάρχει κλικ σε «0». Τα **Collaborative filtering** συστήματα μπορούν επιπρόσθετα να διαχωριστούν σε δύο υποκατηγορίες. Τις memory-based και τις model-based. Παρακάτω παρατίθεται σχετική εργασία σε αλγορίθμους, που έχει πραγματοποιηθεί για κάθε μία από τις δύο αυτές υποκατηγορίες ξεχωριστά.

2.2 ΤΟ Google News.

Το Google News (http://news.google.com/) είναι ένα website – συλλέκτης νέων – ειδήσεων.

Χρησιμοποιεί προσωποποιημένη οπτικοποίηση ανά χρήστη, απομνημονεύοντας αναζητήσεις του χρήστη και επιλογές που έχει κάνει (clicks) και τις χρησιμοποιεί online (πάντοτε για τον εκάστοτε χρήστη). Με βάση την ιστορικότητα στις αναζητήσεις του χρήστη, έχει δημιουργήσει το «χώρο» προτεινόμενων άρθρων. Πρόκειται ουσιαστικά για μια μηχανή αναζήτησης κριτηρίων ομοιότητας, που εφαρμόζεται σε άρθρα, προκειμένου να προκύψουν υποδείξεις, προτάσεις με άρθρα παρόμοιου ενδιαφέροντος [7].

Η χρήση του μπορεί να γίνει απολύτως εξατομικευμένη, τόσο ως προς το τοπικό μέρος, χώρα, περιοχή, που ανήκει ο χρήστης, όσο και στις αναφορές πηγών, το είδος των ειδήσεων και τις πρόσφατες αναζητήσεις.

Η τεχνική που προτιμάται και αποσκοπεί στην εκμάθηση των προτιμήσεων του χρήστη, προκειμένου να του γίνουν οι σχετικές συστάσεις που θα βασίζονται στα δεδομένα του και στα δεδομένα της κοινότητας που κατατάσσεται αυτός, είναι το **collaborative filtering.** Το Google News, όπως επίσης και το Amazon (http://www.amazon.com/) είναι από τα χαρακτηριστικότερα websites που χρησιμοποιούν την τεχνική αυτή.

Αποτύπωση στην εκτέλεση της τεχνικής του collaborative filtering αποτελεί το **Min Hashing**, τεχνική που θα αναλύσουμε και θα αξιοποιήσουμε και στη συνέχεια και από την οποία θα προκύψει τελικά και ο αλγόριθμος **MinHashingMR**, ο οποίος αποτελεί και το «προϊόν» της εργασίας αυτής.

Το πρόβλημα που θα μας απασχολήσει, είναι ότι αν έχουμε N χρήστες και M αντικείμενα (άρθρα στην προκειμένη περίπτωση), τότε για ένα συγκεκριμένο χρήστη u, που έχει καταγεγραμμένο ένα ιστορικό επιλογών άρθρων (clicks) C_u, να προταθούν K άρθρα, τα οποία μπορεί να τον ενδιαφέρουν να διαβάσει.

Για τη βέλτιστη χρήση της τεχνικής αυτής αντιμετωπίζουμε ζητήματα που αφορούν στη δυνατότητα επέκτασης μιας τέτοιας εφαρμογής, σε εκατομμύρια χρηστών που επισκέπτονται τη σελίδα, αλλά και σε εκατομμύρια αντικείμενα (άρθρα) που αναρτώνται διαρκώς, κάτι που δημιουργεί την ανάγκη πολύ συχνών δυναμικών ενημερώσεων των δεδομένων, αφού τα νέα που μπορεί να ενδιαφέρουν, αναρτώνται και φυσικά παραμένουν νέα για πολύ λίγες ώρες [9]. Αυτό σε αντίθεση με το Amazon που αναφέραμε νωρίτερα ως παράδειγμα, όπου θα μπορούσε πιο εύκολα να υπάρχει και μια πιο στατική αντιμετώπιση των δεδομένων, λόγω του αντικειμένου που πραγματεύεται το site. Ταυτόχρονα υπάρχουν και πολύ αυστηρά περιθώρια χρόνου μέσα στον οποίο θα πρέπει να υλοποιηθεί η απαίτηση του χρήστη. Τέλος, η αξιολόγηση που γίνεται λαμβάνοντας υπ' όψη τα click και όχι με κάποια διαβάθμιση, όπως γίνεται αντίστοιχα στο Amazon, μπορεί να μας δώσει ένδειξη για την επιλογή του χρήστη, όχι όμως για τη γνώμη του ως προς τη συγκεκριμένη επιλογή. Μπορεί δηλαδή απλά να έχει γίνει ένα click σε ένα άρθρο, είτε εκ παραδρομής, είτε χωρίς ουσία, είτε εσκεμμένα και μετά την ανάγνωσή του η γνώμη του χρήστη να έχει αντιστραφεί σχετικά με τη συγκεκριμένη επιλογή του.

Για την υλοποίηση του Google News, κάτι που πρακτικά αποτελεί και τη βάση για την ανάπτυξη της πλειοψηφίας των συστημάτων προτεινόμενων αντικειμένων, υπάρχουν τα περιθώρια επιλογής χρήσης δύο τεχνικών φιλτραρίσματος (filtering). Η **Content-based Filtering**, με την οποία επιλέγονται αντικείμενα όμοια προς ένα τρέχον, που έχουν αξιολογηθεί ικανά και προτείνονται και η Collaborative Filtering, με την οποία θα ασχοληθούμε και εκτενέστερα. Η τεχνική Collaborative Filtering διακρίνεται σε δύο τύπους: Την Memory Based, η οποία εκτιμά παρελθούσες αξιολογήσεις χρηστών και υπολογίζει μέσο όρο ποιότητας αξιολογήσεων και την Model Based, η οποία οργανώνει (μοντελοποιεί) τους χρήστες με βάση παρελθούσες επιλογές και αξιολογήσεις, ενώ ταυτόχρονα χρησιμοποιεί τα μοντέλα αυτά προκειμένου να πραγματοποιήσει μελλοντικές προβλέψεις που εκτιμάται ότι ενδιαφέρουν τους χρήστες. Περισσότερα για τον τύπο αυτό θα δούμε παρακάτω.

Συγκεκριμένα στο Google News, χρησιμοποιείται ένας μικτός αλγόριθμος, με το Model based τμήμα του να χρησιμοποιεί τον αλγόριθμο Min Hash. Η λογική του αλγορίθμου είναι να κρατάει

ένα σκορ από τις επιλογές που γίνονται ανά αντικείμενο και στη συνέχεια να συνδυάζει (ουσιαστικά συγκρίνει) αντικείμενα, προκειμένου να πραγματοποιήσει εισηγήσεις.

2.3 Η ΤΕΧΝΙΚΗ Collaborative Filtering.

Έχουν μελετηθεί κατά καιρούς πολλές προσεγγίσεις στην τεχνική collaborative filtering, αλλά σπάνια έχουν αναφερθεί μελέτες για πραγματικά μεγάλο αριθμό δεδομένων (αρκετά εκατομμύρια χρηστών και αντικειμένων) και δυναμικά μεταβαλλόμενο (το set δεδομένων που υπόκειται σε επεξεργασία, διαρκώς μεταβάλλεται). Παράλληλα, η αύξηση χρήσης web υπηρεσιών, δημιούργησε την ανάγκη εμφάνισης προσωποποιημένων συστάσεων σε χρήστες. Θα χρησιμοποιήσουμε για ανάλυση το παράδειγμα με τους χρήστες του Google News [1]. Η προσέγγιση με την τεχνική αυτή, πρέπει να είναι ανεξάρτητη του περιεχομένου και ανεξάρτητη από τον τομέα που θα εφαρμοστεί, κάνοντάς την εύκολο να υιοθετηθεί, τόσο από εφαρμογές, όσο και από γλώσσες, καθώς και με το ελάχιστο κόστος, τόσο στην υλοποίηση και συντήρηση, όσο και στην ανάπτυξη και την προσαρμογή - παραμετροποίηση.

Στο Internet κάθε άλλο από έλλειψη περιεχομένου υπάρχει. Η ανεύρεση του κατάλληλου περιεχομένου για τον κάθε χρήστη, δημιουργεί την πρόκληση, την ανάγκη προσπέλασης μεγάλου όγκου δεδομένων. Και αυτό αφορά όλους τους τομείς αναζήτησης. Ξεκινώντας από κάτι που κάποιος θα ήθελε να διαβάσει, κάτι που θα απαντούσε σε μια ερώτηση, ή κάτι που θα ήθελε κάποιος να ακούσει, ή να δει [16, 17, 18, 19, 20]. Οι μηχανές αναζήτησης, βοηθούν να ξεπεραστεί αυτό το πρόβλημα. Ειδικά εάν η αναζήτηση αφορά σε κάτι συγκεκριμένο, το οποίο μπορεί να εκφραστεί με μια λέξη κλειδί, ή ένα query. Σε πολλές περιπτώσεις όμως ο χρήστης δεν έχει τη δυνατότητα να εκφράσει κατάλληλα αυτό που αναζητά. Συχνά αυτό αποτελεί και το στόχο σε sites με νέα ή ειδήσεις κλπ.

Πολλοί χρήστες, αντί να καταλήγουν σε sites του τύπου news.google.com, www.netflix.com, αναζητούν κάποιο θέμα, με κλειδί αναζήτησης τη φράση «δείξε μου κάτι που να με ενδιαφέρει». Σε τέτοιες ακριβώς περιπτώσεις η παρουσίαση συστάσεων στους χρήστες, που να βασίζονται στα ενδιαφέροντά τους, όπως αυτά έχουν καταγραφεί στο παρελθόν, τόσο από τη δραστηριότητά τους, κυρίως από sites κοντινού ενδιαφέροντος που έχουν ανατρέξει, αποτελεί το αντικείμενο έρευνας.

Η τεχνολογία του Collaborative Filtering, έχει στα στόχο να μάθει τις προτιμήσεις των χρηστών και να κάνει συστάσεις που να βασίζονται στα δεδομένα των χρηστών και των διαδικτυακών κοινοτήτων. Αποτελεί συμπληρωματική τεχνολογία του λεγόμενου content-based filtering, που έχει ως χαρακτηριστικό παράδειγμα την αναζήτηση με βάση λέξεις - κλειδιά. Η πιο γνωστή ίσως χρήση του collaborative filtering, είναι στο Amazon.com, όπου το προηγούμενο ιστορικό αγορών του χρήστη, χρησιμοποιείται για να γίνουν προτάσεις ενδιαφέροντος για νέα προϊόντα.

Ο στόχος είναι να δημιουργηθεί μια μηχανή αναζήτησης, που θα μπορούσε να χρησιμοποιηθεί για να κάνει προσωποποιημένες προτάσεις ενδιαφέροντος, δραστηριοποιούμενη σε ένα πολύ μεγάλο web χώρο, όπως για παράδειγμα το Google News (http://news.google.com), τον οποίο επισκέπτονται εκατομμύρια χρήστες με εντελώς διαφορετικές προτιμήσεις. Αντίστοιχα και ο αριθμός των άρθρων που δημοσιεύονται σε αυτό, είναι της τάξης μερικών εκατομμυρίων. Το κύριο χαρακτηριστικό, είναι ότι ανανεώνονται ανά πολύ μικρά χρονικά διαστήματα και τα άρθρα που περιέχονται σε αυτό και οι εγγραφές και οι διαγραφές, είναι πάρα πολλές σε ελάχιστο χρονικό διάστημα. Έτσι, κάποιο από τα υπάρχοντα συστήματα αναζήτησης και πρότασης, δεν καλύπτει τις προϋποθέσεις, αφού υπάρχει η απαίτηση τα άρθρα που εμφανίζονται να είναι έγκυρα και «φρέσκα». Μια αντίστοιχη προσέγγιση έχουμε και στο Amazon, όπου κάνει αντιστοίχου επιπέδου και επικαιρότητας συστάσεις.

Το Google News είναι ένα site που παρουσιάζει νέα τα οποία δημιουργούνται από υπολογιστές, οι οποίοι τα συλλέγουν από πάνω από 4500 σημεία του διαδικτύου, ομαδοποιεί παρόμοιες ιστορίες – άρθρα και τα παρουσιάζει ανάλογα με τα πλήρως προσωποποιημένα ενδιαφέροντα του εκάστοτε χρήστη. Στην περίπτωση που ο χρήστης κάνει logon στο σύστημα, το Google ξεκινάει να καταγράφει το ιστορικό αναζήτησης του συγκεκριμένου χρήστη, ανακαλύπτει ιστορίες – άρθρα πιθανού ενδιαφέροντος και τα παραθέτει, ενώ ταυτόχρονα εμφανίζει και αναζητήσεις που έχουν γίνει από το χρήστη στο παρελθόν και τις διαθέτει σε αυτόν.

Κατά μία φιλοσοφία, το κλικ που κάνει ο χρήστης σε ένα άρθρο, θεωρείται ως μία ψήφος από ένα χρήστη στο συγκεκριμένο άρθρο. Αυτό ουσιαστικά, μας παραπέμπει σε sites όπως το Netflix για παράδειγμα, με δύο διαφορές. Κατά πρώτον, η θεώρηση του κλικ από το χρήστη ως πιθανής ψήφου, διαφέρει από τη βαθμολόγηση σε μια κλίμακα από 1 – 5. Κατά δεύτερον, δεν περιλαμβάνεται πουθενά η αντίθετη περίπτωση. Η αρνητική δηλαδή προδιάθεση του χρήστη για ένα άρθρο, δεν αποτυπώνεται σε κάποιο αντίστοιχο σημείο. Για να αποφευχθεί τέλος το πρόβλημα της επιλογής από το χρήστη, ενός άρθρου όπου αρχικά φαινόταν ενδιαφέρον, ενώ κατά την ανάγνωση μπορεί να απορριπτόταν και επομένως η πρόταση αυτού και παρομοίων με

αυτό άρθρων, να αποτελεί μια έγκυρη σύσταση του συστήματος, υπάρχει ένα σύστημα προεπισκόπησης, αρκετά αναλυτικό, ώστε με μια ανάγνωση σε αυτό, αποφεύγονται ανεπιθύμητες επιλογές.

2.3.1 Μοντελοποίηση του προβλήματος.

Προσπαθώντας να μοντελοποιήσουμε το πρόβλημα της αναζήτησης κατάλληλων προτάσεων στο σύστημα, υποθέτοντας ότι έχουμε N στο σύνολο χρήστες, δηλαδή, $U = \{u_1, u_2, \ldots, u_N\}$ και πάνω από M αντικείμενα, δηλαδή, $S = \{s_1, s_2, \ldots, s_M\}$, το σύνολο των κλικ που έχει κάνει ένας συγκεκριμένος χρήστης u, C_u, βασιζόμενο στα άρθρα που έχει αναζητήσει, είναι $\{s_{i1}, \ldots, s_{i|Cu|}\}$. Υποθέτουμε επίσης, ότι K είναι ο αριθμός των άρθρων που έχουν προταθεί στο χρήστη.

2.3.2 Memory – based αλγόριθμοι.

Οι **Memory – based αλγόριθμοι** κάνουν προβλέψεις αξιολογήσεων (ratings) χρηστών, βασιζόμενοι σε προηγούμενες αξιολογήσεις που οι ίδιοι οι χρήστες έχουν κάνει. Τυπικά, η πρόβλεψη υπολογίζεται ως ένας μέσος όρος αξιολογήσεων, με χρήση όμως ενός «βάρους» το οποίο είναι αναλογικό της ομοιότητας των χρηστών. Ενδεικτικά, μέτρα ομοιότητας είναι ο συντελεστής συσχέτισης Pearson και η ομοιότητα cosine [2]. Ο πίνακας ομοιότητας ζευγών χρηστών, εκτελείται στο παρασκήνιο. Στον πίνακα αυτό υπάρχουν δυαδικά - 1 ή 0 – ανάλογα με την αξιολόγηση ή όχι αντίστοιχα ενός αντικειμένου. Ταυτόχρονα με χρήση ενός κατώτατου σημείου (threshold) αποφασίζεται η ομοιότητα ή όχι δύο χρηστών.

Οι Memory – based μέθοδοι αναπτύχθηκαν σε δημοτικότητα λόγω της απλότητάς τους, αλλά και της σχετικά απλής φάσης εκπαίδευσης του συστήματος. Υπάρχει όμως μεγάλο περιθώριο βελτίωσής τους στον τομέα της επεκτασιμότητας.

2.3.3 Model-based αλγόριθμοι.

Σε αντίθεση με τους Memory – based αλγόριθμους, οι Model-based αλγόριθμοι χρησιμοποιούν τις προηγούμενες προβλέψεις των χρηστών, για να προβλέψουν τις αξιολογήσεις σε αντικείμενα που οι χρήστες δεν έχουν ακόμα δει. Η λογική αυτή, δημιούργησε αρχικά δύο μοντέλα ανάπτυξης, το Cluster μοντέλο και το Bayesian μοντέλο. Οι περισσότεροι αλγόριθμοι που έχουν αναπτυχθεί, έχουν να κάνουν με τη διαχείριση χρηστών με πολλαπλά ενδιαφέροντα και βρίσκονται ταξινομημένοι σε διαφορετικά και περισσότερα του ενός clusters. Στους αλγόριθμους αυτούς και στο επίπεδο που απαιτείται, χρησιμοποιούμε ένα τύπο ομαδοποίησης (clustering) το MinHash. Η μέθοδος **MinHashing** είναι μια μέθοδος ομαδοποίησης πιθανοτήτων, η οποία αναθέτει ένα ζεύγος χρηστών στο ίδιο cluster, με πιθανότητα ανάλογη της επικάλυψης των αντικειμένων για τα οποία οι δύο χρήστες έχουν ψηφίσει – επιλέξει – κάνει κλικ. Ο κάθε χρήστης $u \in U$, αντιπροσωπεύεται από ένα set αντικειμένων, τα οποία αποτελούν το ιστορικό αναζήτησης του χρήστη C_u. Η ομοιότητα μεταξύ δύο χρηστών u_i, u_j, καθορίζεται από την επικάλυψη (κοινά αντικείμενα) μεταξύ των set αντικειμένων, η οποία δίνεται από τον τύπο:

$$S(u_i, u_j) = |Cu_i \cap Cu_j| \,/\, |Cu_i \cup Cu_j|$$

Το παραπάνω μέτρο ομοιότητας, είναι γνωστό και ως «συντελεστής Jaccard» λαμβάνει τιμές «0» ή «1». Δεδομένου επίσης ενός χρήστη u_i, θέλουμε να υπολογίσουμε την ομοιότητά του με όλους τους υπόλοιπους χρήστες u_j, την οποία συμβολίζουμε $S(u_i, u_j)$ και να συστήσουμε στο χρήστη u_i αντικείμενα – ιστορίες για την περίπτωση του Google news – που έχουν ψηφιστεί από τους υπόλοιπους χρήστες με βαρύτητα $S(u_i, u_j)$. Ωστόσο το να γίνει αυτό σε πραγματικό χρόνο δεν είναι πάντα εφικτό. Προκύπτουν γι' αυτό τεχνικές που περιορίζουν την αναζήτηση, όπως για παράδειγμα η δημιουργία ενός hash table, για ανακαλύψουμε χρήστες που έχουν τουλάχιστον μία κοινή ψήφο – επιλογή.

Η βασική ιδέα στο MinHashing είναι να μετατίθενται στιγμιαία τα set των αντικειμένων (το S) και για κάθε χρήστη u_i να υπολογίζεται μια τιμή που θα ονομάζουμε «τιμή hash» και θα απεικονίζουμε ως $h(u_i)$. Δημιουργείται ένας πίνακας με αυτές τις hash τιμές και συγκρίνονται τα αντικείμενα. Αποδεικνύεται ότι η πιθανότητα δύο χρήστες να έχουν τις ίδιες hash function, είναι ουσιαστικά ίδια με το συντελεστή Jaccard. Επομένως μπορούμε να σκεφτούμε το MinHashing ως ένα αλγόριθμο που μπορεί να καταδείξει την πιθανότητα ομοιότητας, κάνοντάς το με το να αντιστοιχεί σε μια θέση – ένα cluster, μια hash τιμή και να κατατάσσει εκεί όλα τα ζεύγη χρηστών με πιθανότητα ομοιότητας $S(u_i, u_j)$. Όμοια μπορούν να συνενωθούν p hash κλειδιά για χρήστες,

με $p \geq 1$, ώστε η πιθανότητα οποιοιδήποτε δύο χρήστες να συμφωνούν να γίνει $S(u_i, u_j)^p$. Με άλλα λόγια, συνενώνοντας hash functions, μπορούμε να πετύχουμε ένα φιλτράρισμα στα υπό έλεγχο clusters, να τα «εκκαθαρίσουμε» με αποτέλεσμα να έχουμε περιπτώσεις ομοιότητας πιο ξεκάθαρες.

2.4 Η ΜΕΘΟΔΟΣ Map Reduce.

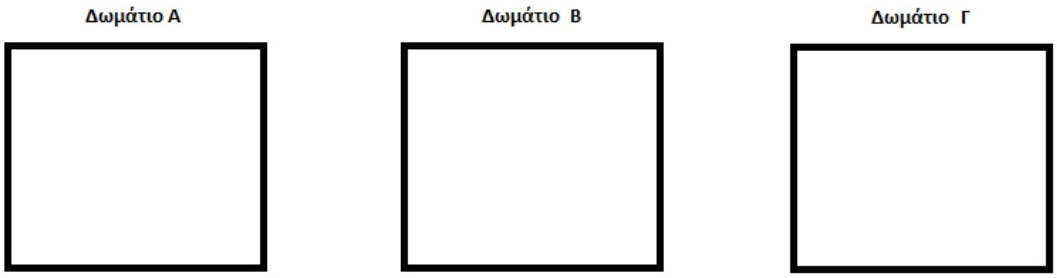
Το Map Reduce εφευρέθηκε από τους μηχανικούς της Google, ως ένα σύστημα δημιουργίας μηχανών αναζήτησης που θα έβγαιναν στην παραγωγή ενημέρωσης για την εφαρμογή του Google News. Έκτοτε, έχει χρησιμοποιηθεί για πολλές άλλες εφαρμογές. Το φάσμα των αλγορίθμων που μπορεί να εκφραστεί σε Map Reduce [2, 13, 15], ποικίλει από εφαρμογές ανάλυσης εικόνας, έως και αλγορίθμους μηχανικής μάθησης. Σίγουρα δε μπορεί να επιλύσει κάθε πρόβλημα, αλλά σίγουρα επίσης, είναι ένα σπουδαίο εργαλείο για γενική επεξεργασία δεδομένων.

Χρησιμοποιούμε γενικά τη μέθοδο αυτή, για να διαχειριστούμε υπολογισμούς μεγάλης κλίμακας, με τέτοιο τρόπο, που να είναι ανεκτός από το hardware σύστημα που χρησιμοποιείται. Αυτό που ουσιαστικά χρειάζεται να γίνει, είναι να δημιουργηθούν δύο functions, μία για τη Map διαδικασία και μία για τη Reduce αντίστοιχα [16, 17, 18, 19, 20]. Ταυτόχρονα, το σύστημα, επιχειρεί την παράλληλη εκτέλεση και συντονισμό των εργασιών που εκτελούνται, ενώ συγχρόνως καλύπτεται και η πιθανότητα κάποια από τις εργασίες να μην διεκπεραιωθεί λόγω αστοχίας υλικού κόστος.

Για την καλύτερη κατανόηση του συστήματος Map Reduce, που εκθέτουμε παρακάτω, θα αποτυπώσουμε ένα παράδειγμα που ουσιαστικά θα αποτελεί και το case που έχουμε να μελετήσουμε και με βάση αυτό θα περιγράψουμε τι ακριβώς συμβαίνει στη διαδικασία, από την αρχή έως το τέλος της.

Έστω λοιπόν ένα σύνολο από μπάλες τεσσάρων διαφορετικών χρωμάτων, πράσινες, κόκκινες, κίτρινες και μπλε. Οι μπάλες αυτές βρίσκονται σε τρία διαφορετικά δωμάτια, όλες σε ένα κουτί, σε κάθε δωμάτιο και αυτό που θέλουμε να κάνουμε, είναι να τις μετρήσουμε ανά χρώμα.

Σχηματικά το παραπάνω εικονίζεται στο παρακάτω σχήμα 2.1. Σημειώνουμε ότι στην αρχική φάση που βρισκόμαστε, δε γνωρίζουμε την αναλογία χρωμάτων, ούτε πόσες μπάλες έχει το κάθε δωμάτιο.



Σχήμα 2.1. Τα δωμάτια, χωρίς να γνωρίζουμε πόσες μπάλες, ή τι χρώματα περιέχονται στο κάθε ένα από αυτά.

Με τη μέθοδο Map Reduce, όλος ο υπολογισμός διαιρείται σε πολλούς μικρούς υπολογισμούς δύο τύπων, έναν τύπου Map και έναν τύπου Reduce.

Η παραδοχή που υιοθέτησε η μέθοδος του Map Reduce, είναι ότι ολόκληρο το σύνολο δεδομένων - ή τουλάχιστον ένα μεγάλο μέρος από αυτό, υποβάλλεται σε επεξεργασία για κάθε αναζήτηση (query). Ουσιαστικά όμως αυτή είναι και η δύναμή του. Έχει την ικανότητα να τρέξει ένα ad hoc query σε ολόκληρο το σύνολο των δεδομένων και να δώσει τα αποτελέσματα σε εύλογο χρονικό διάστημα με το μετασχηματισμό που επιζητείται. Αλλάζει γενικά τον τρόπο με τον οποίο σκεφτόμαστε για τα δεδομένα. Ερωτήσεις που απαιτούταν πολύς χρόνος για να απαντηθούν πριν, μπορεί τώρα να απαντηθούν πολύ γρήγορα, κάτι που με τη σειρά του οδηγεί σε νέα ερωτήματα και νέες ιδέες.

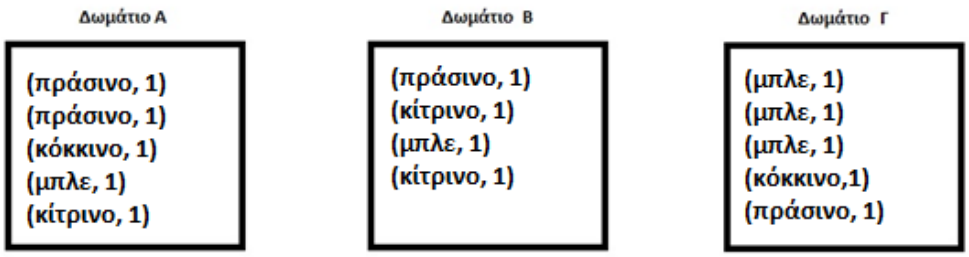
Το Map Reduce χρησιμοποιεί πρόσβαση στα δεδομένα που βρίσκονται αποθηκευμένα στον εκάστοτε τοπικό υπολογιστή, έτσι ώστε η πρόσβαση στα δεδομένα να είναι γρήγορη. Αυτό το χαρακτηριστικό, που ονομάζεται **data locality**, βρίσκεται στο επίκεντρο της φιλοσοφίας του Map Reduce και είναι και ένας από τους κύριους λόγους της καλής απόδοσής του. Κάθε ένα από τα τρία δωμάτια του παραπάνω παραδείγματός μας, αποτελεί και ένα διαφορετικό ηλεκτρονικό υπολογιστή. Αντίστοιχα, αναγνωρίζοντας ότι το εύρος ζώνης ενός δικτύου είναι ο πιο πολύτιμος πόρος σε ένα περιβάλλον με επίκεντρο τα δεδομένα, οι Map Reduce εφαρμογές μοντελοποιούνται με άξονα την τοπολογία του δικτύου.

Το Map Reduce μπορεί να εμφανίζεται έχοντας τα χαρακτηριστικά ενός πολύ περιοριστικού μοντέλου προγραμματισμού και κατά μία έννοια είναι, αφού αφενός περιοριζόμαστε σε εκφράσεις όπως θα δούμε παρακάτω, ζευγών της μορφής key – value, ενώ αντίστοιχα και οι Mappers και οι Reducers, απαιτούν συγκεκριμένο συντονισμό για τη λειτουργία τους.

2.5 Η ΥΛΟΠΟΙΗΣΗ ΤΟΥ Map Reduce.

Μια διαδικασία Map Reduce, υλοποιείται ως εξής [2]:

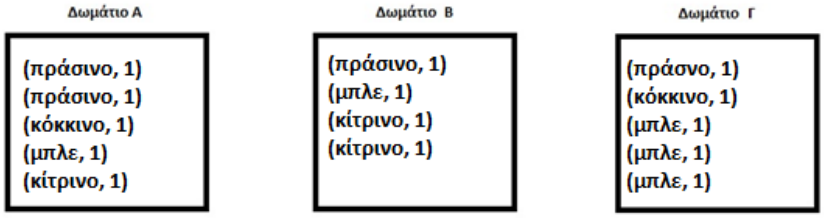
1. Σε κάθε μία από ένα σύνολο Map εργασιών, δίνεται ένα υπολογιστικό κομμάτι για να διεκπεραιωθεί. Η Map εργασία δημιουργεί ζεύγη κλειδιού - τιμής (key - value pairs). Το τι θα περιέχεται ουσιαστικά σε αυτά τα ζεύγη, καθορίζεται από αυτό που θα επιλέξει ο προγραμματιστής της διαδικασίας, με βάση αυτό που θα οδηγήσει με περισσότερη ακρίβεια στο να προσδιοριστεί το επιθυμητό αποτέλεσμα. Έστω τώρα, ότι για το παράδειγμά μας, επιλέγεται από τον προγραμματιστή να δοθεί ως κλειδί (key) το χρώμα κάθε μπάλας, δηλαδή ένα από τα χρώματα πράσινο, κόκκινο, κίτρινο και μπλε και ως τιμή (value), ο αριθμός 1. Τώρα το κάθε δωμάτιο του παραδείγματός μας, έχει μια πιο ξεκάθαρη εικόνα, που αποτυπώνεται στο σχήμα 2.2 παρακάτω.



Σχήμα 2.2. Οι μπάλες τακτοποιημένες όπως ανακαλύπτονται σε ζεύγη (key, value).

2. Τα ζεύγη που παράγονται, συλλέγονται από ένα Master Controller και ταξινομούνται κατά κλειδί. Τα κλειδιά κατανέμονται ανά κατηγορία σε όλες τις Reduce εργασίες που έχουμε διαθέσιμες (Reducers), έτσι ώστε όλα τα ζεύγη με το ίδιο κλειδί να βρίσκονται στον ίδιο Reducer μετά το πέρας της διαδικασίας. Σε κάθε δωμάτιο του παραδείγματός μας, ταξινομούνται τα

ζεύγη, ανά κλειδί – χρώμα στην περίπτωσή μας και αντίστοιχα, το παράδειγμά μας γίνεται όπως εικονίζεται στο σχήμα 2.3.



Σχήμα 2.3. Τα ευρήματα – οι μπάλες - ταξινομημένα κατά κλειδί σε κάθε ένα από τα δωμάτια που βρέθηκαν.

3. Οι εργασίες Reduce, εργάζονται με ένα κλειδί κάθε φορά και συνδυάζουν όλες τις τιμές που συνοδεύουν το εκάστοτε κλειδί, ανάλογα με το τι επιθυμεί ο προγραμματιστής να λάβει στην έξοδο, από τη διαδικασία Reduce. Καταλήγοντας και πάλι στο παράδειγμα, θέλουμε στην έξοδο να λάβουμε πόσες μπάλες από κάθε χρώμα υπάρχουν συνολικά και στα τρία δωμάτια. Στη διαδικασία Reduce, αρκεί να μετρήσουμε πόσες φορές εμφανίζεται κάθε ένα από τα κλειδιά, που είναι τα χρώματα, σε όλα τα δωμάτια. Έτσι θα έχουμε:

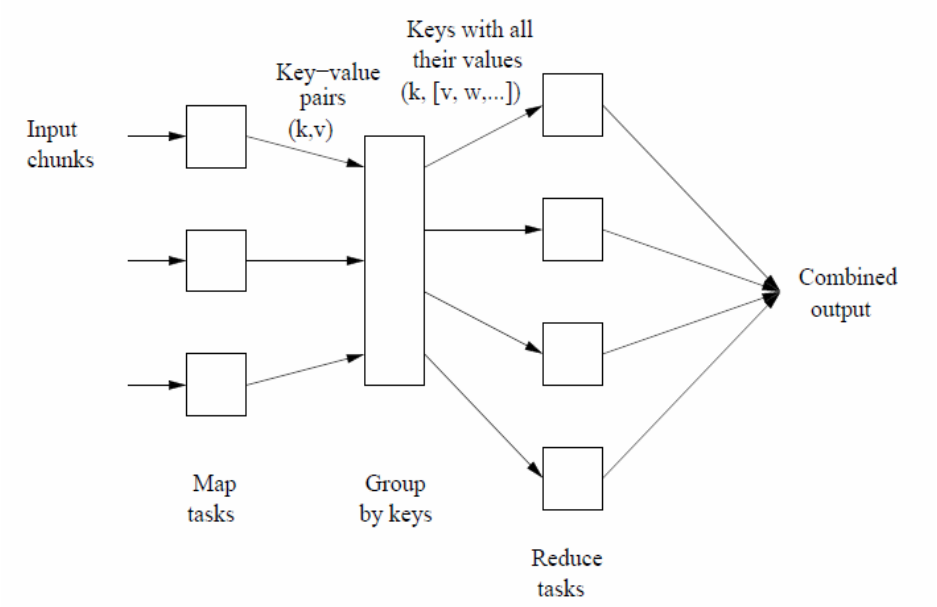
(πράσινο, 1) 4 φορές.

(κόκκινο, 1) 2 φορές.

(κίτρινο, 1) 3 φορές και

(μπλε, 1) 5 φορές.

Στο παρακάτω Σχήμα 2.4, απεικονίζεται ο υπολογισμός Map Reduce.



Σχήμα 2.4. Ο υπολογισμός Map Reduce.

Όπως φαίνεται και στο σχήμα 2.4, το Map Reduce framework, παρέχει τα κατάλληλα Interfaces, ώστε ο χρήστης, χρησιμοποιώντας τα, να πρέπει μόνο να παρέχει στο σύστημα μια περιγραφή του επιθυμητού υπολογισμού, που θα έχει επικεντρωθεί στη λειτουργικότητα του συστήματος και όχι στην παράλληλη λειτουργία της αναζήτησης, κάτι που αναλαμβάνει να διεκπεραιώσει το Map Reduce framework [6].

2.5.1 Οι εργασίες Map.

Ας δούμε όμως πιο αναλυτικά, ποιες είναι οι εργασίες που διεκπεραιώνονται κατά τη Map διαδικασία. Θεωρούμε τα δεδομένα εισόδου σε μια Map διαδικασία, ως ένα σετ από στοιχεία, που μπορεί να είναι οποιουδήποτε είδους (μπάλες για παράδειγμα). Τα στοιχεία βρίσκονται αποθηκευμένα σε ένα συγκεκριμένο κομμάτι του hardware (στα δωμάτια), αλλά κανένα στοιχείο δεν έχει αποθηκευτεί σε δύο διαφορετικά hardware κομμάτια. Τεχνικά, τόσο τα στοιχεία εισόδου για τις Map εργασίες, όσο και τα στοιχεία εξόδου από αυτές, τα οποία γίνονται είσοδος για τις Reduce εργασίες, έχουν τη μορφή ζεύγους Κλειδιού – Τιμής (key - value ζεύγη του τύπου (χρώμα, 1)).

Ουσιαστικά μια Map διαδικασία γράφεται για να μετατρέψει τα δεδομένα εισόδου, σε ζεύγη Κλειδιού – Τιμής. Οι τύποι τόσο των κλειδιών (keys) όσο και των τιμών (values), είναι

αυθαίρετοι. Επίσης, ο τύπος του κλειδιού, δε χρειάζεται να είναι μοναδικός. Σε μια Map διαδικασία μπορεί να παραχθεί πολλές φορές ένα ζεύγος key - value με το ίδιο κλειδί ακόμα και για το ίδιο στοιχείο.

Στη συνέχεια ακολουθεί η διαδικασία ομαδοποίησης. Η διεργασία που εκπληρώνει ο Master Controller, γνωρίζει εκ των προτέρων πόσες εργασίες Reduce απαιτούνται να γίνουν – έστω ένα πλήθος r - και τον αριθμό αυτών των r εργασιών, τον προσδιορίζει συνήθως ο χρήστης. Στη συνέχεια ο Master Controller επιλέγει μια hash function, την οποία εφαρμόζει στα κλειδιά, η οποία παράγει ένα αριθμό από 0 έως r-1. Έστω και πάλι στο παράδειγμά μας ότι αντιστοιχίζει το 0 στο ζεύγος (πράσινο, 1), το 1 στο (κόκκινο, 1), το 2 στο (κίτρινο, 1) και το 3 στο (μπλε, 1). Κάθε κλειδί το οποίο έρχεται στην έξοδο μιας διαδικασίας Map, γίνεται hashed (κατακερματίζεται) και το ζεύγος που προκύπτει από αυτό, κατατάσσεται σε ένα από ένα σύνολο r (4 συγκεκριμένα για το παράδειγμα) τοπικών αρχείων και το κάθε ένα από αυτά προορίζεται για εισαγωγή σε μια διαδικασία Reduce. Υπάρχει φυσικά και η ευελιξία της επιλογής από το χρήστη κάποιας hash function, ή κάποιας άλλης μεθόδου ανάθεσης κλειδιών σε διαδικασίες Reduce. Σε κάθε περίπτωση όμως, ανεξάρτητα από το ποιος αλγόριθμος χρησιμοποιείται, κάθε κλειδί που προκύπτει από μια Map διαδικασία, ανατίθεται σε μία και μόνη διαδικασία Reduce.

Αφού ολοκληρωθούν επιτυχώς όλες οι Map διαδικασίες που προορίζονται για μια συγκεκριμένη διαδικασία Reduce, ο Master Controller συνενώνει το αρχείο που προκύπτει και τροφοδοτεί στη συνέχεια με αυτό το αρχείο, ως ακολουθία ζευγών key – value, τη συγκεκριμένη διαδικασία. Αυτό πρακτικά σημαίνει ότι για κάθε κλειδί k, η είσοδος στη διαδικασία Reduce, είναι ζεύγη της μορφής $(k, (u_1, u_2, \dots, u_n))$, ή διαφορετικά $(k, u_1), (k, u_2), \dots, (k, u_n)$, με το k να εξάγεται ως κλειδί από όλες τις διαδικασίες Map. Όπως και παραπάνω είδαμε την ομαδοποίηση που έγινε στα ζεύγη για τις μπάλες.

2.5.2 Οι εργασίες Reduce.

Η διαδικασία Reduce είναι γραμμένη για να βασίζεται στο να λαμβάνει ζεύγη που αποτελούνται από ένα κλειδί (key), συνοδευόμενο από μία λίστα από συνδεδεμένες τιμές (values) και να τα συνδυάζει με τον επιθυμητό – καθορισμένο από το χρήστη - τρόπο. Η έξοδος μιας διαδικασίας Reduce είναι μια ακολουθία ζευγών κλειδιού – τιμής (key – value), που αποτελούνται από το εκάστοτε κλειδί εισόδου που λαμβάνει η διαδικασία, σε ζευγάρι, με την τιμή που έχει προκύψει

για το συγκεκριμένο κλειδί. Όλες οι έξοδοι από κάθε Reduce διαδικασία, εμφανίζονται συνενωμένες σε ένα και μοναδικό αρχείο.

Οι εργασίες που πραγματοποιούνται κατά τη διαδικασία Reduce, είναι μια ακολουθία από (w, m) ζεύγη, όπου w είναι μια λέξη που εμφανίζεται τουλάχιστον μία φορά ανάμεσα στα αρχεία εισόδου και m είναι ο συνολικός αριθμός των εμφανίσεων του w ανάμεσα σε όλα τα παραπάνω αρχεία.

Η έξοδος από το Reduce, στο παράδειγμα με τις μπάλες, θα είναι του τύπου:

(πράσινο, 4)

(κόκκινο, 2)

(κίτρινο, 3)

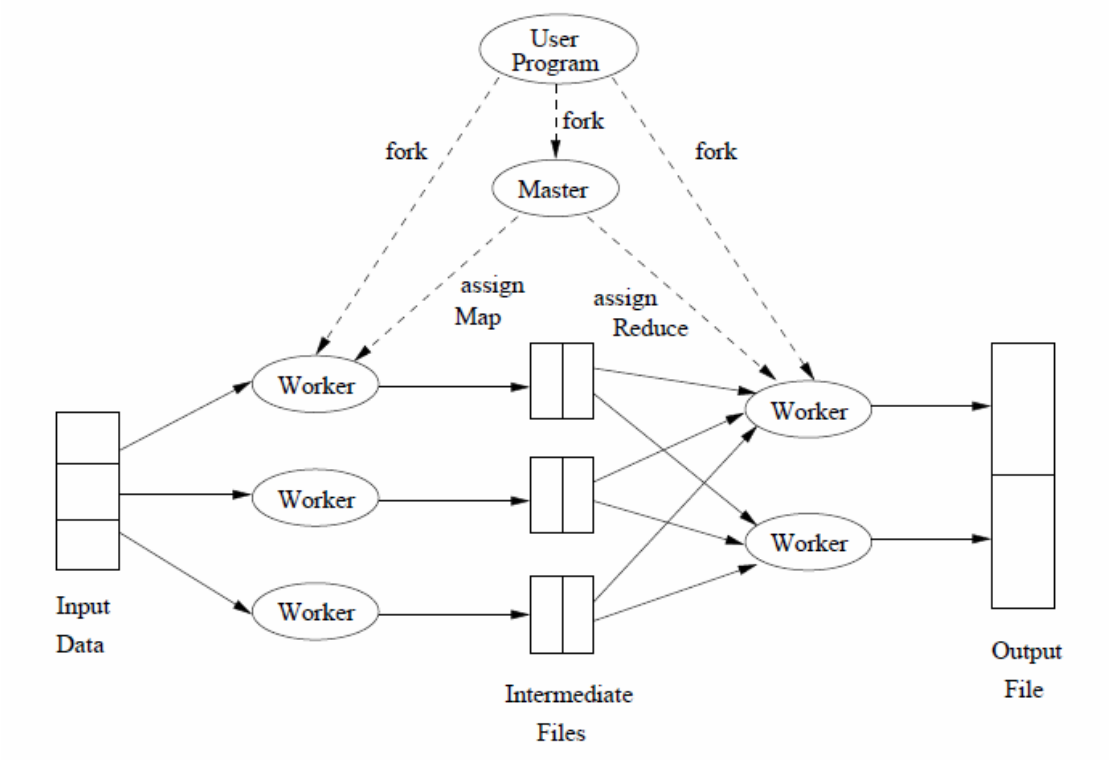
(μπλε, 5)

Η διαδικασία Reduce είναι συνειρμική και αντιμεταθετική. Αυτό σημαίνει ότι οι τιμές μπορούν να συνδυαστούν με οποιαδήποτε σειρά και το αποτέλεσμα να είναι το ίδιο.

Οι εργασίες που πραγματοποιούνται κατά τη διαδικασία Reduce, ποικίλουν, ανάλογα με το αποτέλεσμα που ζητάμε να λάβουμε στην έξοδο. Έτσι μπορούμε να έχουμε απλές ή πιο σύνθετες συναρτήσεις υπολογισμών για εξαγωγή των στοιχείων που ζητάμε να έχουμε, προϋποθέτοντας ότι με την προηγούμενη Map διαδικασία, έχουν επιλεγεί τα κατάλληλα στοιχεία, ο συνδυασμός των οποίων θα μας αποφέρει το επιθυμητό αποτέλεσμα.

2.5.3 Λεπτομέρειες κατά την εκτέλεση του Map Reduce.

Ας δούμε με λίγο περισσότερη λεπτομέρεια πώς εκτελείται ένα πρόγραμμα που χρησιμοποιεί Map Reduce. Στο σχήμα 2.5, δίνεται μια περιγραφή του πώς οι διαδικασίες, οι εργασίες και τα αρχεία αλληλεπιδρούν.



Σχήμα 2.5. Επισκόπηση της εκτέλεσης ενός Map Reduce προγράμματος.

Με τη χρήση βιβλιοθηκών που παρέχονται από το Map Reduce σύστημα, π.χ. το Hadoop, που θα αναλύσουμε στη συνέχεια, το πρόγραμμα «τσιμπάει» (forks) μια διαδικασία Master Controller και ένα αριθμό «διαδικασιών Εργάτη» (Worker processes) από διαφορετικούς υπολογιστικούς κόμβους (nodes). Συνήθως ο ίδιος Worker χειρίζεται ή εργασίες Map (Map Worker), είτε εργασίες Reduce (Reduce Worker), αλλά ποτέ και τις δύο ταυτόχρονα.

Μία από τις αρμοδιότητες που έχουν ανατεθεί στο Master, είναι να δημιουργήσει ένα αριθμό από Map εργασίες και ένα αριθμό από Reduce διαδικασίες. Οι αριθμοί αυτοί επιλέγονται από την κατασκευή του προγράμματος. Οι εργασίες αυτές ανατίθενται σε Workers από το Master. Είναι λογικό να δημιουργείται μια Map εργασία από κάθε κομμάτι των αρχείων εισόδου, αλλά η επιδίωξη πρέπει να είναι να δημιουργηθούν λιγότερες Reduce εργασίες. Ο λόγος που πρέπει οι Reduce εργασίες να είναι περιορισμένες, είναι επειδή είναι απαραίτητο η κάθε Map εργασία να δημιουργεί και ένα ενδιάμεσο αρχείο για κάθε Reduce εργασία και επομένως έχοντας πολλά ενδιάμεσα αρχεία είναι πιθανό το σύστημά μας να καταρρεύσει.

Ο Master είναι διαρκώς ενήμερος για την κατάσταση σε κάθε εργασία Map και Reduce που πραγματοποιείται ή έχει πραγματοποιηθεί, η οποία μπορεί να βρίσκεται σε μία από τις καταστάσεις «idle», «executing» (σε συγκεκριμένο Worker), ή «completed». Όταν μια εργασία σε

ένα Worker ολοκληρωθεί, τότε αναφέρεται στο Master, ο οποίος στη συνέχεια αναθέτει στο συγκεκριμένο Worker μια νέα εργασία.

Σε κάθε Map εργασία ανατίθεται ένα ή περισσότερα κομμάτια από το αρχείο, ή τα αρχεία εισόδου και εκτελεί σε αυτό το πρόγραμμα που έχει γραφτεί από το χρήστη. Η εργασία Map δημιουργεί ένα αρχείο για κάθε διαδικασία Reduce στον τοπικό δίσκο του Worker που έχει αναλάβει τη συγκεκριμένη εργασία. Ο Master παραμένει διαρκώς ενημερωμένος τόσο για τη θέση και το μέγεθος των εν λόγω αρχείων, όσο και για τη Reduce διαδικασία στην οποία πρόκειται να ανατεθούν. Όταν μια εργασία Reduce ανατεθεί από το Master σε μια διαδικασία Worker, σε αυτή την εργασία δίνονται όλα τα αρχεία που τροποποιούν την είσοδό της. Στο τέλος, η Reduce διαδικασία εκτελεί τον κώδικα που έχει γραφτεί από το χρήστη και εγγράφει την έξοδό της σε ένα αρχείο το οποίο είναι μέρος του συστήματος.

2.5.4 Οι ροές εργασίας του Map Reduce.

Πριν προχωρήσουμε σε τεχνικές και μηχανικά μέρη του πώς θα γράψουμε πρόγραμμα στο Map Reduce, θα πρέπει να μελετήσουμε τον τρόπο, με τον οποίο ένα πρόβλημα επεξεργασίας δεδομένων μετατρέπεται σε ένα μοντέλο Map Reduce.

Ανάλογα με το πόσο πολύπλοκη είναι η επεξεργασία που θέλουμε να κάνουμε στα δεδομένα, προτιμούμε γενικά να εκτελέσουμε περισσότερες, απλές, Map Reduce εργασίες, παρά μία πολύπλοκη. Σαν κανόνα λοιπόν έχουμε ότι προτιμούμε να χρησιμοποιούμε πολλές απλές εργασίες, παρά να προσθέτουμε πολυπλοκότητα στις εργασίες.

Ας δούμε ένα πιο σύνθετο παράδειγμα, το οποίο θα αναλύσουμε σε Map Reduce πρόβλημα. Ας πούμε ότι θέλουμε να εξετάσουμε ένα σύνολο από μετεωρολογικούς σταθμούς και να βρούμε τη μέση μέγιστη καταγεγραμμένη θερμοκρασία για κάθε μέρα του χρόνου και κάθε ένα από αυτούς τους μετεωρολογικούς σταθμούς. Ας πούμε ότι ζητάμε τη μέση μέγιστη θερμοκρασία που συνέβη στη 1 Ιανουαρίου, στο μετεωρολογικό σταθμό 029070-99999. Θα πρέπει τότε να αναζητήσουμε τη μέση μέγιστη θερμοκρασία που κατεγράφη στο σταθμό αυτό στη 1 Ιανουαρίου 1901, τη 1 Ιανουαρίου 1902 κλπ, μέχρι και τη 1 Ιανουαρίου 2000. Ο υπολογισμός αυτός μπορεί να διαιρεθεί σε δύο στάδια:

1. Υπολογισμός της μέγιστης ημερήσιας θερμοκρασίας για κάθε ζεύγος (σταθμού – ημερομηνίας). Εδώ υπάρχει ένα συνδυασμένο ζεύγος σταθμός – ημερομηνία.

2. Υπολογισμός της μέσης ημερήσιας θερμοκρασίας για κάθε κλειδί που θα είναι σταθμός – μέρα – μήνας. Ο Mapper λαμβάνει την έξοδο από την προηγούμενη εργασία (ζεύγος του κλειδιού σταθμού – ημερομηνίας με value τη μέγιστη θερμοκρασία) και τη μετατρέπει σε ζεύγη τύπου ((σταθμός – ημέρα – μήνας), μέγιστη θερμοκρασία). Η Reduce διαδικασία στη συνέχεια, παίρνει τη μέση από τις μέγιστες θερμοκρασίες, για κάθε κλειδί (σταθμός - ημέρα – μήνας).

Μετά την πρώτη διαδικασία, η έξοδος για το σταθμό που ενδιαφερόμαστε, θα είναι:

029070-99999 19010101 0

029070-99999 19020101 -94

...

Τα δύο πρώτα πεδία είναι το κλειδί και το τρίτο η τιμή. Μετά το δεύτερο στάδιο το αποτέλεσμα είναι:

029070-99999 0101 -68

το οποίο μεταφράζεται ως: η μέση μέγιστη θερμοκρασία στο σταθμό 029070-99999 που μετρήθηκε στη 1 Ιανουαρίου είναι -6,8^{o}C. Είναι δυνατό όλος παραπάνω υπολογισμός να γίνει με μία και μόνη Map Reduce διαδικασία, αλλά θα χρειαστεί περισσότερη δουλειά από την πλευρά του προγραμματιστή. Γενικά, το επιχείρημα να έχουμε περισσότερα, αλλά απλούστερα, Map Reduce στάδια, στέκει, αφού με τον τρόπο αυτό έχουμε Mappers και Reducers, πολύ πιο εύκολα να γίνουν μέρος μια σύνθεσης και πολύ πιο εύκολο να τους διαχειριστεί ένα σύστημα.

2.6 ΚΟΣΤΟΣ ΑΛΓΟΡΙΘΜΩΝ Map Reduce.

Για κάθε αλγόριθμο που σχεδιάζεται για Map Reduce, υπολογίζουμε τα ακόλουθα στοιχεία για να υπολογίσουμε το κόστος εκτέλεσής του [5]:

- Το κόστος προεπεξεργασίας για όλες τις εγγραφές εισόδου, έστω M.
- Το συνολικό κόστος επικοινωνίας για τη μεταγωγή όλων των δεδομένων από τους Mappers στους Reducers, έστω C.
- Το συνολικό κόστος υπολογισμών σε όλους τους Reducers, έστω R.

Το παραπάνω κόστος εκφράζεται όσον αφορά τις ακόλουθες παραμέτρους:

- Το dataset που έχουμε στην είσοδο, S και το μέγεθός του |S|.
- Το όριο απόκλισης (threshold) που καθορίζεται από τη similarity function που χρησιμοποιούμε και το οποίο αποτελεί τη μέγιστη απόσταση απόκλισης, d.
- Τις ιδιότητες των στοιχείων εισόδου, όπως για παράδειγμα το μήκος των strings.
- Τον αριθμό των Reducers K.

Με εξαίρεση τον Naive αλγόριθμο (από μια σειρά αλγορίθμων που θα εξετάσουμε στη συνέχεια), όταν αναφερόμαστε στον αριθμό K των Reducers που χρησιμοποιούνται σε μια διαδικασία Map Reduce, εννοούμε στην πραγματικότητα τον αριθμό των κλειδιών και τις τιμές που σχετίζονται με κάθε ένα από αυτά. Ο αριθμός δηλαδή των διαφορετικών κλειδιών, δίνει και το μέγιστο δυνατό αριθμό των Reduce διαδικασιών που θα χρειαστούν. Τις πιο πολλές φορές, οι διαδικασίες Reduce που απαιτούνται είναι λιγότερες από αυτό το μέγιστο αριθμό K. Επίσης, ο συνολικός χρόνος που απαιτείται για να εκτελέσουν οι Reducers, δεν εξαρτάται από το πόσα διαφορετικά κλειδιά δέχεται μία Reduce διαδικασία.

Αντίστοιχα, ο αριθμός των Mappers δε λαμβάνεται ποτέ υπ' όψη στον υπολογισμό του κόστους. Υποθέτουμε πάντα, ότι ο εκάστοτε αλγόριθμος χρησιμοποιεί όσους Mappers απαιτούνται για να γίνει η απαραίτητη διαχείριση της εισόδου. Από τη στιγμή που οι Mappers τυπικά χειρίζονται ένα στοιχείο εισόδου κάθε φορά, το συνολικό κόστος για τη Map διαδικασία, δεν επηρεάζεται

ιδιαίτερα από τον αριθμό των Mappers. Παρόλα αυτά, εάν χρησιμοποιήσουμε πολύ λίγους, ο χρόνος ολοκλήρωσης της δουλειάς θα είναι υπερβολικά υψηλός.

Ορίζουμε ως **D** το domain όλων των υπό σύγκριση εγγραφών. Το εν λόγω domain, μπορεί να είναι αναπτυγμένο σε πολλά σημεία (υπολογιστές - clusters).

Ορίζουμε τη similarity function **Sim: D X D → R**

Ορίζουμε ένα κατηγόρημα **F = (Sim, τ)**, το οποίο καθορίζεται από τη similarity function και ένα κατώτατο όριο (threshold) **τ**, που ουσιαστικά πρόκειται για την απόσταση απόκλισης (distance measure).

Το αποτέλεσμα της εφαρμογής του F σε ένα set εγγραφών (στοιχείων) **S**, για το οποίο ισχύει ότι $\boldsymbol{S \subseteq D}$, θα είναι **F(R)={(x,y) | x,y ∈ R, Sim(x,y) ≥ τ}**. Όταν (x,y) ∈ F(R), θα λέμε ότι **F(x,y)=1**.

Η ομοιότητα δηλώνεται με τη χρήσης μιας απόστασης (distance measure), η οποία χρησιμοποιείται για να καθορίσει, ή όχι, την ομοιότητα μεταξύ των στοιχείων του set.

Η τελική μας επιθυμία, θα είναι να παραλάβουμε στην έξοδο του συστήματος Map Reduce, που θα χρησιμοποιήσουμε, όλα τα (x,y) ∈ F(R).

Ορίζουμε την έννοια του (M, C, R)-map-reducible αλγορίθμου, με την οποία μπορούμε να συγκρίνουμε αλγορίθμους για το Map Reduce, χρησιμοποιώντας συγκεκριμένα χαρακτηριστικά. Δεδομένου δηλαδή ενός dataset S, ένα κατηγόρημα F θεωρείται (M, C, R)-map-reducible, όταν υπάρχει κάποιο G από το D, σε ένα domain D' = {1, …, K}, όπου K είναι ο αριθμός των Reducers που θα χρειαστούν και ακολουθεί τα παρακάτω:

- Εάν F(x, y) = 1, τότε G(x)∩G(y) ≠ 0. Κάθε ζεύγος δηλαδή όμοιων εγγραφών τοποθετούνται σε ένα τουλάχιστο ίδιο Reducer.

- Ο υπολογισμός του $G(r_i)$, για όλα τα $r_i \in S$ χρειάζεται χρόνο O(M). Αυτό αντιπροσωπεύσει το συνολικό κόστος προεπεξεργασίας για τους Mappers.

- Το συνολικό κόστος επικοινωνίας είναι $\Sigma_{r_i \in S} |G(r_i)| \leq C$. Αυτό δηλαδή είναι και το συνολικό μέγεθος των δεδομένων που μεταφέρονται στους Reducers.

- Η εύρεση όλων των ζευγών (x, y) σε κάθε Reducer i όπου $i \in G(x)$, $i \in G(y)$ και $F(x,y)=1$, απαιτεί χρόνο O(M). Αυτός είναι και ο συνολικός χρόνος επεξεργασίας στους Reducers.

2.7 ΑΠΟΣΤΑΣΕΙΣ (Distances).

Κατά την ανίχνευση ομοιότητας σε χρήστες προκειμένου να τους γίνουν υποδείξεις – προτάσεις για αντικείμενα που μπορεί να τους ενδιαφέρουν, δεν είναι απαραίτητο να έχουμε απόλυτη ταύτιση. Ενδεχομένως και να προτιμάται τελικά οι δύο χρήστες να ομοιάζουν σε ορισμένα από το σύνολο αντικείμενα και όχι σε όλα. Κατά τη σύγκριση των δύο χρηστών, ορίζεται ένας «βαθμός ομοιότητας» (στην ορολογία το συναντάμε ως threshold). Ένα ποσοστό δηλαδή, πάνω από το οποίο μπορούμε να θεωρήσουμε ότι το υπό σύγκριση ζεύγος χρηστών πληροί τις προδιαγραφές ομοιότητας.

Δεδομένου δε, ότι ο όγκος των δεδομένων που θα έχουμε σε πραγματική χρήση θα είναι απείρως μεγαλύτερος από ότι ένα πεπερασμένο πλήθος, θεωρείται ότι η απόλυτη ομοιότητα στις προτιμήσεις χρηστών είναι περίπτωση πολύ σπάνια. Ταυτόχρονα επιδίωξη μας θα πρέπει να είναι η προσέγγιση σε ομοιότητα συμπεριφοράς και όχι η απόλυτη ταύτιση χρηστών. Για όλους τους παραπάνω λόγους, χρησιμοποιούμε την έννοια της απόστασης (distance).

Πρόκειται ουσιαστικά για τον καθορισμό και τη χρήση μιας απόκλισης που ο προγραμματιστής θα μπορεί να ανεχτεί, στα αποτελέσματα σύγκρισης των χρηστών, έτσι ώστε να μην αναζητείται η απόλυτη ομοιότητα, αλλά μια παρόμοια συμπεριφορά, σε μέτρο τέτοιο όμως που η πιθανότητα ύπαρξης ομοιότητας να είναι άμεσα εξαρτώμενη και φυσικά να καθορίζεται από το πόσο πολύ ή όχι επιθυμούμε να ομοιάζουν τα υπό σύγκριση ζεύγη.

Με βάση τις ανάγκες που παρουσιάζονται σε ένα σύστημα, μπορούν να χρησιμοποιηθούν διάφοροι τύποι αποστάσεων. Εδώ θα παρουσιάσουμε τους εξής δύο: την απόσταση Hamming (**Hamming Distance**) και την απόσταση Jaccard (**Jaccard Distance**) [2, 5].

Για να κατανοήσουμε καλύτερα το πώς υπολογίζεται κάθε μία από τις αποστάσεις, θα χρησιμοποιήσουμε για παράδειγμα τον παρακάτω Πίνακα 2.1, ως dataset:

USER/PRODUCT

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
u1	1	0	1	0	1	1	1	0	1	0
u2	0	1	0	1	0	0	0	0	1	1
u3	0	1	1	0	0	1	0	0	1	0
u4	1	0	0	0	0	1	0	0	1	1
u5	1	0	0	0	1	1	0	1	1	0
u6	1	0	0	0	1	1	0	0	1	1
u7	0	1	0	1	1	1	1	0	0	0
u8	0	1	1	1	0	0	0	1	0	1

Πίνακας 2.1

Στον πίνακα 2.1 έχουμε δημιουργήσει 8 users, είναι οι χρήστες και 10 products, τα προϊόντα. Συμβολίζουμε με «1» το γεγονός ότι ένας καταναλωτής έχει αγοράσει ένα προϊόν. Αντίστοιχα, συμβολίζουμε με «0», εάν κάποιος καταναλωτής δεν έχει αγοράσει κάποιο προϊόν.

Παρατηρώντας τις γραμμές του πίνακα 1, μπορούμε να κάνουμε την παραδοχή, ότι ένας χρήστης, μπορεί για τις συγκεκριμένες επιλογές του, να απεικονιστεί ως μια σειρά από bits (bit string), συγκεκριμένα για παράδειγμα, ο χρήστης u_1 = (1010111010) και γενικά για το χρήστη u_i και για ένα αριθμό k αντικειμένων p, θα είναι $u_i = (p_1 p_2 ... p_k)$. Συγκρίνοντας επομένως τα bit strings δύο χρηστών και λαμβάνοντας υπ' όψιν και τις αποστάσεις (distances), που αναλύουμε παρακάτω, θα μπορούμε να ανιχνεύσουμε την ύπαρξη ομοιότητας καταναλωτικής συμπεριφοράς μεταξύ αυτών.

2.7.1 Hamming Distance.

Η απόσταση Hamming μεταξύ δύο strings, ίσου μήκους, είναι ο αριθμός των θέσεων στις οποίες τα αντιπροσωπευτικά σύμβολα είναι διαφορετικά. Με άλλα λόγια, μετράει τον ελάχιστο αριθμό αντικαταστάσεων που απαιτούνται για να αλλάξει ένα string σε ένα άλλο, ή ο αριθμός των σφαλμάτων κατά το μετασχηματισμό ενός string σε ένα άλλο. Στα παρακάτω παραδείγματα, φαίνεται ακριβώς πώς υπολογίζεται η απόσταση Hamming.

<u>Παράδειγμα 1</u>

Λαμβάνοντας ένα παράδειγμα από τα δεδομένα του πίνακα 1, θέλοντας να συγκρίνουμε τα strings u3 και u6, θα έχουμε:

u3 = 0 1 1 0 0 1 0 0 1 0 και

u6 = 1 0 0 0 1 1 0 0 1 1

με κόκκινο χρώμα φαίνονται τα διαφορετικά στοιχεία των δύο strings και επομένως η απόσταση Hamming θα είναι 5.

Παράδειγμα 2

Υπολογίζοντας αντίστοιχα την απόσταση Hamming μεταξύ των strings u5 και u6, έχουμε:

u5 = 1 0 0 0 1 1 0 1 1 0 και

u6 = 1 0 0 0 1 1 0 0 1 1

Διαπιστώνουμε ότι η απόσταση Hamming είναι 2.

Γίνεται αντιληπτό ότι όσο πιο μικρή είναι η απόσταση, τόσο πιο κοντά στην ομοιότητα βρισκόμαστε.

2.7.2 Jaccard Distance.

Η απόσταση Jaccard, γνωστή και ως Συντελεστής Ομοιότητας Jaccard [2], συγκρίνει τις ομοιότητες και τις διαφορές σε set δεδομένων. Ορίζεται ως το πηλίκο της τομής δύο συνόλων, προς την ένωσή τους. Είναι δηλαδή, για δύο σύνολα A και B,

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

Εφαρμόζοντας τον ορισμό στα δύο προηγούμενα παραδείγματα, θα έχουμε:

Παράδειγμα 3

Στη σύγκριση των u3 και u6, έχουμε:

u3 ∩ u6 = 5 και

u3 υ u6 =20, άρα

J(u3,u6)=5/20=0,25.

Παράδειγμα 4

Στη συνέχεια στη σύγκριση των u5 και u6, έχουμε:

u5 ∩ u6 = 8 και

u5 υ u6 =20, επομένως

J(u5,u6)=0,4.

Είναι προφανές από τα παραδείγματα 3 και 4, ότι όσο πιο μεγάλο είναι το J, τόσο πιο κοντά σε ομοιότητα βρισκόμαστε.

2.7.3 Η χρησιμότητα των αποστάσεων.

Δουλεύοντας σε πραγματικές συνθήκες, ο όγκος των δεδομένων που θα έχουμε να διαχειριστούμε είναι πάρα πολύ μεγάλος. Ως εκ τούτου η ανεύρεση της απόλυτης ομοιότητας, εκτός του ότι είναι πρακτικά αδύνατη, είναι και ανώφελη. Το ενδιαφέρον μας πρέπει να εστιαστεί στην ανίχνευση χρηστών οι οποίοι ομοιάζουν σε συμπεριφορά, με άλλους. Έχουν δηλαδή μια τάση, να προμηθεύονται αντικείμενα που μεγάλο ποσοστό αυτών έχουν προμηθευτεί και οι εν δυνάμει - άλλοι - όμοιοι προς αυτούς καταναλωτές. Αυτό το ποσοστό

απόκλισης από την απόλυτη ομοιότητα ή αντίστοιχα η ανοχή στην επιλογή ενός αριθμού διαφορετικών προϊόντων, είναι αυτό που ορίζει η απόσταση.

Επιπλέον, για δυαδικά strings (όπως έχουμε στην περίπτωσή μας) η σύγκριση δύο από αυτά, πχ τα u5 και u6, για τον υπολογισμό της απόστασης Hamming, αντιστοιχεί στη λογική πράξη XOR. Το αποτέλεσμα δηλαδή θα είναι **u5 XOR u6**. Αντίστοιχα, προσπαθώντας να διατυπώσουμε τη συνθήκη ομοιότητας μεταξύ δύο καταναλωτών u_i και u_j, δεδομένης μιας απόστασης d, θα είναι: $\mathbf{u_i = u_j \Leftrightarrow u_i \text{ XOR } u_j \leq d}$. Επίσης, εάν ορίζαμε ως επιθυμητή απόσταση Hamming το 3, για τους καταναλωτές u5 και u6, τότε αυτοί θα πληρούσαν την προϋπόθεση ομοιότητας, αφού η μεταξύ τους απόσταση Hamming είναι 2. Αντίστοιχα οι u3 και u6 στο παράδειγμα 1, δε θα πληρούσαν την προϋπόθεση, αφού η μεταξύ τους απόσταση Hamming, είναι 5.

2.8 ΤΟ Hadoop.

Το Hadoop δημιουργήθηκε από τον Doug Cutting, το δημιουργό του Apache Lucene, που αποτελεί μια ευρέως χρησιμοποιούμενη βιβλιοθήκη αναζήτησης κειμένου. Το Hadoop έχει τις ρίζες της στον Apache Nutch, μια open source μηχανή αναζήτησης Ιστού, μέρος του Lucene project [12, 14, 21, 22, 23].

Το Nutch ξεκίνησε το 2002, και ένα πολύ ισχυρό σύστημα αναζήτησης κυριάρχησε πολύ γρήγορα. Ωστόσο, η αρχιτεκτονική του δεν ήταν εύκολο να κλιμακωθεί στα δισεκατομμύρια των σελίδων του Web. Βοήθεια ήρθε με τη δημοσίευση ενός εγγράφου, το 2003, που περιέγραφε την αρχιτεκτονική του συστήματος αρχείων διανομής (file system) της Google, που ονομάζεται GFS, το οποίο χρησιμοποιούσε στην παραγωγή της η Google. Το GFS, ή κάτι παρόμοιας λογικής με αυτό, θα έλυνε τα προβλήματα αποθήκευσης για τα πολύ μεγάλα αρχεία που δημιουργούνται ως μέρος της ανίχνευσης ιστού, αλλά και της διαδικασίας indexing. Ειδικότερα, το GFS, απαλλάσσει από χρόνο που δαπανάται σε administrative καθήκοντα, όπως η διαχείριση των κόμβων αποθήκευσης. Το 2004, έκανε την εμφάνισή της μια open source εφαρμογή, η Nutch Distributed File System (NDFS).

Το NDFS μαζί με την εφαρμογή του MapReduce ήταν εφαρμόσιμα και πέρα από τις εφαρμογές αναζήτησης, με αποτέλεσμα το Φεβρουάριο του 2006 να διαχωριστούν από το project Nutch και να σχηματιστεί ένα νέο, ανεξάρτητο υποέργο του Lucene, που ονομάστηκε Hadoop [14].

Παρότι το Hadoop είναι ευρύτερα γνωστό για το MapReduce και το HDFS (μετονομασία του NDFS), ο όρος χρησιμοποιήθηκε επίσης και για μια οικογένεια άλλων συναφών έργων που εμπίπτουν κάτω από την ομπρέλα των υπολογιστικών συστημάτων μεγάλης κλίμακας για επεξεργασία βάσεων δεδομένων [4].

Ενδεικτικά τέτοια Hadoop projects, είναι:

- **Common**. Μια σειρά από στοιχεία και interfaces για κατανεμημένα συστήματα αρχείων και γενικά I / O (serialization, Java RPC, ανθεκτικές δομές δεδομένων).

- **Map Reduce.** Ένα κατανεμημένο μοντέλο επεξεργασίας δεδομένων και περιβάλλον εκτέλεσης, που τρέχει στις μεγάλες συστάδες (clusters) μηχανών παραγωγής.

- **HDFS**. Ένα κατανεμημένο σύστημα αρχείων που τρέχει σε μεγάλα clusters μηχανών.

- **Pig**. Μια γλώσσα ροής δεδομένων και το αντίστοιχο περιβάλλον για εξερεύνηση και εκτέλεση πολύ μεγάλων συνόλων δεδομένων. Τρέχει σε συστήματα HDFS και Map Reduce clusters.

- **Hive**. Distributed Data Warehouse. Κυψέλη που διαχειρίζεται δεδομένα αποθηκευμένα σε HDFS και παρέχει γλώσσα που βασίζεται στην SQL (η οποία και μεταφράζεται από τη μηχανή που ενεργεί Map Reduce εργασίες) για την αναζήτηση των δεδομένων.

- **HBase**. Distributed, column-oriented database. Χρησιμοποιεί το HDFS για το αποθηκευμένο υλικό.

- **ZooKeeper**. Υπηρεσία συντονισμού. Παρέχει εργαλεία όπως για παράδειγμα κλειδώματα, που μπορούν να χρησιμοποιηθούν στο χτίσιμο εφαρμογών.

- **Sqoop**. Εργαλείο για την αποτελεσματική μετακίνηση δεδομένων μεταξύ σχεσιακών βάσεων δεδομένων και του HDFS.

2.8.1 Το Apache Hadoop.

Το Apache Hadoop [14] είναι το framework με το οποίο μπορούμε να εκμεταλλευτούμε τη δύναμη των δεδομένων, να χτίσουμε και να συντηρήσουμε συστήματα που θα είναι αξιόπιστα και επεκτάσιμα. Πρόκειται για μια εκτέλεση του Map Reduce που έχει αναπτυχθεί με ανοιχτό κώδικα (open source). Το Hadoop παρέχει ένα αξιόπιστο, κοινής αποθήκευσης και ανάλυσης σύστημα. Η αποθήκευση παρέχεται από το HDFS και η ανάλυση από το MapReduce [21, 22, 23]. Υπάρχουν και άλλα χαρακτηριστικά μέρη του Hadoop, αλλά αυτές οι δυνατότητες αποτελούν τον πυρήνα του.

Η δημιουργία του Hadoop ξεκίνησε όταν διαπιστώθηκε το πρόβλημα που υπάρχει για να γίνει συντονισμένη διαχείριση υπολογισμών έστω και σε λίγους ηλεκτρονικούς υπολογιστές. Η ακόμα μεγαλύτερη ανάγκη που δημιουργεί η διεύρυνση των υπολογισμών στο διαδίκτυο, ενήργησε ενισχυτικά σε αυτό.

Το Map Reduce είναι ένα προγραμματιστικό μοντέλο για επεξεργασία δεδομένων. Το μοντέλο είναι απλό, όχι όμως τόσο απλό ώστε να εκφράζονται σε αυτό διάφορα προγράμματα. Το Hadoop μπορεί να τρέχει Map Reduce προγράμματα που έχουν γραφτεί σε διάφορες γλώσσες, όπως Java, Ruby, Python, και C++. Και το πιο σημαντικό είναι ότι τα Map Reduce προγράμματα είναι παράλληλα. Αυτό σημαίνει ότι μεγάλος όγκος δεδομένων σε πολλές μηχανές, είναι εύκολο να γίνει αντικείμενο έρευνας ή επεξεργασίας [8].

Θα χρησιμοποιήσουμε ένα – βασικό – παράδειγμα, αυτό της εξόρυξης δεδομένων καιρού. Αισθητήρες εγκατεστημένοι σε σταθερά σημείο σε ένα μεγάλο γεωγραφικό εύρος, συγκεντρώνουν πληροφορίες για τον καιρό και αυτές θα πρέπει να αναλυθούν με το Map Reduce. Δεδομένα για το παράδειγμά μας αυτό, θα χρησιμοποιήσουμε από το National Climatic Data Center (NCDC, *http://www.ncdc.noaa.gov/* [21]). Τα δεδομένα είναι αποθηκευμένα ως ASCII χαρακτήρες σε γραμμές και αυτό πρακτικά σημαίνει ότι κάθε γραμμή αποτελεί μία ξεχωριστή εγγραφή. Απλοποιώντας το παράδειγμα, θα μας απασχολήσει μόνο η παράμετρος «θερμοκρασία».

Το Map Reduce, όπως είδαμε, λειτουργεί διαιρώντας τη διαδικασία σε δύο φάσεις. Τη φάση Map Και τη φάση Reduce. Η κάθε φάση έχει ζεύγη key-value ως είσοδο και έξοδο και οι δύο λειτουργίες, Map λειτουργία και Reduce λειτουργία, καθορίζονται από τον προγραμματιστή.

Η είσοδος για τη φάση Map είναι τα δεδομένα από το NCDC. Επιλέγουμε το format να είναι text, ώστε κάθε γραμμή της εισόδου να έρχεται ως text value. Στη Map διαδικασία θα πρέπει να «τραβήξουμε» το έτος και τη θερμοκρασία, αφού μόνο για αυτές τις δύο παραμέτρους ενδιαφερόμαστε. Επομένως η Map διαδικασία αποτελεί απλά μια φάση προετοιμασίας των δεδομένων και τροποποίησής τους έτσι, ώστε ο Reducer να μπορεί να κάνει σε αυτά τους υπολογισμούς του, που είναι ο υπολογισμός της μέγιστης θερμοκρασίας ανά έτος. Η Map διαδικασία επίσης, αποτελεί ένα καλό σημείο για να φιλτράρουμε λάθη στα δεδομένα μας, όπως για παράδειγμα θερμοκρασίες που λείπουν ή, δεν έχουν καταγραφεί, ή είναι λάθος.

Κατά την είσοδο λοιπόν στη Map διαδικασία του αρχείου δεδομένων, θα μπαίνουν ζεύγη που θα έχουν κλειδί την αύξουσα γραμμή και value τη θερμοκρασία και το έτος, όπως αυτά είναι καταγεγραμμένα στο αρχείο εισόδου. Θέλουμε αντίστοιχα στην έξοδο της Map φάσης να λάβουμε ζεύγη της μορφής:

(1950, 0)

(1950, 22)

(1950, −11)

(1949, 11)

(1949, 18)

Αντίστοιχα στην είσοδο της Reduce διαδικασίας θα πρέπει να διαβαστούν:

(1949, [11, 18])

(1950, [0, 22, −11])

Και το αποτέλεσμα που θα εξαχθεί (κατά την έξοδο της Reduce διαδικασίας) και θα είναι η μέγιστη κατά έτος θερμοκρασία, θα είναι:

(1949, 18)

(1950, 22)

Το παραπάνω αποτελεί ένα απλό παράδειγμα, με μικρή έκταση δεδομένων. Για να εξετάσουμε τη λειτουργία του Hadoop σε μεγάλο όγκο δεδομένων και σε διαφορετικούς υπολογιστές, nodes, πρέπει πρώτα να αναλύσουμε τον τρόπο λειτουργίας του στο Map Reduce.

Μια εργασία Map Reduce είναι ένα κομμάτι δουλειάς που αποτελείται από τα δεδομένα εισόδου, το πρόγραμμα Map Reduce και τις πληροφορίες διαμόρφωσης. Το Hadoop «τρέχει» την εργασία, διαιρώντας την σε δύο κύριους τύπους, τις εργασίες Map και τις εργασίες Reduce.

Υπάρχουν δύο τύποι nodes που ελέγχουν τη διαδικασία εκτέλεσης μιας εργασίας και είναι ο jobtracker, ο οποίος συντονίζει όλες τις εργασίες που τρέχουν στο σύστημα, με το να προγραμματίζει τις εργασίες να τρέχουν στους tasktrackers και ένας αριθμός από tasktrackers, οι οποίοι τρέχουν τις διαδικασίες και επιστρέφουν αναφορές προόδου στον jobtracker, ο οποίος κρατάει ένα κατάλογο για την πρόοδο των εργασιών. Εάν κάποια εργασία αποτύχει, ο jobtracker την αναπρογραμματίζει σε ένα διαφορετικό tasktracker.

Το Hadoop διαιρεί την είσοδο μιας Map Reduce διαδικασίας σε κομμάτια συγκεκριμένου μεγέθους, που ονομάζονται input splits, ή απλά splits. Στη συνέχεια δημιουργεί ένα Map task για κάθε ένα split, με αποστολή να τρέξει το πρόγραμμα για κάθε εγγραφή που βρίσκεται στο split. Χρησιμοποιώντας πολλά splits, σημαίνει ότι ο χρόνος που απαιτείται για να επεξεργαστεί κάθε ένα από τα split, είναι πολύ μικρός σε σχέση με το χρόνο που απαιτείται για να γίνει επεξεργασία όλων των δεδομένων εισόδου. Εάν δε, η επεξεργασία των splits γίνει παράλληλα, τότε και η επεξεργασία είναι καλύτερα επιμερισμένη. Εάν τα split είναι μικρά, μια γρήγορη μηχανή, θα μπορεί να επεξεργαστεί περισσότερα σε σχέση με μια πιο αργή μηχανή. Ακόμα και στην περίπτωση δύο όμοιων μηχανών, υπάρχουν πάντα αστοχίες ή εκτέλεση άλλων εργασιών, οι οποίες συντελούν στην άνιση κατανομή του φόρτου εργασιών.

Από την άλλη μεριά, με τη χρήση πολύ μικρών splits, υπάρχει πάντα ο κίνδυνος η διαχείριση τους να επικαλύπτει το συνολικό χρόνο που απαιτείται για να ολοκληρωθεί η εργασία. Για τις περισσότερες εργασίες ένα καλό μέγεθος split, είναι όσο ένα block του HDFS, δηλαδή 64MB, αν και αυτό μπορεί να μεταβληθεί κατά περίπτωση.

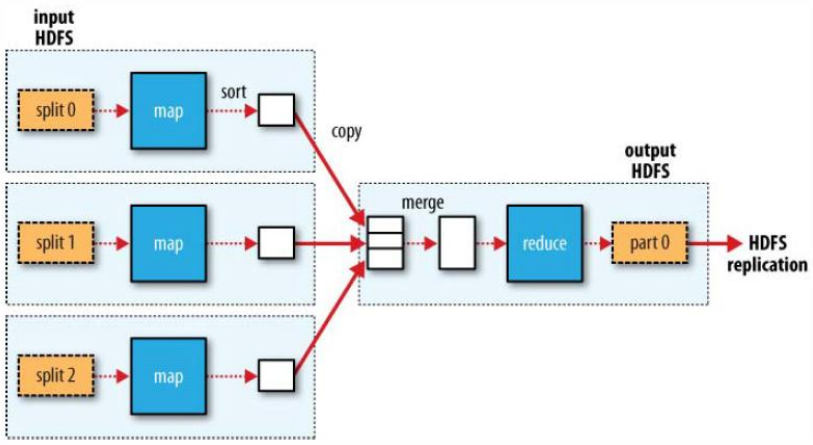
Το Hadoop, βελτιστοποιώντας τη λειτουργία του, προσπαθεί να «τρέξει» τη Map διαδικασία σε ένα node, όπου και έχουν τοποθετηθεί από τον HDFS τα δεδομένα. Η βελτιστοποίηση αυτή καλείται «*data locality optimization*». Ταυτόχρονα, γίνεται πλέον ξεκάθαρο, το γιατί το μέγιστο μέγεθος ενός split είναι όσο και το μέγεθος ενός block: Είναι το μέγιστο μέγεθος το οποίο

εγγυημένα μπορεί να αποθηκευτεί σε ένα node. Εάν το split χωριζόταν σε δύο blocks, η πιθανότητα ο HDFS να αποθήκευε αυτά τα δύο split στον ίδιο node, θα ήταν εξαιρετικά μικρή. Έτσι θα απαιτούταν μεταφορά στοιχείων στο δίκτυο, κάτι που θα σήμαινε μεγάλη καθυστέρηση και αντίστοιχα τη μη έξυπνη κατανομή της λειτουργίας του συστήματος.

Οι εργασίες Map γράφουν τα αποτελέσματα εξόδου στον τοπικό δίσκο και όχι στο HDFS. Η έξοδος της Map διαδικασίας είναι ενδιάμεσο αποτέλεσμα και χρησιμοποιείται από τη Reduce διαδικασία, προκειμένου να εξαχθεί το τελικό αποτέλεσμα και από τη στιγμή που θα χρησιμοποιηθεί, αποτελεί άχρηστη πληροφορία και μπορεί να απορριφτεί. Επομένως αποθηκεύοντας στο HDFS, και λαμβάνοντας υπ' όψη και τα απαιτούμενα replications θα είχαμε πολύ μεγάλη αύξηση χρήσης πόρων. Σε κάθε περίπτωση και εάν η διαδικασία Map για κάποιο λόγο αποτύχει, το Hadoop αυτόματα θα την αναθέσει σε ένα νέο κόμβο, για να ξεκινήσει από την αρχή.

Αντίθετα, οι διαδικασίες Reduce, δεν έχουν το περιθώριο του data locality. Θυμίζουμε εδώ, ότι η είσοδος για μια διαδικασία Reduce, είναι η έξοδος περισσότερων του ενός Mappers. Οι ταξινομημένες έξοδοι των διαδικασιών Map, θα πρέπει να οδηγηθούν μέσω του δικτύου, στον κόμβο όπου η διαδικασία Reduce πρόκειται να εκτελεστεί. Σε αυτό το συγκεκριμένο κόμβο θα συνενωθούν και θα περάσουν ως δεδομένα εισόδου στο Reducer. Η έξοδος από το Reduce, αποθηκεύεται συνήθως στον HDFS για λόγους ασφαλείας και γιατί είναι και αυτή που μας ενδιαφέρει.

Ολόκληρη η ροή δεδομένων με τη χρήση μίας μόνο Reduce διαδικασίας, εικονίζεται στο παρακάτω σχήμα (σχήμα 2.6).

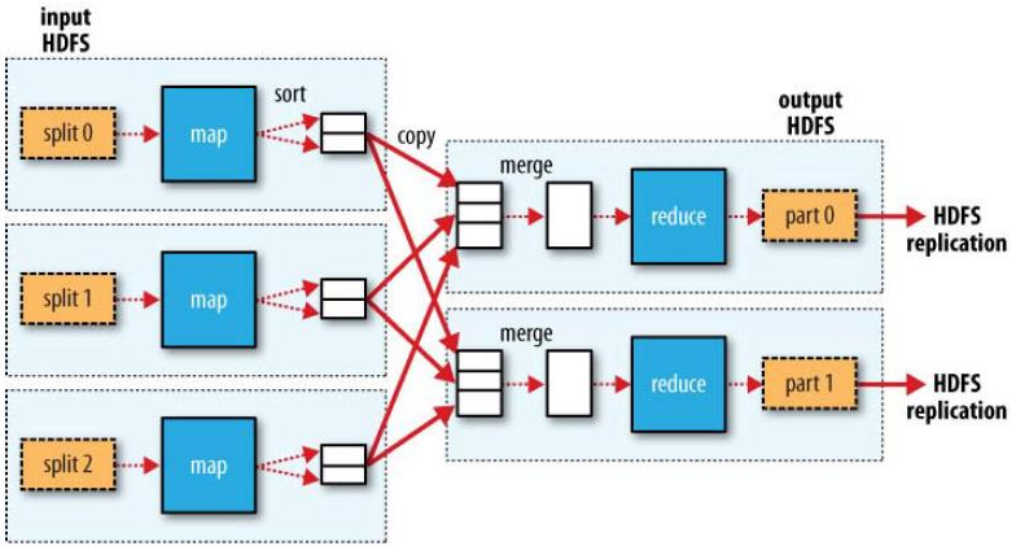


Σχήμα 2.6. Η ροή δεδομένων στο Hadoop με μία Reduce διαδικασία.

Ο αριθμός των Reduce διαδικασιών, δεν καθορίζεται από το μέγεθος των δεδομένων, αλλά ανεξάρτητα.

Στην περίπτωση που έχουμε περισσότερους από ένα Reducers, οι Mappers κατακερματίζουν την έξοδό τους και την ταξινομούν με βάση το κλειδί από το κάθε παραγόμενο ζεύγος. Με βάση αυτό το κλειδί ταξινομούν την έξοδό τους ως είσοδο στον κατάλληλο Reducer. Ο τρόπος που γίνεται ο διαχωρισμός μπορεί να καθοριστεί από το χρήστη, αλλά και ο διαχωρισμός που κάνει το σύστημα από μόνο του, είναι εξίσου αποτελεσματικός.

Η αντίστοιχη ροή δεδομένων για την περίπτωση περισσοτέρων του ενός Reducers, εικονίζεται στο παρακάτω σχήμα (σχήμα 2.7).



Σχήμα 2.7. Η ροή εργασιών στο Hadoop, για περισσότερους του ενός Reducers.

Διακρίνουμε εδώ ξεκάθαρα, το γιατί η διαδικασία Map Reduce χαρακτηρίζεται ως «το ανακάτεμα», αφού κάθε μία διαδικασία Reduce, τρέφεται από πολλές διαδικασίες Map.

2.9 ΤΟ HDFS.

Όταν ένα set δεδομένων (dataset) ξεπερνά το χώρο αποθήκευσης που είναι διαθέσιμος σε μια μηχανή, εμφανίζεται η ανάγκη να κατατμηθεί ανάμεσα σε ένα αριθμό μηχανών. Συστήματα αρχείων που διαχειρίζονται την αποθήκευση σε μηχανές που ανήκουν στο ίδιο δίκτυο,

ονομάζονται Distributed FileSystems [2, 14]. Με δεδομένο το ότι βασίζονται σε πολλές μηχανές που βρίσκονται στο δίκτυο, ισχύουν όλες οι επιπλοκές που απορρέουν από τον διαδικτυακό προγραμματισμό και αυτό κάνει τα Distributed FileSystems να είναι πιο σύνθετα από ότι τα απλά - τοπικά FileSystems. Το Hadoop συνοδεύεται από το δικό του Distributed FileSystem, το επονομαζόμενο HDFS (Hadoop Distributed FileSystem), το οποίο μπορεί να συναντήσουμε και απλά ως DFS.

2.9.1 Ο σχεδιασμός του HDFS.

Το HDFS είναι ένα σύστημα αρχείων για αποθήκευση πολύ μεγάλων αρχείων δεδομένων, το οποίο κάνει πρόσβαση σε δεδομένα, τα οποία «τρέχουν» σε μεγάλα – κοινού τύπου -hardware συστήματα, με streaming.

Ας εξετάσουμε ξεχωριστά τους όρους της παραπάνω πρότασης, για να κατανοήσουμε καλύτερα τη λειτουργία του HDFS.

- Πολύ μεγάλα αρχεία. Αυτό μεταφράζεται σε αρχεία τουλάχιστον εκατοντάδων Mbytes. Υπάρχουν clusters του Hadoop, που σήμερα αποθηκεύουν petabytes δεδομένων.

- Πρόσβαση δεδομένων με streaming. Το HDFS έχει δομηθεί γύρω από την ιδέα της κατά το δυνατόν αποτελεσματικότερης πρόσβασης στα δεδομένα, με τη φιλοσοφία «κάνε εγγραφή μία φορά – πάρε πρόσβαση σε αυτήν πολλές φορές». Κάθε αποθηκευμένο set δεδομένων υπόκειται σε πολλαπλές αναλύσεις. Κάθε ανάλυση χρησιμοποιεί ένα πολύ μεγάλο – αν όχι ολόκληρο – ποσοστό δεδομένων. Επομένως, ο χρόνος ανάγνωσης δεδομένων, είναι πολύ πιο σημαντικός από το χρόνο που απαιτείται για την ανάγνωση της πρώτης εγγραφής των δεδομένων.

- Κοινού τύπου hardware. Το Hadoop δεν απαιτεί ακριβά, υψηλής αξιοπιστίας, εξοπλισμό hardware. Είναι σχεδιασμένο να τρέχει σε clusters, σε κοινό εμπορικό hardware, για το οποίο η περίπτωση αποτυχίας πρόσβασης κάποιου node σε ένα cluster είναι πολύ υψηλή. Το HDFS είναι σχεδιασμένο να τρέχει με τέτοιο τρόπο που να μην επηρεάζεται από αποτυχίες υλικού.

Αξίζει στο σημείο αυτό να δούμε κάποιες περιπτώσεις στις οποίες δεν ενδείκνυται να χρησιμοποιήσουμε το HDFS, τουλάχιστον με τη χρήση της τεχνολογίας του σήμερα και με την επιφύλαξη ότι αυτό μπορεί στο κοντινό μέλλον να αλλάξει.

- Πρόσβαση σε low – latency δεδομένα. Εφαρμογές που απαιτούν low – latency πρόσβαση δεδομένων της τάξης των δεκάδων millisecond, δε θα λειτουργήσουν καλά με το HDFS. Το HDFS έχει τη βέλτιστη συμπεριφορά κατά τη διαχείριση μεγάλων ποσοτήτων δεδομένων, με κόστος στην πρόσβαση στη λανθάνουσα. Η HBase είναι πιο κατάλληλη για την περίπτωση αυτή.

- Μεγάλος αριθμός μικρών αρχείων. Υποθέτοντας ότι ένα αρχείο καταλαμβάνει στο τοπικό δίσκο, τουλάχιστον 150 bytes, ένας μεγάλος αριθμός αρχείων θα καταλάμβανε αντίστοιχα και ένα μεγάλο μέρος αποθηκευτικού χώρου σε blocks.

- Αρχεία τα οποία επεξεργάζονται πολλοί χρήστες, ή αυθαίρετες μεταβολές αρχείων. Τα αρχεία του HDFS πρέπει να έχουν εγγραφεί από ένα χρήστη. Δεν παρέχεται υποστήριξη πολλαπλών χρηστών – συγγραφέων, ή αυθαίρετης, ανοργάνωτης παρέμβασης στα αρχεία από περισσότερα από ένα πρόσωπα.

2.9.2 Blocks.

Ένας δίσκος έχει ένα μέγεθος block, το οποίο είναι το ελάχιστο μέγεθος δεδομένων που μπορεί να διαβάσει, ή να γράψει. Το σύστημα διαχείρισης αρχείων σε ένα δίσκο, εξετάζει τα δεδομένα στα blocks, τα οποία είναι ακέραιο πολλαπλάσιο του μεγέθους των blocks στο δίσκο. Τα block σε ένα σύστημα διαχείρισης αρχείων είναι μεγέθους λίγων kilobytes, ενώ τα blocks ενός δίσκου, είναι συνήθως 512 kilobytes. Αυτό περνάει απαρατήρητο στο χρήστη, ο οποίο διαβάζει ή γράφει ένα αρχείο, ανεξαρτήτως μεγέθους.

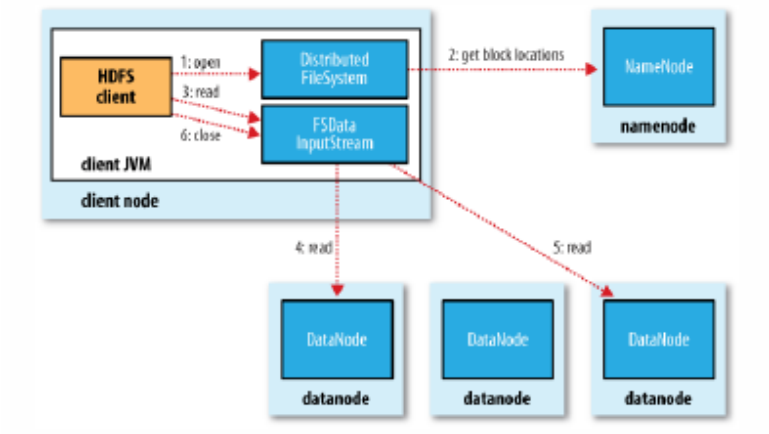
Το HDFS έχει επίσης την έννοια του block, αλλά αποτελεί μια πολύ μεγαλύτερη μονάδα, περίπου 64 MB εξ ορισμού. Επίσης τα αρχεία διαιρούνται σε κομμάτια που ανήκουν σε blocks και αποθηκεύονται σε ανεξάρτητες μονάδες. Αντίθετα από το σύστημα σε ένα απλό δίσκο, το αρχείο στο HDFS, όταν είναι μικρότερο από το μέγεθος του block, δεν το καταλαμβάνει ολόκληρο.

Συγκριτικά με τα blocks των δίσκων, τα HDFS blocks είναι κατά πολύ μεγαλύτερα. Ο λόγος είναι η ελαχιστοποίηση του κόστους αναζήτησης. Κάνοντας ένα block μεγάλο, ο χρόνος της μεταφορά των δεδομένων από το δίσκο, μπορεί να γίνει χαρακτηριστικά μεγαλύτερος από το χρόνο αναζήτησης από την αρχή του block. Άρα, ο χρόνος που απαιτείται για να μεταφερθεί ένα αρχείο που αποτελείται από πολλά blocks, επηρεάζει το εύρος του χρόνου μεταφοράς του δίσκου. Πραγματοποιώντας ένα μικρό υπολογισμό, εάν ο χρόνος αναζήτησης είναι 10 ms και υπάρχει ένα εύρος μεταφοράς που είναι 100 MB/s, τότε για να πετύχουμε ο χρόνος αναζήτησης να είναι το 1% του χρόνου μεταφοράς, απαιτείται να κάνουμε το μέγεθος του block 100 MB. Το default είναι 64 MB, αν και πολλές εγκαταστάσεις HDFS, χρησιμοποιούν blocks των 128 MB.

Η χρήση blocks με την αφηρημένη έννοια, μπορεί να αποφέρει αρκετά οφέλη. Πρώτα από όλα, το προφανές, ένα αρχείο μπορεί να είναι μεγαλύτερο από ένα απλό δίσκο στο δίκτυο. Δεν υπάρχει δηλαδή καμία απαίτηση να είναι τα αρχεία αποθηκευμένα σε ένα και μόνο δίσκο στο ίδιο δίκτυο. Για την ακρίβεια, υπάρχει η δυνατότητα, αν και δε συνηθίζεται, ένα αρχείο να χρησιμοποιεί αποθηκευτικά το HDFS με τα blocks του να κάνουν χρήση όλων των δίσκων στο cluster. Δεύτερον, η χρήση block αντί για αρχείο, απλουστεύει το αποθηκευτικό σύστημα. Η απλότητα επιδιώκεται γενικά σε όλα τα συστήματα, αλλά είναι ιδιαίτερα σημαντική στα κατανεμημένα συστήματα, όπου οι αιτίες αστοχίας ποικίλουν. Το αποθηκευτικό υποσύστημα έχει να κάνει με blocks, απλοποιώντας την αποθηκευτική διαχείριση, μια και τα blocks έχουν συγκεκριμένο μέγεθος και επομένως μπορεί να γίνει υπολογισμός πόσα μπορούν να «χωρέσουν» σε ένα δεδομένο δίσκο, ενώ ταυτόχρονα εξαλείφεται κάθε ανησυχία για τη διαχείριση των λεγόμενων metadata, αφού μπορεί να επιλεγεί ένα ξεχωριστό υποσύστημα διαχείρισης αυτών. Επιπλέον, τα blocks είναι δυνατό να μπουν σε μια διαδικασία replication, με αποτέλεσμα να υπάρχει μεγαλύτερη ανοχή στα σφάλματα και περισσότερη διαθεσιμότητα. Για εξασφάλιση από κατεστραμμένα blocks, ή δίσκο, ή αποτυχία της μηχανής, το κάθε block αντιγράφεται σε ένα μικρό αριθμό διαφορετικών μηχανών (τυπικά τρεις). Εάν ένα block πάψει να είναι διαθέσιμο, ένα αντίγραφό του διαβάζεται από ένα άλλο σημείο, με τρόπο τέτοιο που να είναι διαθέσιμο στον client. Με παρόμοιο τρόπο, δύνεται η δυνατότητα επιλογής υψηλού παράγοντα replication των blocks, ειδικά σε δημοφιλή αρχεία, ώστε να διαμοιραστεί ο φόρτος προσπέλασης σε ολόκληρο το cluster.

2.9.3 Πώς διαβάζεται ένα αρχείο.

Για να πάρουμε μια ιδέα του πώς ένα αρχείο διαβάζεται στο HDFS, παραθέτουμε το παρακάτω σχήμα (σχήμα 2.8).



Σχήμα 2.8. Ανάγνωση αρχείου στο HDFS.

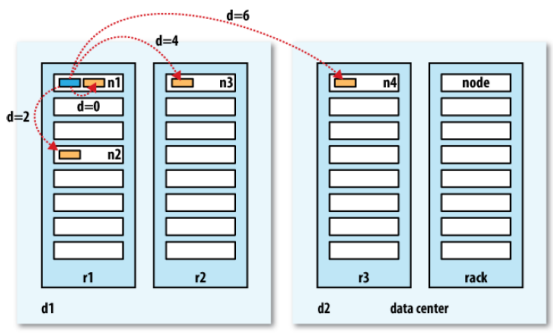
Ο client ανοίγει το αρχείο που επιθυμεί να διαβάσει (βήμα 1). Στη συνέχεια καλείται ο NameNode για να καθορίσει τις τοποθεσίες των πρώτων blocks του αρχείου (βήμα 2). Για κάθε block, ο NameNode επιστρέφει τις διευθύνσεις των DataNodes που έχουν ένα αντίγραφο αυτού του block. Επιπλέον οι DataNodes ταξινομούνται με βάση την εγγύτητά τους στον client. Εάν και ο client είναι ένας DataNode, θα διαβάσει από τον εαυτό του, εάν φιλοξενεί κάποιο αντίγραφο του block. Στη συνέχεια ο client καλεί εντολή ανάγνωσης, read (βήμα 3). Στη συνέχεια γίνεται σύνδεση με τον πρώτο (και εγγύτερο) DataNode που έχει καταγραφεί, για το πρώτο block του αρχείου. Τα δεδομένα διοχετεύονται από τον DataNode στον client, ο οποίος εκτελεί εντολή ανάγνωσης επαναληπτικά (βήμα 4). Όταν φθάσει το τέλος του block κλείνει η σύνδεση με το συγκεκριμένο DataNode και στη συνέχεια αναζητιέται ο βέλτιστος DataNode για το επόμενο block (βήμα 5). Όλο αυτό πραγματοποιείται από τον client, αλλά από τη δική του οπτική, απλά γίνεται διαρκής διοχέτευση δεδομένων σε αυτόν. Τα blocks διαβάζονται με τη σειρά και ανοίγουν νέες συνδέσεις με τους DataNodes, καθόσον στον client διοχετεύονται δεδομένα. Καλεί επίσης και το NameNode, προκειμένου να ανακτήσει την επόμενη ομάδα από blocks για ανάγνωση. Όταν ο client ολοκληρώσει την ανάγνωση, καλεί μια διαδικασία κλεισίματος (close) (βήμα 6). Κατά τη διαδικασία ανάγνωσης, εάν διαπιστωθεί σφάλμα ή αστοχία, επιχειρείται να αναγνωστεί ο επόμενος πλησιέστερος DataNode για το συγκεκριμένο block. Επίσης το σύστημα θυμάται τους προβληματικούς DataNodes, ώστε να μην ανατρέξει σε αυτούς χωρίς να υπάρχει ιδιαίτερη ανάγκη, καθ' όλη τη διαδικασία ανάγνωσης.

Ολόκληρη η παραπάνω διαδικασία ανάγνωσης έχει ένα πολύ σημαντικό χαρακτηριστικό, το γεγονός ότι ο client επικοινωνεί απ' ευθείας με τους DataNodes για να ανακτήσει δεδομένα, ενώ καθοδηγείται από το NameNode στον πιο κατάλληλο DataNode για κάθε block. Ο σχεδιασμός αυτός, επιτρέπει στον HDFS να αναπτυχθεί ταυτόχρονα σε πολλούς clients, μια και τα δεδομένα είναι απλωμένα σε όλους τους DataNodes του cluster. Στο μεταξύ, ο NameNode απλώς πρέπει να εξυπηρετεί αιτήματα εντοπισμού blocks, ενώ – για παράδειγμα – δεν είναι επιφορτισμένος και με το να εξυπηρετεί δεδομένα, κάτι που θα οδηγούσε σύντομα τη διαδικασία να καθυστερήσει πολύ, όσο ο αριθμός των clients θα μεγαλώνει.

2.9.4 Τοπολογία δικτύου στο Hadoop.

Θα εξηγήσουμε εδώ, τι ακριβώς σημαίνει να είναι «κοντά» δύο nodes σε ένα δίκτυο. Στο πλαίσιο της επεξεργασίας δεδομένων μεγάλου μεγέθους, ο περιοριστικός παράγοντας είναι ο βαθμός κατά τον οποίο μπορούμε να διακινήσουμε δεδομένα μεταξύ δύο κόμβων (nodes). Το bandwidth σπάνια έχει χρησιμότητα σε αυτό. Επομένως η ιδέα είναι να χρησιμοποιήσουμε το bandwidth μεταξύ δύο κόμβων ως το μέτρο της μεταξύ τους απόστασης.

Το Hadoop, αντί να χρησιμοποιήσει το bandwidth μεταξύ των κόμβων, κάτι που πρακτικά είναι δύσκολο να γίνει, χρησιμοποιεί μια απλή προσέγγιση, σύμφωνα με την οποία το δίκτυο απεικονίζεται ως ένα δένδρο και η απόσταση μεταξύ δύο κόμβων είναι το άθροισμα των αποστάσεών τους από τον πλησιέστερο κοινό τους γονέα. Τα επίπεδα στο δένδρο δεν είναι προκαθορισμένα, αλλά είναι συνηθισμένο να έχουν επίπεδα που ιεραρχούνται στο κέντρο δεδομένων (data center), το rack και τον κόμβο (node) στον οποίο τρέχει η διαδικασία, όπως εικονίζεται και στο παρακάτω σχήμα (Σχήμα 2.9).



Σχήμα 2.9. Ιεράρχηση επιπέδων στο δένδρο.

Με την ιεράρχηση αυτή, το bandwidth που είναι διαθέσιμο για κάθε ένα από τα παρακάτω σενάρια προοδευτικά λιγοστεύει:

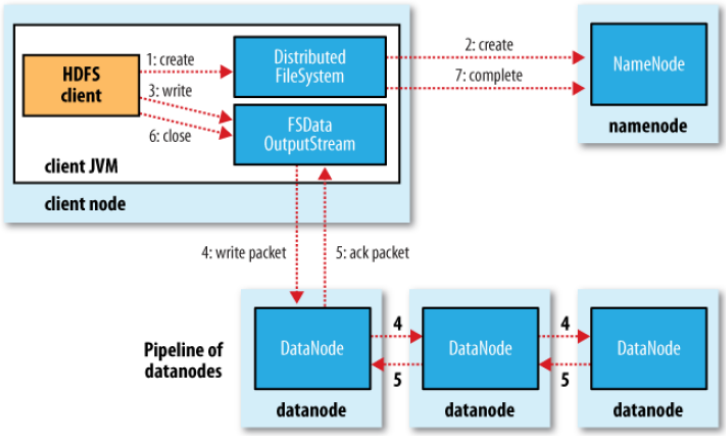
- Διαδικασίες στον ίδιο κόμβο.
- Διαφορετικοί κόμβοι στο ίδιο rack.
- Κόμβοι σε διαφορετικά rack στο ίδιο data center.
- Κόμβοι σε διαφορετικά data centers.

Με βάση το παραπάνω σχήμα, εάν έχουμε ένα κόμβο n1, σε ένα rack r1, στο data center d1 και για συντομία το αναπαραστήσουμε ως d1/r1/n1, οι αποστάσεις που προκύπτουν χρησιμοποιώντας αυτή τη σημειογραφία, θα είναι:

- Απόσταση(/d1/r1/n1, /d1/r1/n1) = 0 (διαδικασίες στον ίδιο κόμβο).
- Απόσταση(/d1/r1/n1, /d1/r1/n2) = 2 (διαφορετικοί κόμβοι στο ίδιο rack).
- Απόσταση(/d1/r1/n1, /d1/r2/n3) = 4 (κόμβοι σε διαφορετικό rack στο ίδιο data center).
- Απόσταση(/d1/r1/n1, /d2/r3/n4) = 4 (κόμβοι σε διαφορετικά data centers).

2.9.5 Πώς γράφεται ένα αρχείο.

Θα εξετάσουμε στη συνέχεια πώς εγγράφεται ένα αρχείο στον HDFS. Η περίπτωση που θα εξετάσουμε περιλαμβάνει τη δημιουργία ενός αρχείου, την εγγραφή δεδομένων σε αυτό και τέλος το κλείσιμό του. Σχηματικά όλο το παραπάνω εικονίζεται στο παρακάτω σχήμα (σχήμα 2.10).



Σχήμα 2.10. Διαδικασία ανάγνωση αρχείου.

Ο client δημιουργεί ένα αρχείο με την εντολή create() (βήμα 1). Στη συνέχεια καλείται ο NameNode να δημιουργήσει ένα αρχείο στο χώρο των ονομάτων του συστήματος αρχείων, χωρίς όμως να συνδέονται με αυτό blocks (βήμα 2). Ο NameNode πραγματοποιεί μια σειρά ελέγχων ότι το όνομα του αρχείου δεν υπάρχει ήδη και ότι ο client έχει τα κατάλληλα permissions να το δημιουργήσει. Εάν αυτοί οι έλεγχοι περάσουν με επιτυχία, ο NameNode δημιουργεί μια εγγραφή με αυτό το όνομα αρχείου, διαφορετικά η δημιουργία νέου αρχείου αποτυγχάνει. Καθώς ο client γράφει δεδομένα (βήμα 3) σε μια εσωτερική ουρά (data queue), ο NameNode κατανέμει νέα blocks με την επιλογή μιας λίστας από DataNodes στους οποίους θα αποθηκεύσει τα αντίγραφα. Η λίστα των DataNodes δημιουργεί ένα αγωγό. Έστω ότι το επίπεδο replication είναι 3. Αυτό σημαίνει ότι υπάρχουν τρεις κόμβοι στον αγωγό. Τα πακέτα διοχετεύονται στον πρώτο DataNode του αγωγού, ο οποίος αποθηκεύει το πακέτο και το προωθεί στον επόμενο, δεύτερο DataNode. Όμοια από το δεύτερο πηγαίνει και στον τρίτο (βήμα 4). Δημιουργείται επίσης και μια ουρά από πακέτα που περιμένουν να αναγνωριστούν από τους DataNodes, η λεγόμενη ack queue. Το πακέτο μετακινείται από την ack queue, μόλις αναγνωριστεί από όλους τους DataNodes στον αγωγό (βήμα 5). Εάν ένας DataNode αποτύχει την ώρα που εγγράφονται σε αυτόν δεδομένα, τότε ο αγωγός κλείνει και όλα τα πακέτα της ack queue προστίθενται μπροστά στην ουρά εκτέλεσης, ώστε να μη χαθούν πακέτα. Το τρέχον block στους DataNodes που λειτουργούν ακόμα σωστά, παίρνει μια νέα ταυτότητα, η οποία επικοινωνείται και στο NameNode, έτσι ώστε ο αποτυχημένος DataNode να διαγραφεί στη συνέχεια. Κατόπιν, τα εναπομείναντα δεδομένα στα block εγγράφονται στου δύο καλούς DataNodes του αγωγού. Ο NameNode γνωρίζοντας ότι το replication δεν έχει ολοκληρωθεί όπως θα έπρεπε, αφού του λείπει ένας ακόμα DataNode, τον ζητάει από ένα άλλο κόμβο.

Όταν ο client ολοκληρώσει την εγγραφή δεδομένων, καλεί μια διαδικασία close() (βήμα 6). Η διαδικασία αυτή στέλνει όλα τα εναπομείναντα πακέτα στον αγωγό των DataNodes και περιμένει επιβεβαίωση, πριν επικοινωνήσει με τον NameNode για να σηματοδοτήσει ότι το αρχείο είναι έτοιμο (βήμα 7). Ο NameNode γνωρίζει επίσης από ποια blocks φτιάχτηκε το αρχείο, ώστε να έχει να περιμένει μόνο τα blocks να γίνουν replicated, προκειμένου να επιστρέψει τη λήξη της διαδικασίας με επιτυχία.

ΚΕΦΑΛΑΙΟ 3
ΑΛΓΟΡΙΘΜΟΙ

3.1. ΑΛΓΟΡΙΘΜΟΙ ΕΥΡΕΣΗΣ ΟΜΟΙΟΤΗΤΑΣ.

Για την εύρεση ομοιότητας έχουν χρησιμοποιηθεί πολλοί αλγόριθμοι. Κάποιοι από αυτούς με πολύ καλά αποτελέσματα, τόσο σε ταχύτητα, όσο και σε υπολογιστική ισχύ. Ωστόσο ο τρόπος προσέγγισης στο πρόβλημα, αλλά και το είδος των αποτελεσμάτων διαφέρει, με αποτέλεσμα κάποιοι από αυτούς να καθίστανται περισσότερο κατάλληλοι από ότι κάποιοι άλλοι. Παρακάτω αναλύουμε κάποιους από τους αλγορίθμους αυτούς, κάθε ένας εκ των οποίων αποτελεί και μια ξεχωριστή κατηγορία, ένα ξεχωριστό είδος προσέγγισης. Να σημειώσουμε εδώ, ότι οι αλγόριθμοι που παρατίθενται, είτε ονομαστικά, είτε μελετώνται, είναι οι λεγόμενοι Map Reducible αλγόριθμοι, εκείνοι δηλαδή που μπορεί να τροποποιηθούν και να εφαρμοστούν στο Map Reduce.

Παρουσιάζουμε λοιπόν παρακάτω, αρχικά τον «αφελή» αλγόριθμο, ως μια πρώτη προσέγγιση στο ζητούμενο, καταδεικνύοντας ταυτόχρονα και τα σημεία τα οποία αποτελούν αδυναμίες.

Στη συνέχεια εξετάζουμε τον Ball-Hashing, ο οποίος όμως είναι ένας αλγόριθμος που βρίσκει πλήρη ομοιότητα, κάτι που απέχει από το δικό μας ζητούμενο. Ενδιάμεσα, υπάρχει και ο Ball-Hashing 2, με ελαφρές τροποποιήσεις, ο Splitting, ο Hamming Code και ο Anchor Points.

Καταλήγουμε στο MinHashing που είναι και ο αλγόριθμος που θα στηριχτούμε και θα αναλύσουμε και περισσότερο, αφού ταιριάζει στα χαρακτηριστικά που θέτουμε για την υλοποίηση του πειραματικού μέρους. Είναι δηλαδή ένας αλγόριθμος που βρίσκει ομοιότητα κατά προσέγγιση, εύκολα τροποποιούμενος ως προς την παραμετροποίησή του και εύκολα τυποποιούμενος.

Η εξέλιξη του MinHashing είναι ουσιαστικά ο LSH,που επιλύει κάποιες ανάγκες σε υπολογιστική ισχύ, γίνεται όμως λιγότερο ακριβής ως προς τον εντοπισμό ομοιοτήτων [2, 3].

3.2 Ο «ΑΦΕΛΗΣ» ΑΛΓΟΡΙΘΜΟΣ (Naive algorithm).

Συνηθίζουμε να αποκαλούμε «Naive» ένα αλγόριθμο όταν είναι απλός, χωρίς όμως να επιδεικνύει ένα επιθυμητό επίπεδο αποδοτικότητας, συνήθως σε παραμέτρους που αφορούν στο χρόνο εκτέλεσης, ή στη διαχείριση των πόρων του συστήματος, ή ακόμα και στη σωστή, ή τη βέλτιστη λύση. Συνήθως επίσης αποτελεί τη βάση του σκεπτικού εύρεσης καλύτερων αλγορίθμων σε σχεδίαση και εφαρμογή [5]. Αναφέρεται εδώ ως η θεμελιώδης υποδομή λογικής για την ανίχνευση ομοιοτήτων, αν και ως προς την υλοποίησή του θα πρέπει να ξεπεραστούν αρκετά εμπόδια, ώστε να γίνει πραγματικά παραγωγικός, επομένως μετά απλά θα προκύπτουν υλοποιήσεις του που έχουν ονομαστεί διαφορετικά και αποτελούν εξέλιξή του.

Ο αλγόριθμος αυτός, λειτουργεί για οποιοδήποτε τύπο δεδομένων και για οποιαδήποτε similarity function. Υποθέτοντας ότι η είσοδός μας είναι ένα set S από στοιχεία κάποιου τύπου, θα κανονίσουμε τους K Reducers που θα έχουμε, σε ένα τρίγωνο, έτσι ώστε κάθε ζεύγος από στοιχεία του S, να εμφανίζεται σε ένα μόνο reducer και να συγκρίνεται εκεί. Για το σχηματισμό του τριγώνου, κάθε reducer προσδιορίζεται από ένα ζεύγος (i , j), τέτοιο ώστε $0 \leq i \leq j < J$, για ένα σταθερό J. Ο αριθμός των reducers που θα χρειαστούν, θα είναι $K=J(J+1)/2$.

Τα στοιχεία του input set, οδεύονται σε J θέσεις, από το 0 έως και το J-1. Ένα στοιχείο που οδεύεται στη θέση i, στέλνεται από το mapper, σε όλους τους reducers που το ζεύγος

αναγνώρισής τους είναι το (i , j), ή το (j , i), για κάποιο j. Επομένως, κάθε στοιχείο της εισόδου οδεύεται και σε ένα reducer, αφού τα ζεύγη (i , j) και (j , i) πηγαίνουν στον ίδιο reducer, εκτός και αν i = j, οπότε και πάλι για ένα reducer πρόκειται.

Πρακτικά, χρησιμοποιώντας ένα naive αλγόριθμο στην ανίχνευση ομοιότητας με χρήση του Map Reduce, σε dataset που αποτελείται από ένα πίνακα που έχει ως γραμμές χρήστες και ως στήλες αντικείμενα που έχουν ή όχι προμηθευτεί οι χρήστες (απεικονίζονται με 1 ή 0 αντίστοιχα) αντιμετωπίζουμε το πρόβλημα της ανίχνευσης όμοιων δυαδικών strings. Θα πρέπει λοιπόν σε κάθε εκτέλεση, να χρησιμοποιούμε ένα από τα strings ως pattern με το οποίο θα συγκρίνουμε όλα τα υπόλοιπα και θα αποφασίζουμε για τη μεταξύ τους ομοιότητα με βάση τη distance d που θα έχουμε προαποφασίσει. Για κάθε string που θα αποτελεί το pattern και για όλα τα υπόλοιπα που θα συγκριθούν με αυτό, θα απαιτηθεί ένας Reducer. Θεωρητικά λοιπόν θα χρειαστούμε τόσους Reducers, όσοι είναι και οι χρήστες. Εάν από αυτή τη διαδικασία εξαιρέσουμε τους ελέγχους που θα γίνουν διπλοί, τότε θα χρειαστούμε τους μισούς.

Μια διατύπωση του αλγορίθμου μπορεί να είναι ως εξής:

1. Έλεγξε εάν υπάρχουν περισσότερα από ένα strings και αν ναι βάλε τα στην είσοδο, καθώς και τη distance d.

2. Ονόμασε το πρώτο string s1.

3. Εάν υπάρχει, ονόμασε το επόμενο string s2.

    a. Εάν δεν υπάρχει επόμενο string, βάλε στην είσοδο όλα τα strings εκτός από το s1 και τη distance d.

    b. Πήγαινε στο βήμα 2.

4. Σύγκρινε το s2 με το s1.

5. Εάν s1 ≤ s2 ≤ d, τότε βγάλε στην έξοδο → s1 όμοιο με s2.

6. Πήγαινε στο βήμα 3.

Για να κατανοήσουμε καλύτερα τον παραπάνω αλγόριθμο, θα εκτελέσουμε το ακόλουθο παράδειγμα:

Έστω ο πίνακας 5.1, στον οποίο απεικονίζονται 5 χρήστες στις γραμμές του και 4 αντικείμενα στις στήλες του, με τη μορφή 1 ή 0, με τη λογική εάν ο κάθε χρήστης έχει, ή όχι προμηθευτεί το κάθε αντικείμενο αντίστοιχα.

Στοιχείο	I1	I2	I3	I4
A	1	0	0	1
B	0	0	1	0
C	0	1	0	1
D	1	0	1	1
E	0	0	1	0

Πίνακας 5.1

Η κάθε γραμμή του πίνακα 1 αποτελεί έτσι ένα δυαδικό string και επομένως αυτό που πρέπει να κάνουμε είναι να συγκρίνουμε τα ακόλουθα strings:

1001

0010

0101

1011

0010

Εάν θέσουμε και ως προϋπόθεση ομοιότητας μία distance d = 0,5 τότε εκτελώντας βηματικά τον αλγόριθμο:

Υπάρχουν περισσότερα από ένα και βάζουμε στην είσοδο τα strings 1001, 0010, 0101, 1011, 0010 και το d=0,5.

Ονομάζουμε s1=1001.

Επόμενο string υπάρχει και το ονομάζουμε s2=0010.

Συγκρίνουμε τα s1 και s2 (εάν ουσιαστικά έχουν τουλάχιστον τα μισά τους στοιχεία όμοια) και δεν ισχύει s1 ≤ s2 ≤ d (έχουν κοινό μόνο το 2ο στοιχείο), οπότε ελέγχουμε αν υπάρχει επόμενο string – υπάρχει – και το ονομάζουμε s2=0101.

Συγκρίνουμε τα s1 και s2 και δεν ισχύει s1 ≤ s2 ≤ d, οπότε ελέγχουμε αν υπάρχει επόμενο string – υπάρχει – και το ονομάζουμε s2=1011.

Συγκρίνουμε τα s1 και s2 και ισχύει s1 ≤ s2 ≤ d (έχουν τρία κοινά στοιχεία), οπότε βγάζουμε στην έξοδο το ζεύγος **(1001, 1011)**.

Επόμενο string υπάρχει και το ονομάζουμε s2=0010.

Συγκρίνουμε τα s1 και s2 και δεν ισχύει s1 ≤ s2 ≤ d, οπότε ελέγχουμε αν υπάρχει επόμενο string – δεν υπάρχει.

---------- Τέλος πρώτου κύκλου ----------

Υπάρχουν περισσότερα από ένα και βάζουμε στην είσοδο όλα τα strings εκτός από το s1, δηλαδή 0010, 0101, 1011, 0010 και το d=0,5.

Ονομάζουμε s1=0010.

Επόμενο string υπάρχει και το ονομάζουμε s2=0101.

Συγκρίνουμε τα s1 και s2 και δεν ισχύει s1 ≤ s2 ≤ d, οπότε ελέγχουμε αν υπάρχει επόμενο string – υπάρχει – και το ονομάζουμε s2=1011.

Συγκρίνουμε τα s1 και s2 και ισχύει s1 ≤ s2 ≤ d (έχουν δύο κοινά στοιχεία), οπότε βγάζουμε στην έξοδο το ζεύγος **(0010, 1011)**.

Επόμενο string υπάρχει και το ονομάζουμε s2=0010.

Συγκρίνουμε τα s1 και s2 και δεν ισχύει s1 ≤ s2 ≤ d, οπότε ελέγχουμε αν υπάρχει επόμενο string – δεν υπάρχει.

---------- Τέλος δεύτερου κύκλου ----------

Υπάρχουν περισσότερα από ένα και βάζουμε στην είσοδο όλα τα strings εκτός από το s1, δηλαδή 0101, 1011, 0010 και το d=0,5.

Ονομάζουμε s1=0101.

Επόμενο string υπάρχει και το ονομάζουμε s2=1011.

Συγκρίνουμε τα s1 και s2 και δεν ισχύει s1 ≤ s2 ≤ d, οπότε ελέγχουμε αν υπάρχει επόμενο string – υπάρχει – και το ονομάζουμε s2=0010.

Συγκρίνουμε τα s1 και s2 και δεν ισχύει s1 ≤ s2 ≤ d, οπότε ελέγχουμε αν υπάρχει επόμενο string – δεν υπάρχει.

---------- Τέλος τρίτου κύκλου ----------

Υπάρχουν περισσότερα από ένα και βάζουμε στην είσοδο όλα τα strings εκτός από το s1, δηλαδή 1011, 0010 και το d=0,5.

Ονομάζουμε s1=1011.

Επόμενο string υπάρχει και το ονομάζουμε s2=0010.

Συγκρίνουμε τα s1 και s2 και ισχύει s1 ≤ s2 ≤ d (έχουν δύο κοινά στοιχεία), οπότε βγάζουμε στην έξοδο το ζεύγος **(0010, 1011)**.

---------- Τέλος τέταρτου κύκλου ----------

Δεν υπάρχουν περισσότερα από ένα strings – τέλος εκτέλεσης.

Μετά την εκτέλεση του Naive επί του παραδείγματός μας, στην έξοδο θα έχουμε τρία ζεύγη, τα (1001, 1011), (0010, 1011), (0010, 1011), εκ των οποίων τα δύο όμοια, οπότε το τελικό μας αποτέλεσμα θα είναι τα (1001, 1011), (0010, 1011).

Παρατηρούμε ότι και στα δύο ζεύγη υπάρχει η ομοιότητα που απαιτεί η distance που εξ αρχής είχαμε θέσει (d=0,5).

Παρατηρούμε επίσης ότι κατά την εκτέλεση του αλγορίθμου πραγματοποιήθηκαν 10 συγκρίσεις επί ενός συνόλου 5 χρηστών. Αυτό πρακτικά σημαίνει ότι εάν είχαμε ένα διευρυμένο dataset της τάξης του ενός εκατομμυρίου χρηστών, θα έπρεπε να πραγματοποιηθούν 500.000 συγκρίσεις, κάτι που θα ανέβαζε κατά πολύ το κόστος εκτέλεσης, τόσο σε χρόνο όσο και σε υπολογιστική ισχύ.

3.3 Ο ΑΛΓΟΡΙΘΜΟΣ Ball-Hashing.

Με τη μελέτη του Ball-Hashing, εισερχόμαστε σε μια κατηγορία αλγορίθμων ανίχνευσης ομοιοτήτων. Πρακτικά όμως ο αλγόριθμος αυτός ανακαλύπτει όλα τα απολύτως όμοια αντικείμενα, γεγονός που δεν είναι αποδοτικό για τις ανάγκες που προκύπτουν στην ανίχνευση καταναλωτικής συμπεριφοράς, αλλά και στους λόγους που υπαγορεύουν αυτή να είναι προσεγγιστική.

Ο Ball-Hashing είναι ένας αλγόριθμος που χρησιμοποιεί ένα Reducer για κάθε ένα από τα υπό έλεγχο strings [5]. Λειτουργεί ως εξής:

Οι Mappers στέλνουν κάθε string, σε όλα τα υπόλοιπα strings για να συγκριθεί, Με καθορισμένη μία απόσταση d. Από το string, έστω ότι είναι το string s, ο Mapper δημιουργεί ζεύγη key, value, (s, -1) και – παραστατικά – μπορούμε να θεωρήσουμε ότι όλα τα ζεύγη με τα συγκρινόμενα strings (t, s), με $t \neq s$, ανήκουν σε μια σφαίρα με ακτίνα d και κέντρο s.

Χρησιμοποιούμε στον αλγόριθμο αυτό, ένα Reducer για κάθε ένα από strings μήκους b. Ωστόσο εδώ, κατακερματίζουμε σε Reducer, μόνο τα strings που έχουν απόσταση d/2, ή λιγότερο.

Στη συνέχεια ο κάθε ένας από τους n Reducers, ελέγχει εάν το string s που αντιστοιχεί στο συγκεκριμένο Reducer, είναι ένα από τα strings της εισόδου. Ο συγκεκριμένος έλεγχος είναι πολύ γρήγορος, γιατί το -1 είναι το πρώτο στοιχείο στη λίστα που σχετίζεται με τα κλειδιά s και μόνο σε περίπτωση που το s είναι μέσα στο S, θα εμφανιστεί το -1.

Όλοι οι n reducers είναι «ενεργοί» και αυτό γιατί κάθε ένας από αυτούς, δεν έχει μόνο να συγκρίνει το string το οποίο αντιστοιχεί σε αυτόν – δηλαδή το κλειδί του – με όλα τα υπόλοιπα strings που καταφθάνουν σε αυτόν, αλλά πρέπει να συγκρίνει και όλα τα strings μεταξύ τους.

Σαν αποτέλεσμα, για κάθε ζευγάρι με απόσταση d, ή μικρότερη, που παράγεται, απαιτείται να γίνει φιλτράρισμα στην έξοδο, ώστε να αποφύγουμε διπλές εγγραφές.

3.3.1 Πώς «λειτουργεί» ο Ball-Hashing.

Υποθέτοντας ότι δύο υπό σύγκριση strings s και t, απέχουν κατά μια απόσταση e ≤ d, ξεκινώντας από το s, μπορούμε να αντικαταστήσουμε τα «1» με «0», από αριστερά προς τα δεξιά, στις θέσεις όπου το s και το t διαφωνούν, για να παραγάγουμε ένα ενδιάμεσο string, u.

Ειδικότερα, εάν e < d, μας δίνεται μια ακόμα επιλογή: Μπορούμε να αλλάξουμε (d-e)/2 «1» σε «0», σε θέσεις που το s και το t έχουν «1», ξεκινώντας από τα αριστερά του string s, ώστε να παραχθεί το ενδιάμεσο string u. Οι θέσεις αυτές αλλάζουν και πάλι από «0» σε «1», κατά τη μετατροπή του u σε t.

Έτσι, για να κατασκευάσουμε το u από το s, ελέγχουμε το s από τα αριστερά προς τα δεξιά. Εάν συναντήσουμε «1» εκεί που το t έχει «0», μετατρέπουμε αυτό το «1» σε «0». Εάν συναντήσουμε «1», εκεί που και το t έχει «1», τότε μετατρέπουμε τον «1» σε «0», με την προϋπόθεση ότι δεν έχουμε υπερβεί το όριο των (d-e)/2 θέσεων. Σταματάμε τη διαδικασία όταν έχουμε αλλάξει d/2 θέσεις από «1» σε «0». Το αποτέλεσμα που λαμβάνουμε, θα είναι το string u, το οποίο θα αποτελεί και την απόσταση το πολύ d/2 μεταξύ των υπό σύγκριση strings s και t.

Παράδειγμα 5

Θα χρησιμοποιήσουμε στοιχεία από τον παρακάτω Πίνακα 5.2.

USER/PRODUCT

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
u1	1	0	1	0	1	1	1	0	1	0
u2	0	1	0	1	0	0	0	0	1	1
u3	0	1	1	0	0	1	0	0	1	0
u4	1	0	0	0	0	1	0	0	1	1
u5	1	0	0	0	1	1	0	1	1	0
u6	1	0	0	0	1	1	0	0	1	1
u7	0	1	0	1	1	1	1	0	0	0
u8	0	1	1	1	0	0	0	1	0	1

Πίνακας 3.2

Σύμφωνα με την αρχή λειτουργίας του Ball-Hashing κάθε ένα από τα 8 strings που φαίνονται στον πίνακα 1, ανατίθεται σε ένα Reducer.

Έστω ότι

s = u1 = 1 0 1 0 1 1 1 0 1 0 και

t = u4 = 1 0 0 0 0 1 0 0 1 1,

διαπιστώνουμε ότι η μεταξύ τους απόσταση Hamming, είναι e = 4 (διαφέρουν δηλαδή τα δύο strings σε 4 bits). Για να είναι αποτελεσματικό το παράδειγμα, δεχόμαστε ότι η μέγιστη απόσταση είναι d = 6. Επομένως (d – e)/2 = 1 και d/2 = 3.

Ξεκινώντας από αριστερά προς τα δεξιά στον έλεγχο των θέσεων του string s, διαπιστώνουμε ότι στην πρώτη θέση (σημειώνεται παρακάτω με κόκκινο), τόσο το s όσο και το t, έχουν «1».

s = u1 = 1 0 1 0 1 1 1 0 1 0

t = u4 = 1 0 0 0 0 1 0 0 1 1

Επομένως αλλάζουμε στο string s, το «1» σε «0» και επειδή (d – e)/2 = 1, δε μπορούμε να αλλάξουμε άλλο «1» του s όταν και το t είναι «1».

Το υπό διαμόρφωση string u, είναι μέχρι στιγμής u=0 0 1 0 1 1 1 0 1 0, σημειώνουμε με κόκκινο χρώμα, τη μέχρι στιγμής μεταβολή.

Στη συνέχεια εντοπίζουμε στην τρίτη θέση διαφορετικά στοιχεία και το «1» του s, το μετατρέπουμε σε «0».

s = u1 = 1 0 1 0 1 1 1 0 1 0

t = u4 = 1 0 0 0 0 1 0 0 1 1

Το u τροποποιείται ως εξής: u=0 0 0 0 1 1 1 0 1 0

Ομοίως πράττουμε και στη θέση πέντε, όπου υπάρχουν διαφορετικά bits στα δύο υπό σύγκριση strings και σταματάμε τη διαδικασία, αφού την έχουμε πραγματοποιήσει ήδη για d/2 = 3 θέσεις.

s = u1 = 1 0 1 0 1 1 1 0 1 0

t = u4 = 1 0 0 0 0 1 0 0 1 1

Το ενδιάμεσο string u, θα είναι τελικά: u=0 0 0 0 0 1 1 0 1 0.

3.4 Ο ΑΛΓΟΡΙΘΜΟΣ MinHashing.

Χρησιμοποιώντας τη μέθοδο κατακερματισμού, μπορούμε να αξιοποιήσουμε τον επιμερισμό χαρακτηριστικών για την ανίχνευση ομοιότητας, ειδικά όταν αυτό που μας ενδιαφέρει, είναι ένα αποτέλεσμα που δεν απαιτεί ταύτιση των υπό σύγκριση αντικειμένων, αλλά απλά, ένα βαθμό ομοιότητας και μάλιστα καθορισμένο από τον προγραμματιστή.

Διατυπώθηκε σχετικά, ο αλγόριθμος MinHashing [5] και μάλιστα με προσανατολισμό στη μετατροπή του για εκτέλεση εν παραλλήλω, καλύπτοντας τις απαιτήσεις και αποκομίζοντας τα οφέλη του Map Reduce [2].

Γενικά οι hash functions που θα χρησιμοποιήσουμε, είναι της μορφής h=(ax+b) modn, όπου το n είναι ο αριθμός των γραμμών του πίνακα που πρόκειται να κάνουμε hash, a το βήμα και b το άλμα ανά τις γραμμές.

Η διαδικασία του MinHashing, είναι:

Θεωρούμε ότι το υπό σύγκριση data set, βρίσκεται σε ένα πίνακα με διαστάσεις n x m, τον οποίο στο εξής θα αποκαλούμε πίνακα Α.

Αντικαθιστούμε τα στοιχεία του πίνακα Α (τις στήλες m), με ακέραιους αριθμούς 0, 1, 2, ..., (m-1).

Εντάσσουμε στον πίνακα Α - σε στήλες – και τα αποτελέσματα που υπολογίζουμε από κάθε hash function. Ουσιαστικά μεταθέτουμε τις γραμμές του πίνακα Α με βήμα και άλματα που καθορίζονται από το hash, όπως περιγράψαμε παραπάνω.

Από τα αποτελέσματα του υπολογισμού των hash, μας ενδιαφέρουν μόνο τα σημεία του πίνακα Α που έχουν την τιμή 1.

Δημιουργείται ο πίνακας S, ο οποίος έχει στήλες τα αντικείμενα (τις στήλες ουσιαστικά του πίνακα Α) και γραμμές τόσες, όσες είναι και οι hash functions που χρησιμοποιούνται. Ο πίνακας S ονομάζεται πίνακας υπογραφών (signature table) και θα πρέπει να «γεμίσουμε» με τις μικρότερες τιμές από αυτές που θα έχουν οι hash functions, όταν αυτές θα εφαρμόζονται σε όλα τα στοιχεία – αντικείμενα του κύριου πίνακα (πίνακας Α), που έχουν 1. Τα στοιχεία του πίνακα με 0 δε μας ενδιαφέρουν.

Ο τελικός πίνακας S που θα προκύψει, θα είναι και εκείνος του οποίου θα συγκριθούν οι στήλες μεταξύ τους. Ανάλογα με το πόσα στοιχεία βρεθούν όμοια ανά στήλη, αυτός θα είναι και ο βαθμός ομοιότητας των υπό σύγκριση στηλών.

Παρακάτω, στο κεφάλαιο 4 που αποτελεί και το ερευνητικό μέρος της εργασίας, έχουμε δοκιμάσει να αποτυπώσουμε τον αλγόριθμο με ψευδοκώδικα, εισάγοντάς τον στο Map Reduce και τροποποιώντας τον για να λειτουργήσει παράλληλα, ταυτόχρονα δηλαδή σε όλα τα σημεία που θα υπάρχουν στοιχεία του πίνακα Α.

Παραθέτουμε παρακάτω το σχετικό αλγόριθμο, με βάση τα παραπάνω.

Input data: Ο πίνακας εισόδου Α που βρίσκεται σε κάθε κόμβο και οι ακέραιοι n, m και k

for each $l_1 = 1, \ldots, k$ // Ο I_1 μετράει το πλήθος των Hash Functions

  for each $l_2 = 1, \ldots, m$ // Ο I_2 μετράει το πλήθος των στηλών (Items)

    $S(l_1, l_2) = n+1;$ // Αρχικοποίηση του πίνακα υπογραφών S με τον ακέραιο αριθμό που είναι το πλήθος των στηλών συν ένα.

  endoffor

endoffor

for $l = 0, \ldots, k$ // και για κάθε μία από τις Hash Functions

    if ($S(l, j) > h_l(i)$ && $A(i, j) == 1$) then

```
                                    // Ελέγχουμε εάν το κελί που προσδιορίζεται από την εκάστοτε Hash
                                    Function και το αντικείμενο που έχει είναι «1», είναι μικρότερο από αυτό
                                    που υπάρχει ήδη στον πίνακα υπογραφών.
        S(l,j) = h_l(i) ;           // αν είναι, τότε το αντικαθιστούμε.
      endofif
endoffor
for j1, j2 ε R
int countMatch=0;                   // Δημιουργούμε ένα μετρητή που θα μας προσδιορίσει τον αριθμό όμοιων
                                    αντικειμένων.
    for ( i = 0; i < k ; i ++)
        if(S(i, j1) == S(i,j2)) then countMatch++;
            endofif
    endoffor
  if (countMatch / k >= threshold ) then
                                    // Για κάθε όμοιο ζεύγος που ανακαλύπτουμε και βρίσκεται μέσα στο
                                    προτεινόμενο threshold, δημιουργούμε ένα ζεύγος εξόδου.
  endofif
endoffor
output ← ζεύγη από στήλες
```

3.5 Locality Sensitive Hashing (LSH).

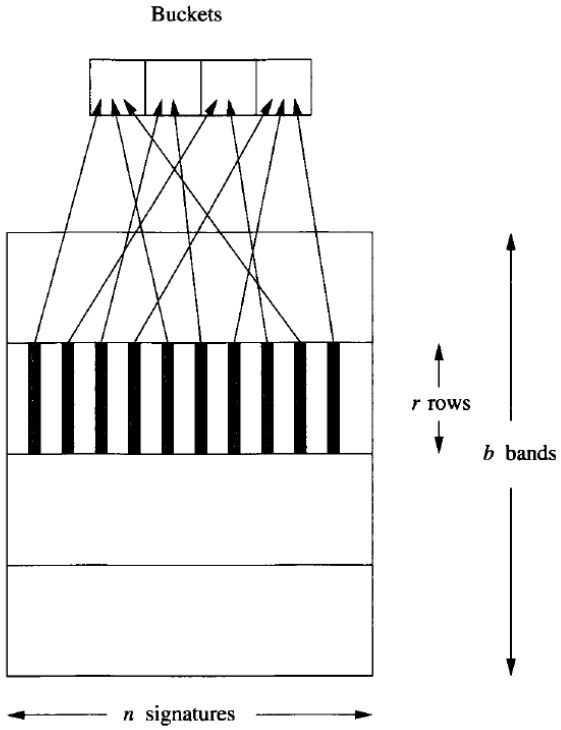
Από τη μελέτη του Min Hashing διαπιστώσαμε τον τρόπο με τον οποίο ένα μεγάλο, κατανεμημένο σε πολλούς κόμβους (μηχανές), αρχείο, μπορεί να γίνει αντικείμενο επεξεργασίας και να λάβουμε από αυτό το επιθυμητό αποτέλεσμα, με χρήση Hash Functions.

Ενδεικτικά παρατηρήσαμε ότι με τη χρήση ενός text αρχείου που απεικονίζει ένα πίνακα περίπου 88,000 X 16,000 εγγραφές, κατορθώσαμε να περιορίσουμε σε πολύ μεγάλο βαθμό τις 16,000 στήλες στον αριθμό των Hash Functions που χρησιμοποιήσαμε, δηλαδή 20 έως και 100, για να υπολογίσουμε κατά προσέγγιση του threshold που προσδιορίσαμε εξ αρχής το βαθμό ομοιότητας εγγραφών.

Εκείνο που δεν έγινε κατορθωτό να περιορίσουμε, είναι οι 88,000 γραμμές, αφού πρέπει να γίνουν συγκρίσεις ανά δύο σε 88,000 χρήστες, άρα ένας υπολογισμός της τάξης (½ X 10^8).

Επομένως μπορούμε να ισχυριστούμε ότι ναι μεν έχουμε εντοπίσει το πρόβλημα, όχι όμως και ότι το έχουμε επιλύσει σε επίπεδο ικανοποιητικό. Σε αυτό ακριβώς το στόχο προσανατολίζεται ο αλγόριθμος LSH [2, 5], που αποτελεί μια εξελιγμένη εκδοχή του Min Hashing και ουσιαστικά διαφοροποιείται στο γεγονός ότι κατακερματίζει σε μικρότερους πίνακες τον τελικό πίνακα υπογραφών που προκύπτει και τοποθετεί στο ίδιο bucket αντικείμενα που θα εντοπίσει να έχουν ίδιο κλειδί [2, 3]. Εάν λοιπόν είχαμε προς επεξεργασία – ενδεικτικά – 1 εκατομμύριο εγγραφές (όχι ιδιαίτερα μεγάλος αριθμός σε πραγματική βάση), θα έπρεπε να συγκριθούν ανά δύο, να δημιουργηθούν, πάνω από 500 δισεκατομμύρια ζεύγη εγγραφών. Υποθέτοντας όμως ότι υπάρχουν διαθέσιμα 1000 buckets και η διανομή των εγγραφών γινόταν ισοκατανεμημένα, τότε αυτόματα οι αναγκαίοι υπολογισμοί μας θα λιγόστευαν στο 1/1000. Στην πραγματικότητα βέβαια, το παραπάνω σενάριο δε θα μπορούσε να υλοποιηθεί με ακρίβεια, μπορούμε όμως να πλησιάσουμε πάρα πολύ στο ιδεατό. Πρακτικά, θεωρώντας ότι ο αριθμός των απαιτούμενων buckets καθορίζεται αυθαίρετα από το δημιουργό, μπορούμε να πούμε ότι υπάρχει η δυνατότητα να καθορίσουμε το συνολικό αριθμό των συγκρίσεων σε ό,τι βαθμό επιθυμούμε, με κάποιους όμως περιορισμούς. Εάν δηλαδή επιλέξουμε ένα πολύ μεγάλο αριθμό από buckets, είναι φυσικό να μην έχουμε διαθέσιμο μεγάλο μέρος της μνήμης του συστήματός μας. Ανεξάρτητα επίσης από τον αριθμό των buckets που επιλέγουμε να χρησιμοποιήσουμε, δε μπορούμε να αποφύγουμε την ύπαρξη ομοίων, εντελώς, εγγραφών.

Ερχόμενοι τώρα στην εφαρμογή του LSH στον πίνακα υπογραφών, σε ένα πίνακα υπογραφών με n στήλες που είναι τα αντικείμενα, επιλέγουμε b τμήματα γραμμών (bands) – οι γραμμές του πίνακα υπογραφών περιέχουν είπαμε τις τιμές των hash functions – με το κάθε ένα να περιέχει r γραμμές, από το σύνολο (b X r) γραμμών του πίνακα υπογραφών. Εάν δύο υπογραφές συμφωνούν σε όλες τις γραμμές – ουσιαστικά δηλαδή, εάν δύο στήλες είναι όμοιες μέσα στο τμήμα (band) του πίνακα, τοποθετούνται στο ίδιο bucket. Τα περιεχόμενα κάθε bucket συγκρίνονται για κάθε μία hash function. Η παραπάνω διαδικασία που περιγράψαμε, εικονίζεται στο παρακάτω σχήμα 3.1.



Σχήμα 3.1. Διασπορά των υπογραφών σε τμήματα και η τοποθέτησή τους σε buckets.

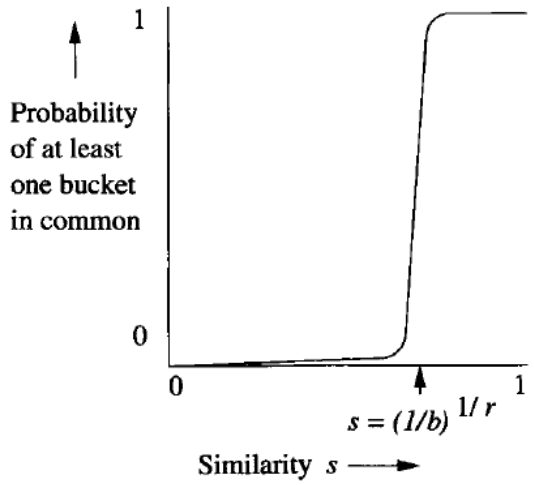
Υποθέτοντας πρακτικά ότι έχει προκύψει ένας πίνακας υπογραφών όπου για ένα στοιχείο του S1 έχουν αντίστοιχα υπολογιστεί οι hash functions h1, h2, … , hn, από ένα απόσπασμα του πίνακα αυτού, χρησιμοποιούμε – έστω – τέσσερεις hash functions και δημιουργούμε ζεύγη (κλειδιού, τιμής) της μορφής (k, v) = ((1, h1, h2, h3, h4), S1). Το «1» που βλέπουμε στο κλειδί, καταδεικνύει το απόσπασμα του πίνακα υπογραφών στο οποίο έχει εντοπιστεί το συγκεκριμένο ζεύγος. Εάν στη συνέχεια με τη σύγκριση βρεθεί παρόμοιο κλειδί (k) σε δεύτερο στοιχείο, τότε αυτά τα δύο παρόμοια στοιχεία κατατάσσονται σε κοινό bucket, με σκοπό να γίνουν αντικείμενο σύγκρισης στο επόμενο στάδιο. Την ίδια διαδικασία πραγματοποιούμε σε όλα τα αντίστοιχα bands του πίνακα υπογραφών. Αυτόματα οι υπολογισμοί περιορίζονται αισθητά, με ταυτόχρονο αποτέλεσμα την εύρεση των παρόμοιων στοιχείων σε πολύ μικρότερο χρόνο.

Εάν για λόγους απλότητας θα υποθέσουμε ότι έχουμε στη διάθεσή μας όσα buckets επιθυμούμε, έτσι ώστε να μην προκύπτουν συμπτώσεις και οι υπογραφές να καταχωρίζονται στο ίδιο bucket μόνο και μόνο τότε, όταν έχουν την ίδια τιμή σε ολόκληρο το band που εξετάζεται. Τότε, αν η πιθανότητα να συμφωνούν οι υπογραφές σε μία γραμμή είναι s, η αντίστοιχη πιθανότητα να συμφωνούν σε όλες τις γραμμές του band, θα είναι sr, η αντίθετη πιθανότητα, να μη συμφωνούν δηλαδή σε όλες τις γραμμές, είναι 1 - sr και η πιθανότητα σε κανένα από τα να bands b να μη

συμφωνούν οι υπογραφές σε όλες τις γραμμές, είναι (1 - sr)b. Τέλος η πιθανότητα οι υπογραφές να συμφωνούν σε όλες τις γραμμές σε τουλάχιστον ένα band, είναι 1 - (1 - sr)b.

Δίνοντας ενδεικτικές τιμές στους παραπάνω υπολογισμούς, αν r = 5 και b = 20, αυτό σημαίνει ότι έχουμε ένα πίνακα υπογραφών 100 γραμμών διαιρεμένο σε 20 bands των 5 γραμμών το κάθε ένα. Ο τύπος της πιθανότητας που υπάρχει να συγκριθούν μεταξύ τους δύο υπογραφές με ομοιότητα s, θα είναι: 1 - (1 – s5)20. Υποθέτοντας ότι η Jaccard ομοιότητα (την έχουμε χαρακτηρίσει και ως threshold) είναι s = 0,8, ή 80%, s5 = 0.328, η πιθανότητα να συμφωνούν δύο υπογραφές, είναι περίπου μία στις τρεις (1/3), αρκετά μικρή. Αν και έχουμε 20 «ευκαιρίες» να πετύχουμε, το (1 – s5)20 = 0,00035, είναι πάρα πολύ μικρό. Η περίπτωση όμως να βρούμε το ζευγάρι αυτό των υπογραφών σε ένα bucket, είναι πολύ μεγάλες, 1 - 0,00035 = 0,99965.

Αν στο ίδιο παράδειγμα απλά διαφοροποιούσαμε την τιμή της Jaccard ομοιότητας και ορίζαμε ότι s = 0.4, τότε 1 -(1 – 0,45)20 = 1-(1 – 0,01024)20 = 1-(0,98976)20 = 1–0,81395 = 0,19, δηλαδή πιθανότητα κάτω από 20%. Καταλήγουμε επομένως σε ένα συμπέρασμα για το παράδειγμά μας, ότι τα r = 5 και b = 20 είναι κατάλληλα επιλεγμένα για υψηλή Jaccard ομοιότητα και αντίστοιχα δεν είναι κατάλληλο να επιλεγούν για χαμηλό ποσοστό Jaccard ομοιότητας. Η καμπύλη της πιθανότητας 1 - (1 - sr)b απεικονίζεται στο ακόλουθο σχήμα 2 και επιβεβαιώνει το παραπάνω συμπέρασμα.



Σχήμα 3.2. Η πιθανότητα ένα ζεύγος υπογραφών να εμφανιστεί μαζί σε ένα τουλάχιστον bucket.

3.6 ΣΥΜΠΕΡΑΣΜΑΤΑ ΓΙΑ ΤΟΥΣ ΑΛΓΟΡΙΘΜΟΥΣ.

Πραγματοποιήθηκε η ανάλυση και η μελέτη ομάδων αλγορίθμων με κάποια συγκεκριμένα χαρακτηριστικά. Είδαμε ότι για κάθε μία ομάδα από αυτές, υπάρχει και ένα συγκεκριμένο πεδίο εφαρμογής, άμεσα εξαρτώμενο από τον τρόπο με τον οποίο προσεγγίζεται το πρόβλημα από τον κάθε αλγόριθμο.

Συγκεκριμενοποιώντας την αναζήτησή μας, ως προ το ποιος από τους αλγορίθμους αυτούς ταιριάζει στα χαρακτηριστικά που έχει η αναζήτηση όμοιας καταναλωτικής συμπεριφοράς, καταλήγουμε στη χρήση του MinHashing ως προτύπου και αυτό γιατί μας δίνεται η δυνατότητα να λάβουμε αποτελέσματα, όχι «ακριβώς όμοια» σε όλο το εύρος του υπό εξέταση αρχείου δεδομένων, αλλά «περίπου όμοια» που είναι και το ζητούμενο. Θέλουμε να εντοπίζουμε κάθε φορά χρηστές ομοιάζοντες και όχι ίδιους, ώστε για παράδειγμα, να μπορούν να τους γίνουν προτάσεις προϊόντων που δεν έχουν αγοράσει. Ο MinHashing, χρησιμοποιώντας τις Hash Functions, μας εξασφαλίζει ότι θα βρεθούν παρόμοιοι καταναλωτές, μπορεί όχι όλοι, αλλά υπάρχει ανεκτό περιθώριο απώλειας και το τελικό αποτέλεσμα θα είναι κοντά στην πραγματικότητα, σε χρόνο ο οποίος θα είναι πολύ πιο σύντομος από οποιαδήποτε άλλη διαδικασία και με τη δυνατότητα παραμετροποίησης της αναζήτησης, τόσο ως προς το βαθμό, όσο και ως προς «βάθος» ομοιότητας. Τα τελευταία αυτά χαρακτηριστικά αποτελούν το προϊόν της υλοποίησης του MinHashingMR, όπως θα δούμε και παρακάτω.

ΚΕΦΑΛΑΙΟ 4
ΥΛΟΠΟΙΗΣΗ

4.1 ΑΠΟ ΤΟΝ Min-Hashing ΣΤΟΝ MinHashingMR.

Εκείνο που θα μας απασχολήσει στη συνέχεια, είναι η δημιουργία ενός αλγορίθμου, ο οποίος θα μπορεί να εφαρμοστεί στο Map Reduce και ουσιαστικά να παραγάγει όμοια ζεύγη που θα αποτελούν και το προϊόν της αναζήτησής μας. Τα ζεύγη θα είναι όμοια κατά προσέγγιση, την οποία εμείς κάθε φορά θα καθορίζουμε (threshold). Ο αλγόριθμος στον οποίο βασιστήκαμε, είναι όπως αναφέραμε ο Min Hashing και από αυτόν θα προκύψει ο **MinHashingMR**, όπως ονομάσαμε το νέο αλγόριθμο. Παρακάτω αναλύεται το σύνολο της διαδικασίας.

Στην είσοδο του συστήματός μας θα έχουμε μέρος του συνολικού πίνακα με τα στοιχεία προς ανάλυση (dataset). Οι γραμμές του πίνακα αυτού αποτελούνται από τους χρήστες οι οποίοι έχουν προμηθευτεί τουλάχιστον ένα αντικείμενο από αυτά που αποτελούν τις στήλες του πίνακα αυτού. Για κάθε αντικείμενο που προμηθεύεται ένας χρήστης, Περιέχεται το «1» στο αντίστοιχο κελί , ενώ σε όλα τα υπόλοιπα κελιά περιέχεται το «0». Κάθε γραμμή αρχίζει με ένα

αριθμό – τον αύξοντα αριθμό καταναλωτή / χρήστη. Ο συνολικός πίνακας θα είναι της μορφής του πίνακα 3.3 που εικονίζεται παρακάτω.

USER/PRODUCT

	p1	p2	p3	p4	p5	p6	p7	p8	…	pm
0	1	0	1	0	1	1	1	0	…	0
1	0	1	0	1	0	0	0	0	…	1
2	0	1	1	0	0	1	0	0	…	0
3	1	0	0	0	0	1	0	0	…	1
4	1	0	0	0	1	1	0	1	…	0
5	1	0	0	0	1	1	0	0	…	1
…	…	…	…	…	…	…	…	…	…	…
n	0	1	1	1	0	0	0	1	…	1

Πίνακας 3.3

Ο πίνακας 3.3 βρίσκεται κατακερματισμένος σε - περισσότερους του ενός – σταθμούς εργασίας / κόμβους. Η διαδικασία που θα ακολουθηθεί, έχει τη φιλοσοφία να αντιμετωπιστεί το περιεχόμενο κάθε κόμβου ξεχωριστά ως προς τη δημιουργία ενός πίνακα υπογραφών. Κατόπιν όλοι οι πίνακες υπογραφών που θα προκύψουν στους κόμβους, θα συγκριθούν και θα συμπτυχθούν σε έναν, με περιεχόμενό του τον τελικό πίνακα υπογραφών, ο οποίος θα έχει τόσες στήλες όσες και τα αντικείμενα, όσες δηλαδή και οι στήλες του αρχικού πίνακα (πίνακας 3.3) και τόσες γραμμές όσες και οι hash functions που χρησιμοποιήθηκαν. Τα στοιχεία του τελικού πίνακα υπογραφών, συγκρινόμενα ανά στήλες και με βάση το threshold που θα έχει τεθεί και θα είναι η Jaccard ομοιότητα, θα δώσουν και την ομοιότητα που αναζητούμε.

Παρακάτω θα περιγράψουμε τον αλγόριθμο που θα χρησιμοποιήσουμε για το MinHashing και ταυτόχρονα θα αποτυπώνουμε κάθε του βήμα σε ένα παράδειγμα, εκτελώντας ουσιαστικά τον ίδιο αλγόριθμο. Έστω ότι για το παράδειγμά μας θα χρησιμοποιηθεί ο παρακάτω πίνακας 3.4, ως ο πίνακας εισόδου.

Στοιχείο	**I1**	**I2**	**I3**	**I4**
a	1	0	0	1
b	0	0	1	0
c	0	1	0	1
d	1	0	1	1
e	0	0	1	0

Πίνακας 3.4

Έστω επίσης ότι βρίσκεται κατακερματισμένος σε τέσσερεις σταθμούς εργασίας, τους 0, 1, 2 και 3, με κατάτμηση η οποία υποδηλώνεται με διαφορετικό χρώμα για κάθε σταθμό (πίνακας 3.5.α)

και στη συνέχεια διαχωρίζεται για κάθε σταθμό και αυτό απεικονίζεται στους πίνακες 3.5.α, 3.5.β, 3.5.γ και 3.5.δ, για να οπτικοποιηθεί καλύτερα.

Στοιχείο	I1	I2	I3	I4
0	1	0	0	1
1	0	0	1	0
2	0	1	0	1
3	1	0	1	1
4	0	0	1	0

Πίνακας 3.5.α.

Παρατηρούμε ότι για λόγους διαχείρισης του πίνακα εισόδου, αντικαθιστούμε τη στήλη που αναγράφονται τα στοιχεία, με αύξουσα αρίθμηση, ξεκινώντας από το 0. Αυτό αποτελεί και το πρώτο βήμα που θα πρέπει να εκτελέσει ο αλγόριθμος. Άρα λοιπόν:

1. Αντικατέστησε τα στοιχεία του πίνακα εισόδου με αύξουσα αρίθμηση.

Παρακάτω απεικονίζουμε και τους επιμέρους πίνακες όπως είπαμε παραπάνω και είναι:

Στοιχείο	I1	I2
0	1	0
1	0	

Πίνακας 3.5.β.

Στοιχείο	I1	I2	I3	I4
0			0	1
1		0	1	0
2		1		
3		0		1
4	0			

Πίνακας 3.5.γ.

Στοιχείο	I1	I3	I4
2	0	0	1
3	1	1	

Πίνακας 3.5.δ.

Στοιχείο	I2	I3	I4
4	0	1	0

Πίνακας 3.5.ε.

Στη συνέχεια συγκεντρώνουμε όλες τις hash functions, οι οποίες θα είναι του τύπου

h = (ax+b) modn,

όπου a θα είναι το βήμα που θα χρησιμοποιηθεί, b το άλμα και n ο αριθμός των γραμμών του πίνακα εισόδου και για το παράδειγμά μας θα είναι οι εξής δύο:

h_1= (x+1) mod5 και

h_2= (3x+1) mod5.

Άρα προκύπτει το βήμα 2 του αλγορίθμου που είναι:

2. Όρισε τις hash functions που θα χρησιμοποιηθούν.

Και αμέσως μετά στο τρίτο βήμα, υπολογίζουμε την τιμή των hash functions αυτών για κάθε γραμμή, χρησιμοποιώντας ως x τον αύξοντα αριθμό των στοιχείων. Οπότε:

3. Υπολόγισε την τιμή της κάθε hash function για κάθε γραμμή του πίνακα εισόδου.

Εάν στον πίνακα 3.5.α συμπληρώσουμε τόσες στήλες όσες και οι hash functions και υπολογίσουμε και τις αντίστοιχες τιμές, θα έχουμε τον παρακάτω πίνακα 3.6.

Στοιχείο	I1	I2	I3	I4	(x+1)mod5	(3x+1)mod5
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

Πίνακας 3.6. Ο μετασχηματισμένος πίνακας εισόδου, συμπληρωμένος από τις δύο hash functions.

Παρατηρούμε στον πίνακα 3.4 ότι ουσιαστικά με τη χρήση των hash functions προκαλούμε τις απαραίτητες μετακινήσεις των γραμμών του πίνακα για να ακολουθήσουμε τη Min Hashing διαδικασία.

Κατόπιν φθάνουμε στο τέταρτο βήμα του αλγορίθμου, στο οποίο ξεκινάει η διαδικασία Map.

4.1.1 Η Map διαδικασία.

4. Είσοδος Map. Έχουμε ως είσοδο ένα κομμάτι του πίνακα εισόδου (έναν από τους παραπάνω υποπίνακες κάθε φορά) για κάθε σταθμό.

(για το παράδειγμα τους αριθμούς n=5 (ο αριθμός των γραμμών του), m=4 (ο αριθμός των στηλών του) και λ=2 (ο αριθμός των hash functions που θα χρησιμοποιηθούν)).

Αντίστοιχα προσδιορίζουμε και την έξοδο της Map διαδικασίας που θα είναι ζεύγη της μορφής (k_1, k_2), (S (*,k_1), S(*,k_2)).

5. Αρχικοποίησε τον πίνακα S γεμίζοντας τα κελιά του με τον αριθμό n+1.

Ο πίνακας υπογραφών S, για κάθε ένα σταθμό ξεχωριστά, μετά την παραπάνω διαδικασία, θα είναι αντίστοιχα αρχικοποιημένος (έχοντας τη μέγιστη τιμή n+1=6 στα κελιά τους) :

Node 1 (κίτρινο χρώμα):

Στοιχείο	I1	I2	I3	I4
h1	6	6	6	6
h2	6	6	6	6

Node 2 (πράσινο χρώμα):

Στοιχείο	I1	I2	I3	I4
h1	6	6	6	6
h2	6	6	6	6

Node 3 (πορτοκαλί χρώμα):

Στοιχείο	I1	I2	I3	I4
h1	6	6	6	6
h2	6	6	6	6

Node 4 (μπλε χρώμα):

Στοιχείο	I1	I2	I3	I4
h1	6	6	6	6
h2	6	6	6	6

Στον πίνακα υπογραφών για κάθε σταθμό θα καταχωρίσουμε την πιο μικρή τιμή που λαμβάνει κάθε hash function, όταν στην αντίστοιχη γραμμή υπάρχει «1». Διαφορετικά αγνοούμαι την τιμή της.

Επομένως τα επόμενα βήματα του αλγορίθμου θα είναι:

6. Για τη γραμμή του πίνακα εισόδου, εάν στη γραμμή υπάρχει «1» πάρε την τιμή της hash function και σύγκρινέ την με αυτή του κελιού που ορίζεται από τη γραμμή της hash function και τη στήλη με το αντίστοιχο αντικείμενο.

7.α. Εάν είναι μικρότερη η νέα τιμή αντικατέστησε με αυτήν το περιεχόμενο του εν λόγω κελιού.

7.β. Εάν δεν είναι μικρότερη η νέα τιμή προχώρησε στο επόμενο βήμα.

8. Εάν υπάρχει άλλη hash function πήγαινε στο βήμα 6 για την επόμενη hash function.

9. Εάν δεν υπάρχει άλλη hash function, αποτύπωσε τα ζεύγη που έχουν δημιουργηθεί για να σχηματιστεί η έξοδος της διαδικασίας Map.

Επομένως για το παράδειγμά μας θα είναι:

Για τον πίνακα S του κόμβου 0:

Ο πίνακας Α του σταθμού 0, έχοντας συμπεριλάβει και τις hash functions σε αυτόν, είναι ο παρακάτω:

Στοιχείο	I1	I2	(x+1)mod5	(3x+1)mod5
0	1	0	1	1
1	0		2	4

Εξετάζουμε τα κελιά όπου υπάρχει 1 και τοποθετούμε στον πίνακα S του σταθμού 0, τη μικρότερη τιμή για κάθε hash function που συναντάμε όπου η τιμή κελιού είναι 1. Ο S δηλαδή, στο σταθμό 0 θα είναι:

Στοιχείο	I1	I2	I3	I4
h1	1	0	6	6
h2	0	6	6	6

Για τον πίνακα S του κόμβου 1:

Ο πίνακας A του node 1, έχοντας συμπεριλάβει και τις hash functions σε αυτόν, είναι ο παρακάτω:

Στοιχείο	I1	I2	I3	I4	(x+1)mod5	(3x+1)mod5
0			0	1	1	1
1		0	1	0	2	4
2		1			3	2
3		0		1	4	0
4	0				0	3

Ενεργώντας αντίστοιχα με την παραπάνω περίπτωση, έχουμε:

Στοιχείο	I1	I2	I3	I4
h1	6	3	2	1
h2	6	2	4	0

Για τον πίνακα S του κόμβου 2:

Ο πίνακας A του node 2, έχοντας συμπεριλάβει και τις hash functions σε αυτόν, είναι ο παρακάτω:

Στοιχείο	I1	I3	I4	(x+1)mod5	(3x+1)mod5
2	0	0	1	3	2
3	1	1		4	0

Ο αντίστοιχος πίνακας S είναι:

Στοιχείο	I1	I2	I3	I4
h1	4	6	4	3
h2	0	6	0	2

Για τον πίνακα S του κόμβου 3:

Στοιχείο	I2	I3	I4	(x+1)mod5	(3x+1)mod5
4	0	1	0	0	3

και ο S:

Στοιχείο	I1	I2	I3	I4
h1	6	6	0	6
h2	6	6	3	6

Για το σχηματισμό των ζευγών που θα έχουμε στην έξοδο της Map διαδικασίας, αντιστοιχίζοντας τις μεταβλητές k_1, λ και k_2 και τα ζεύγη που ακολουθούν το μοντέλο $((k_1, k_2), (S(\lambda,k_1), S(\lambda,k_2)))$ για το παράδειγμά μας, η k_1 είναι ο αριθμός του κόμβου, η k_2 είναι ο αριθμός της στήλης (δηλαδή του αντικειμένου) και η λ ο αριθμός της γραμμής (δηλαδή η hash function).

Συνολικά τα ζεύγη που θα λάβουμε από την έξοδο της Map διαδικασίας για σύγκριση, εικονίζονται στον ακόλουθο πίνακα 3.7.

Οι γραμμές που έχουν σημειωθεί με γκρίζο χρώμα έχουν επαναληφθεί και πιο πάνω στον πίνακα και μπορούν να παραλειφθούν. Χρησιμοποιούμε μόνο τα κελιά στα οποία έχουμε τιμή 1.

k1 (αριθμός κόμβου)	K2 (αριθμός στήλης)	λ (αριθμός γραμμής)	Ζεύγη εξόδου Map
0	1	0	((0,1),(S(0,0),S(0,1)))
0	1	1	((0,1),(S(1,0),S(1,1)))
0	2	0	((0,2),(S(0,0),S(0,2)))
0	2	1	((0,2),(S(1,0),S(1,2)))
0	3	0	((0,3),(S(0,0),S(0,3)))
0	3	1	((0,3),(S(1,0),S(1,3)))
0	4	0	((0,4),(S(0,0),S(0,4)))
0	4	1	((0,4),(S(1,0),S(1,4)))
1	1	0	((1,1),(S(0,1),S(0,1)))
1	1	1	((1,1),(S(1,1),S(1,1)))
1	2	0	((1,2),(S(0,1),S(0,2)))
1	2	1	((1,2),(S(1,1),S(1,2)))
1	3	0	((1,3),(S(0,1),S(0,3)))
1	3	1	((1,3),(S(1,1),S(1,3)))
1	4	0	((1,4),(S(0,1),S(0,4)))
1	4	1	((1,4),(S(1,1),S(1,4)))
2	1	0	((2,1),(S(0,2),S(0,1)))
2	1	1	((2,1),(S(1,2),S(1,1)))
2	2	0	((2,2),(S(0,2),S(0,2)))
2	2	1	((2,2),(S(1,2),S(1,2)))
2	3	0	((2,3),(S(0,2),S(0,3)))
2	3	1	((2,3),(S(1,2),S(1,3)))
2	4	0	((2,4),(S(0,2),S(0,4)))
2	4	1	((2,4),(S(1,2),S(1,4)))
3	1	0	((3,1),(S(0,3),S(0,1)))
3	1	1	((3,1),(S(1,3),S(1,1)))
3	2	0	((3,2),(S(0,3),S(0,2)))
3	2	1	((3,2),(S(1,3),S(1,2)))
3	3	0	((3,3),(S(0,3),S(0,3)))
3	3	1	((3,3),(S(1,3),S(1,3)))
3	4	0	((3,4),(S(0,3),S(0,4)))
3	4	1	((3,4),(S(1,3),S(1,4)))

Πίνακας 3.7. Η έξοδος που προκύπτει έπειτα από τη Map phase.

Πριν περάσουμε στην αποτύπωση του αλγορίθμου για τη διαδικασία Reduce, ας συνοψίσουμε τα βήματα εκτέλεσης του μέχρι τώρα:

1. Αντικατέστησε τα στοιχεία του πίνακα εισόδου με αύξουσα αρίθμηση.

2. Όρισε τις hash functions που θα χρησιμοποιηθούν.

3. Υπολόγισε την τιμή της κάθε hash function για κάθε γραμμή του πίνακα εισόδου.

4. Είσοδος Map. Έχουμε ως είσοδο ένα κομμάτι του πίνακα εισόδου (έναν από τους παραπάνω υποπίνακες κάθε φορά) για κάθε σταθμό.

5. Αρχικοποίησε τον πίνακα S γεμίζοντας τα κελιά του με τον αριθμό n+1.

6. Για τη γραμμή του πίνακα εισόδου, εάν στη γραμμή υπάρχει «1» πάρε την τιμή της hash function και σύγκρινέ την με αυτή του κελιού που ορίζεται από τη γραμμή της hash function και τη στήλη με το αντίστοιχο αντικείμενο.

7.α. Εάν είναι μικρότερη η νέα τιμή αντικατέστησε με αυτήν το περιεχόμενο του εν λόγω κελιού.

7.β. Εάν δεν είναι μικρότερη η νέα τιμή προχώρησε στο επόμενο βήμα.

8. Εάν υπάρχει άλλη hash function πήγαινε στο βήμα 6 για την επόμενη hash function.

9. Εάν δεν υπάρχει άλλη hash function, αποτύπωσε τα ζεύγη που έχουν δημιουργηθεί για να σχηματιστεί η έξοδος της διαδικασίας Map.

4.1.2 Η Reduce διαδικασία.

Στη συνέχεια **περνάμε στη φάση Reduce.**

Η είσοδος αποτελείται από την έξοδο της Map διαδικασίας, δηλαδή από ζεύγη της μορφής ((k1, k2), (Se (*,k1, Se(*,k2))).

Η έξοδος θα μας δώσει αντίστοιχα όμοια ζεύγη και θα είναι και το ζητούμενο από την όλη διαδικασία.

Το επόμενο βήμα που πρέπει να γίνει, είναι να συγκεντρώσουμε τους επιμέρους πίνακες S που βρίσκονται διασπαρμένοι ανά υπολογιστή, σε ένα πίνακα υπογραφών, ο οποίος θα έχει σε κάθε του κελί τη μικρότερη τιμή του αντίστοιχου κελιού (όπου αυτό υπάρχει) όλων των επιμέρους πινάκων υπογραφών.

Αυτό πρακτικά αποτυπώνεται ως εξής:

11. Σύγκρινε όλα τα όμοια (σε συντεταγμένες κελιά) που προκύπτουν από όλους τους σταθμούς και τοποθέτησε τη μικρότερη τιμή στο αντίστοιχο κελί του τελικού πίνακα υπογραφών.

Ο τελικός, συνολικός, πίνακας υπογραφών S, για το παράδειγμα που ακολουθούμε, θα είναι ο ακόλουθος πίνακας 3.6.

Στοιχείο	**I1**	**I2**	**I3**	**I4**
h1	1	3	0	1
h2	0	2	0	0

Πίνακας 3.8. Ο συγκεντρωμένος πίνακας υπογραφών S.

Στο τελευταίο κομμάτι της Reduce διαδικασίας, εξετάζουμε τις στήλες του S ανά δύο, για ομοιότητες. Δηλαδή στο παράδειγμα, θα εξετάσουμε για όμοια στοιχεία τη στήλη I_1 με τις I_2, I_3 και I_4, τη στήλη I_2 με τη I_3 και τη I_4 και τη I_3 με τη I_4. Αυτό θα γίνει έχοντας θέσει ως threshold, την ομοιότητα Jaccard για παράδειγμα και συγκρίνουμε εάν ο αριθμός ομοιότητας είναι εντός των τεθειμένων ορίων.

Θα είναι δηλαδή:

12. Σύγκρινε ένα προς ένα τα στοιχεία των στηλών του τελικού πίνακα υπογραφών.

13. Εάν τα εκάστοτε δύο υπό σύγκριση στοιχεία έχουν κοινά - τουλάχιστον ίσα με την Jaccard ομοιότητα – κελιά, τότε να εμφανίζονται στην έξοδο ως επιθυμητό αποτέλεσμα.

Εάν δίναμε ένα ποσοστό ομοιότητας 50%, δηλαδή 0,5, τότε βλέπουμε από τον πίνακα 3.6 ότι οι στήλες I_1, I_3 και I_4, πληρούν το κριτήριο ομοιότητας και μπορούν να συμπεριληφθούν στο αποτέλεσμα εξόδου. Πιο αναλυτικά, το R=((1, 0), (3, 2), (0, 0), (1, 0)) και το threshold 0,5 για τις δύο hash functions, τα ζεύγη εξόδου μας θα είναι:

(2, (1, 0))

(2, (0, 0))

(2, (1, 0))

Συνοψίζοντας τον αλγόριθμό που σχηματίσαμε, θα έχουμε:

1. Αντικατέστησε τα στοιχεία του πίνακα εισόδου με αύξουσα αρίθμηση.

2. Όρισε τις hash functions που θα χρησιμοποιηθούν.

3. Υπολόγισε την τιμή της κάθε hash function για κάθε γραμμή του πίνακα εισόδου.

4. Είσοδος Map. Έχουμε ως είσοδο ένα κομμάτι του πίνακα εισόδου (έναν από τους παραπάνω υποπίνακες κάθε φορά) για κάθε σταθμό.

5. Αρχικοποίησε τον πίνακα S γεμίζοντας τα κελιά του με τον αριθμό n+1.

6. Για τη γραμμή του πίνακα εισόδου, εάν στη γραμμή υπάρχει «1» πάρε την τιμή της hash function και σύγκρινέ την με αυτή του κελιού που ορίζεται από τη γραμμή της hash function και τη στήλη με το αντίστοιχο αντικείμενο.

7.α. Εάν είναι μικρότερη η νέα τιμή αντικατέστησε με αυτήν το περιεχόμενο του εν λόγω κελιού.

7.β. Εάν δεν είναι μικρότερη η νέα τιμή προχώρησε στο επόμενο βήμα.

8. Εάν υπάρχει άλλη hash function πήγαινε στο βήμα 6 για την επόμενη hash function.

9. Εάν δεν υπάρχει άλλη hash function, αποτύπωσε τα ζεύγη που έχουν δημιουργηθεί για να σχηματιστεί η έξοδος της διαδικασίας Map.

10. Έξοδος Map. $((k_1, k_2), (S(\lambda,k_1), S(\lambda,k_2)))$, όπου k_1 είναι ο αριθμός του κόμβου, η k_2 είναι ο αριθμός της στήλης (δηλαδή του αντικειμένου) και η λ ο αριθμός της γραμμής (δηλαδή η hash function).

11. Σύγκρινε όλα τα όμοια (σε συντεταγμένες κελιά) που προκύπτουν από όλους τους σταθμούς και τοποθέτησε τη μικρότερη τιμή στο αντίστοιχο κελί του τελικού πίνακα υπογραφών.

12. Σύγκρινε ένα προς ένα τα στοιχεία των στηλών του τελικού πίνακα υπογραφών.

13. Εάν τα εκάστοτε δύο υπό σύγκριση στοιχεία έχουν κοινά - τουλάχιστον ίσα με την Jaccard ομοιότητα – κελιά, τότε να εμφανίζονται στην έξοδο ως επιθυμητό αποτέλεσμα.

Η παραπάνω περιγραφή του αλγορίθμου, μπορεί να περιγραφεί και με ψευδοκώδικα, εξελίσσοντας τον αλγόριθμο MinHashing που παρουσιάσαμε στο Κεφάλαιο 3, στη διαδικασία του Map Reduce, όπως φαίνεται και στην παρακάτω έκφραση:

Διαδικασία Map

Input data: Ο πίνακας εισόδου που βρίσκεται σε κάθε κόμβο και οι ακέραιοι n, m και k.

for each $l_1 = 1, \ldots, k$ // Ο l_1 μετράει το πλήθος των Hash Functions

for each $l_2 = 1, \ldots, m$ // Ο l_2 μετράει το πλήθος των στηλών (Items)

$S(l_1, l_2) = n+1;$ // Αρχικοποίηση του πίνακα υπογραφών S με τον ακέραιο αριθμό που είναι το πλήθος των στηλών συν ένα.

endoffor

endoffor

for each A(i,j) // Για κάθε ένα από τους πίνακες εισόδου των κόμβων

for $l = 0, \ldots, k$ // και για κάθε μία από τις Hash Functions

if ($S(l, j) > h_l(i)$ && $A(i, j) == 1$) then

// Ελέγχουμε εάν το κελί που προσδιορίζεται από την εκάστοτε Hash Function και το αντικείμενο που έχει είναι «1», είναι μικρότερο από αυτό που υπάρχει ήδη στον πίνακα υπογραφών.

$S(l,j) = h_l(i)$; // αν είναι, τότε το αντικαθιστούμε.

endofif

endoffor

for each $k_1=0,\ldots,(m - 1)$ // Για να δημιουργήσουμε τα απαραίτητα ζεύγη εξόδου της Map διαδικασίας, για κάθε μία Hash Function, βρίσκουμε το περιεχόμενο των κελιών από κάθε κόμβο που προσδιορίζονται από τη Hash Function και το αντικείμενο.

for each $k_2=(k_1+1),\ldots,m$

for each λ=0,…,k-1

functions k_1, k_2

Map output <-- ((k_1, k_2), (S(λ,k_1), S(λ,k_2)))

endoffor

endoffor

endoffor

endoffor

Διαδικασία Reduce

Χρησιμοποιούμε στην είσοδο, την έξοδο της Map διαδικασίας από κάθε κόμβο e.

for each i= 1, …, k

for each j = 1, …, m

S(i, j) = min Se(i,j); // Αναζητούμε από όλους τους πίνακες υπογραφών που έχουν δημιουργηθεί στους κόμβους, το μικρότερο στοιχείο για κάθε κελί του συνολικού πίνακα υπογραφών και το τοποθετούμε σε αυτόν.

endoffor

endoffor

for j_1, j_2 ε R

int countMatch=0; // Δημιουργούμε ένα μετρητή που θα μας προσδιορίσει τον αριθμό όμοιων αντικειμένων.

endoffor

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

if(S(i, j_1) == S(i,j_2)) then countMatch++;

if (countMatch / k >= threshold) then // Για κάθε όμοι ο ζεύγος που ανακαλύπτουμε και βρίσκεται μέσα στο προτεινόμενο threshold, δημιουργούμε ένα ζεύγος εξόδου.

output ← (k, (l_{j1}, l_{j2}));

endofif

endofif

endoffor

Output: Όμοια ζεύγη που ακολουθούν τον περιορισμό του τεθειμένου threshold.

Ολόκληρη η παραπάνω διαδικασία που αναλύθηκε και μελετήθηκε, μέχρι και την αποτύπωσή της σε αλγόριθμο, αποτελεί την έκφραση του **MinHashingMR**, του αλγορίθμου, που βασιζόμενος στη φιλοσοφία ανάπτυξης του MinHashing, δημιουργήθηκε για να χρησιμοποιηθεί στο Map Reduce για εύρεση παρόμοιων αντικειμένων.

4.1.3 «Σειριακή εκτέλεση» του Min-Hashing.

Ας δούμε όμως πώς θα εκτελούταν ο συγκεκριμένος αλγόριθμος, εάν επρόκειτο για ένα υπολογιστή, εάν δηλαδή ολόκληρο το dataset βρισκόταν σε ένα υπολογιστή.

Μια σειριακή εκδοχή του αλγορίθμου παρατίθεται σε ψευδοκώδικα, εδώ, κρατώντας ότι:

k, είναι ο αριθμός των hash functions.

m, ο αριθμός των στηλών και

n, ο αριθμός των γραμμών.

1. Δημιούργησε τον πίνακα υπογραφών S

2. Τοποθέτησε σε όλα τα κελιά του πίνακα υπογραφών το n+1 (ο αριθμός 6, για το παράδειγμά μας).

3. Υπολόγισε το στοιχείο του πίνακα υπογραφών που προσδιορίζεται από τη hash function και το αντικείμενο.

4. Σύγκρινέ το με το αντίστοιχο που υπάρχει στον πίνακα υπογραφών.

5. Εάν είναι μικρότερο, αντικατάστησέ το και προχώρα στο βήμα 7.

6. Εάν όχι, προχώρα στο βήμα 7.

7. Προχώρησε στο επόμενο αντικείμενο και αν υπάρχει πήγαινε στο βήμα 3.

8. Εάν δεν υπάρχει επόμενο αντικείμενο πήγαινε στην επόμενη hash function και το πρώτο αντικείμενο και εκτέλεσε από το βήμα 3.

9. Εάν δεν υπάρχει επόμενη hash function δώσε τον τελικό πίνακα υπογραφών.

10. Σύγκρινε τις στήλες του πίνακα υπογραφών.

11. Εάν τα εκάστοτε δύο υπό σύγκριση στοιχεία έχουν κοινά - τουλάχιστον ίσα με την Jaccard ομοιότητα – κελιά, τότε να εμφανίζονται στην έξοδο ως επιθυμητό αποτέλεσμα.

Ο πίνακας εισόδου στη μορφή όπου τα στοιχεία του έχουν αντικατασταθεί από αύξουσα αρίθμηση, ξεκινώντας από το 0 και έχουν προστεθεί δύο ακόμα στήλες, μία για κάθε hash function, εικονίζεται παρακάτω (πίνακας 3.9).

Στοιχείο	I1	I2	I3	I4	(x+1)mod5	(3x+1)mod5
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

Πίνακας 3.9.

Θα δημιουργήσουμε τον πίνακα υπογραφών S, τον οποίο και πάλι θα «γεμίσουμε» με το n+1 (το 6 για το παράδειγμα) και τον οποίο παραθέτουμε (πίνακας 3.10).

Στοιχείο	I1	I2	I3	I4
h1	6	6	6	6

h2	6	6	6	6

Πίνακας 3.10. Ο πίνακας υπογραφών S.

Στη συνέχεια τοποθετούμε στον S το μικρότερο στοιχείο που ανακαλύπτουμε στο εκάστοτε κελί και τελικά λαμβάνουμε (και πάλι) τον παρακάτω πίνακα 3.11.

Στοιχείο	**I1**	**I2**	**I3**	**I4**
h1	1	3	0	1
h2	0	2	0	0

Πίνακας 3.11. Ο τελικός πίνακας υπογραφών S.

4.1.4 Σύγκριση παράλληλης και σειριακής εκτέλεσης.

Το παράδειγμα που χρησιμοποιήσαμε παραπάνω, δεν είναι ιδιαίτερα αντιπροσωπευτικό για εξαγωγή συμπερασμάτων προτίμησης ως προς τη διαδικασία εκτέλεσης. Χρησιμοποιήθηκε περισσότερο για να εξηγηθεί η βηματική εκτέλεση του αλγορίθμου, τόσο κατά τη χρήση του σε περισσότερους από έναν υπολογιστές, όσο και σε ένα υπολογιστή. Στην πραγματικότητα, έχουμε να κάνουμε με ένα κατά πολύ μεγαλύτερο όγκο δεδομένων. Επομένως θα πρέπει να έχουμε υπόψη μας ότι το συνολικό data set :

Μπορεί να μη «χωράει», αποθηκευτικά, σε ένα υπολογιστή,

Μπορεί να μη «χωράει», επεξεργαστικά, σε ένα υπολογιστή,

Μπορεί να μην είναι δυνατό να συγκεντρωθεί σε ένα υπολογιστή, λόγω του ότι μπορεί να μην είναι συγκεντρωμένο σε μία βάση,

Μπορεί να μην είναι δυνατό να συγκεντρωθεί σε ένα υπολογιστή λόγω έλλειψης πόρων του συστήματος και

Μπορεί να μην είναι δυνατό να συγκεντρωθεί σε ένα υπολογιστή, λόγω έλλειψης πόρων του δικτύου.

Όλα τα παραπάνω μας οδηγούν και επιβεβαιώνουν την ανάγκη εξεύρεσης και χρήσης μια παράλληλης λύσης. Και για το λόγο αυτό είναι οικονομικότερη η παράλληλη εκτέλεση του MinHashing.

4.2 Distributed Cache.

Πριν προχωρήσουμε στην παράθεση της διαδικασίας με την οποία «τρέξαμε» τον κώδικα, θα αναφέρουμε κάποια στοιχεία, καθώς και τον τρόπο λειτουργίας και την αιτία χρήσης της distributed cache [10].

Η distributed cache επιτρέπει το διαμοιρασμό στατικών δεδομένων, ανάμεσα σε όλους τους σταθμούς εργασίας. Με την έννοια «στατικά δεδομένα» εννοούμε δεδομένα τα οποία για λόγους προγραμματιστικής ευχέρειας επιθυμούμε να γνωρίζουν όλοι οι σταθμοί που συμμετέχουν στη Map Reduce διαδικασία. Για να επωφεληθούμε από αυτή τη λειτουργία, θα πρέπει τα σημεία στα οποία βρίσκονται δεδομένα (οι σταθμοί εργασίας), να έχουν γνωστοποιηθεί στο σύστημα, πριν ξεκινήσει η Map Reduce διαδικασία.

Τα αρχεία που μπορούν να γίνουν cached με αυτό τον τρόπο, είναι αρχεία που χρειάζονται στην εκτέλεση του Map Reduce και χρειάζονται από τις εφαρμογές και είναι της μορφής text, archives, jars κλπ.

Οι εφαρμογές καθορίζουν τα αρχεία που χρειάζονται μέσω urls (hdfs:// ή http://) να γίνουν cached μέσω της εντολής JobConf.

Η JobConf είναι η κύρια εντολή, με την οποία ένας προγραμματιστής μπορεί να περιγράψει μια εργασία Map Reduce στο Hadoop προς εκτέλεση.

Η DistributedCache υποθέτει ότι τα αρχεία που ορίζονται μέσω ενός url που «δείχνει» στον hdfs:// υπάρχουν ήδη στο file system, στο συγκεκριμένο path που αναγράφεται στο url.

Το σύστημα αντιγράφει όλα τα προσδιορισμένα αρχεία σε όλους τους υποκείμενους σταθμούς εργασίας, πριν ακόμα ξεκινήσει να εκτελείται οποιαδήποτε εργασία της διαδικασίας σε αυτούς.

Η DistributedCache ελέγχει την τροποποίηση των αρχείων που έχουν γίνει cached. Εννοείται φυσικά ότι τα αρχεία που έχουν γίνει cached, δε γίνεται να τροποποιηθούν με κανένα τρόπο, ούτε από την εφαρμογή, ούτε και εξωτερικά καθόσον η εργασία εκτελείται.

Η DistributedCache μπορεί να χρησιμοποιηθεί για τη διανομή απλών text αρχείων read – only, ή και πιο σύνθετων τύπων όπως archives και jars. Τα archives (zip, tar, tgz και tar.gz files) είναι *un-archived* στους υποκείμενους σταθμούς εργασίας. Στα αρχεία όμως έχει δοθεί άδεια να εκτελούνται.

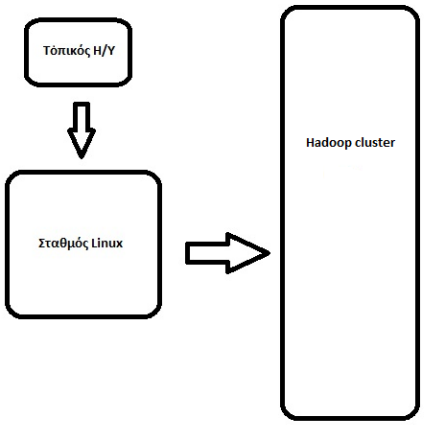
Τα αρχεία που γίνονται DistributedCache, μπορούν να είναι private, ή public, κάτι που καθορίζει με ποιο τρόπο μπορούν αυτά να διαμοιραστούν στους υποκείμενους σταθμούς εργασίας.

- Τα Private DistributedCached αρχεία, γίνονται cached στο τοπικό directory, χωρίς να έχει πρόσβαση σε αυτά ο χρήστης του οποίου οι εργασίες χρειάζονται αυτά τα αρχεία. Τα αρχεία αυτά μοιράζονται σε όλες τις εργασίες που εκτελούνται, μόνο από το συγκεκριμένο χρήστη, ενώ δε μπορεί να υπάρχει πρόσβαση σε αυτά από άλλες εργασίες άλλων χρηστών. Το αρχείο μετατρέπεται σε private μέσα από τη διαδικασία που ένα αρχείο μπορεί να γίνει read only από το file system, το HDFS συνήθως. Ένα αρχείο είναι read only, όταν δεν έχει άδεια να έχει πρόσβαση σε αυτό ο καθένας, ή όταν το directory στο οποίο ανήκει δε μπορεί να ανοιχτεί από τον οποιοδήποτε.

- Τα Public DistributedCached αρχεία, γίνονται cached σε global directory και η πρόσβαση σε αυτά μπορεί να γίνει από όλους τους χρήστες. Προφανώς λοιπόν, μπορούν να διαμοιραστούν σε όλες τις εργασίες, τις διαδικασίες και τους χρήστες στους σταθμούς εργασίας. Όμοια με προηγουμένως, τα αρχεία μπορούν να γίνουν global, μέσα από το HDFS. Για να είναι όμως ένα αρχείο public, πρέπει τόσο τα αντίστοιχα permission s στο αρχείο, όσο και το path που οδηγεί στο συγκεκριμένο αρχείο να έχουν permissions για πλήρη πρόσβαση.

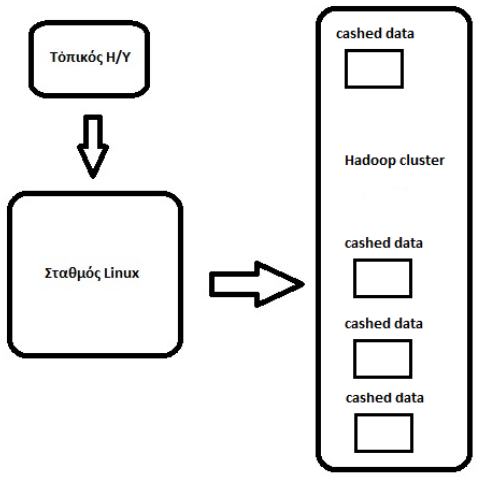
4.2.1 Διανομή αρχείων για τη διενέργεια των πειραμάτων.

Εφαρμόζοντας το παραπάνω θεωρητικό μέρος στις διαδικασίες του πειράματος, αποτυπώνουμε παρακάτω, στα σχήματα 4.1α 4.1.β και 4.1.γ, τη λειτουργία της Distributed Cache.

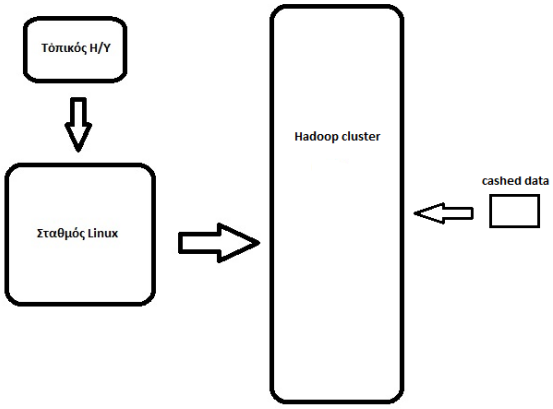


Σχήμα 4.1.α. Διασύνδεση τοπικού υπολογιστή με το σύστημα.

Θεωρώντας από το σχήμα 4.1.α, ότι ο υπολογιστής που εργαζόμαστε είναι ο «τοπικός Η/Υ», συνδέεται με ένα σταθμό Linux, ο οποίος επικοινωνεί με το Hadoop cluster. Με τη χρήση της Distributed Cache, μπορούμε να έχουμε τις παρακάτω δύο διανομές, σχήματα 4.1.β και 4.1.γ, ανάλογα με τον τρόπο που επιλέγουμε να τη χρησιμοποιήσουμε. Μπορούμε δηλαδή να διανείμουμε τα κοινώς χρησιμοποιούμενα δεδομένα είτε σε κάθε ένα από όλους τους clusters (σχήμα 4.1.β), ή κεντρικά στο Hadoop το οποίο θέτει στη διάθεση όλων, αυτά τα κοινώς χρησιμοποιούμενα δεδομένα (σχήμα 4.1.γ).



Σχήμα 4.1.β. Cache distributed σε όλους τους clusters.



Σχήμα 4.1.γ. Cache distributed στο Hadoop.

4.3 ΕΚΤΕΛΕΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ.

4.3.1 Γενικές ρυθμίσεις – εγκαταστάσεις.

Για να εκτελέσουμε το πρόγραμμα που κατασκευάσαμε, θα χρησιμοποιήσουμε περιβάλλον Linux, εγκατεστημένο σε εικονική μηχανή, συγκεκριμένα στο VMPlayer.

Στο VMPlayer, εγκαταστήσαμε λειτουργικό Ubuntu, και σε αυτό εγκαταστήσαμε το Eclipse, με τη βοήθεια του οποίου κατασκευάστηκε και έγινε compile ο κώδικας. Στο περιβάλλον του Ubuntu εγκαταστάθηκε περιβάλλον Java, έκδοση 1.6 και το Hadoop, στο οποίο έγιναν τα πειράματα. Τη συνολική εγκατάσταση της πλατφόρμας που αναφέραμε παραπάνω, περιγράφουμε αναλυτικά στο Παράρτημα Α του παρόντος.

4.3.2 Προετοιμασία εκτέλεσης του προγράμματος.

Για την εκτέλεση του προγράμματος απαιτείται προετοιμασία την οποία θα περιγράψουμε αναλυτικά εδώ.

Το κυρίως πρόγραμμα είναι το minhashinghadoopV2 (δες Παράρτημα Β). Η υλοποίησή του βασίζεται στις Hash Functions. Οι παράμετροι για κάθε Hash Function, είναι αποθηκευμένες στο αρχείο parameters.txt. Το dataset που θα χρειαστούμε, είναι ένα text αρχείο, στη γραμμογράφηση του οποίου, κάθε γραμμή περιέχει μία σειρά τιμών (values) και αντιπροσωπεύει μία στήλη του πίνακα δεδομένων. Κάθε τιμή, είναι ένας αριθμός που καταγράφει τις γραμμές του πίνακα δεδομένων. Για παράδειγμα, ο ακέραιος 12 που περιέχεται στη 15η γραμμή του αρχείου, υποδηλώνει ότι το κελί [15, 12] του πίνακα δεδομένων εισόδου περιέχει «1». Το «1» αυτό, υποδηλώνει αντίστοιχα ότι το προϊόν της στήλης 15 του πίνακα δεδομένων, το έχει προμηθευτεί ο χρήστης της γραμμής 12. Οι τιμές σε κάθε γραμμή, χωρίζονται από ένα <κενό> (space). Η πρώτη τιμή σε κάθε γραμμή, καταδεικνύει τον αριθμό της γραμμής (ουσιαστικά της στήλης του πίνακα δεδομένων), δηλαδή, η πρώτη τιμή της γραμμής 15, θα είναι το 15.

Η επιδίωξή μας είναι να εντοπίσουμε τους καταναλωτές εκείνους (δηλαδή τις στήλες του πίνακα δεδομένων – άρα τις γραμμές του data file), οι οποίοι έχουν προμηθευτεί ένα αριθμό (ο αριθμός αυτός καθορίζεται από το threshold που επιθυμούμε) κοινών προϊόντων.

Είναι σημαντικό να επισημάνουμε, ότι αφού κάθε στήλη του πίνακα υπογραφών κατασκευάζεται στο Mapper, δε χρειάζεται να γίνει σύγκριση για τη δημιουργία του τελικού πίνακα υπογραφών στο Reducer.

Επίσης, οι τιμές των Hash Functions θα πρέπει να είναι γνωστές σε όλους τους κόμβους που θα τρέξουν τη Map διαδικασία. Για το λόγο αυτό θα χρησιμοποιήσουμε τη Distributed Cache, που αναφέραμε στην παράγραφο 4.1, για να εισαγάγουμε τοπικά, σε όλους τους κόμβους τις απαραίτητες τιμές, ώστε να αποφύγουμε τη δικτυακή αναζήτηση.

Θεωρώντας ότι δουλεύουμε σε περιβάλλον Ubuntu, δημιουργούμε ένα directory στο file system, στο οποίο τοποθετούμε το data file. Θέτουμε στη συνέχεια το συγκεκριμένο path, να είναι το τοπικό path για αυτό το directory.

Στη συνέχεια ονομάζουμε το αρχείο που περιέχει το data file, datafile.txt. Οι τιμές σε κάθε γραμμή του data file διαχωρίζονται από ένα <κενό>.

Στη συνέχεια δημιουργούμε ένα text αρχείο, το οποίο ονομάζουμε parameters.txt και το τοποθετούμε στο ίδιο directory με το datafile.txt.

Αποθηκεύουμε τα δύο jar files (MinHashingHashFunctionGenV2.jar και MinHashingHadoopV2.jar), στο path (το εκάστοτε local path που έχουμε ορίσει): /home/stelmoul/Downloads/exp, το οποίο βρίσκεται στο local σύστημα.

Στο ίδιο path έχουμε αποθηκεύσει το dataset που έχουμε σε μορφή text και με όνομα datafile.txt.

Στο ίδιο επίσης path δημιουργούμε ένα κενό text αρχείο με όνομα parameters.txt, στο οποίο θα αποθηκευτεί το αποτέλεσμα της εκτέλεσης του MinHashingHashFunctionGenV2.jar και θα αποτελεί το – εκάστοτε – αρχείο παραμέτρων, του οποίου η δημιουργία και η χρήση αναλύεται ακριβώς παρακάτω.

4.3.3 Δημιουργία αρχείου παραμέτρων.

Για να δημιουργήσουμε το αρχείο που θα περιέχει τις παραμέτρους που επιθυμούμε να δώσουμε για την εκτέλεση του προγράμματος και είναι ο αριθμός των Hash Functions και το επιθυμητό threshold, ακολουθούμε τα παρακάτω βήματα.

1. Εκτελούμε το MinHashingHashFunctionGenV2.jar. Η εκτέλεσή του έχει σα σκοπό να δημιουργήσει ένα αρχείο, το parameters.txt, το οποίο θα εμπεριέχει τις ακόλουθες παραμέτρους – όπως φαίνονται και στον κώδικα και είναι:

- Γραμμή 1: ο αριθμός των γραμμών του πίνακα εισόδου.
- Γραμμή 2: ο αριθμός των στηλών του πίνακα εισόδου.
- Γραμμή 3: ο αριθμός των Hash Functions που θα χρησιμοποιηθούν.
- Γραμμή 4: η τιμή του threshold που θα χρησιμοποιηθεί (εισάγουμε ένα θετικό ακέραιο και εξυπακούεται ότι είναι ποσοστό, π.χ. εάν δώσουμε 30, θα σημαίνει ότι το επιθυμητό threshold θα είναι 30%).
- Γραμμή 5: η φράση «Parameters of Hash Functions».

- Γραμμή 6 και μετά: Ανά γραμμή αναφέρεται και μία τιμή που προκύπτει από Hash Function, που όπως έχουμε αναφέρει και νωρίτερα, θα είναι της μορφής h(x)=(ax+b)modN, όπου N είναι ο αριθμός των γραμμών του πίνακα εισόδου, το a είναι της μορφής a=2k-1, παίρνει τιμές από το σύνολο ακεραίων [0 – N] και δε διαιρείται ακριβώς με το N και από το ίδιο αυτό σύνολο παίρνει τιμές και το b. Το format που έχουν οι γραμμές από την έκτη και μετά, είναι της μορφής i; a; b, όπου i είναι ο αριθμός της Hash Function.

Στην εκτέλεση αυτού του αρχείου δίνουμε ως arguments τον αριθμό των Hash Functions που επιθυμούμε να έχουμε, έστω 50 και το ποσοστό του threshold, έστω 20 (%).

2. Δημιουργούμε στο HDFS του Hadoop δύο folders το ένα με όνομα input και το άλλο με όνομα metadata. Οι εντολές εκτελούνται όπως φαίνεται παρακάτω:

- Hadoop fs –mkdir ./input και αντίστοιχα

- Hadoop fs –mkdir ./metadata

3. Στη συνέχεια αντιγράφουμε το αρχείο datafile.txt στο directory input και στο metadata το αρχείο parameters.txt. Οι εντολές κατ' αντιστοιχία, είναι οι ακόλουθες:

- hadoop fs -copyFromLocal /home/stelmoul/Downloads/datafile.txt /user/stelmoul/input/datafile.txt

και

- hadoop fs -copyFromLocal /home/stelmoul/Downloads/parameters.txt /user/stelmoul/metadata/parameters.txt

Να σημειώσουμε εδώ, ότι κατά τη διάρκεια διεξαγωγής του πειράματος **τροποποιήθηκε** αρκετές φορές το path εκτέλεσης. Ως εκ τούτου αναφέρεται εδώ ενδεικτικά ένα path λειτουργίας.

4.3.4 Εκτέλεση του κυρίως προγράμματος.

Θεωρώντας ως local_path που ζητάει το πρόγραμμα, το path /home/stelmoul/Downloads/exp, όπως ορίσαμε και παραπάνω, τρέχουμε το πρόγραμμα MinHashingHadoopV2.jar, δίνοντας προσοχή τόσο στο path που είναι αποθηκευμένο το εκτελέσιμο (.jar) αρχείο, όσο και το local path στο οποίο έχουμε προ – αποθηκεύσει τα δεδομένα εισόδου, δηλαδή τα αρχεία datafile.txt και parameters.txt.

Για την παρακολούθηση της διαδικασίας χρησιμοποιούμε τη διεύθυνση localhost:50030/, στην οποία ξεκινάει ο λεγόμενος jobtracker, με την εκκίνηση του run, όπου φαίνεται τόσο η πρόοδος της συνολικής διαδικασίας, όσο και η πρόοδος των επιμέρους εργασιών (jobs), και για τη Map και για τη Reduce διαδικασίες. Στην έξοδό μας λαμβάνουμε το αρχείο time.txt, στο οποίο έχει καταγραφεί ο χρόνος που διήρκεσε η εκτέλεση της συνολικής διαδικασίας. Στο αρχείο αυτό μπορούμε να έχουμε ανά γραμμή όλα τα στοιχεία ανά εκτέλεση που έχουμε πραγματοποιήσει στο πρόγραμμα.

4.3.5 Περιπτώσεις εκτέλεσης του προγράμματος.

Προκειμένου να πάρουμε τιμές, εξετάζοντας σε ένα μεγάλο εύρος περιπτώσεων τα αποτελέσματα που θα μας δώσει η εκτέλεση του προγράμματος, θα εξετάσουμε τις ακόλουθες περιπτώσεις:

α. Θα χρησιμοποιήσουμε datafiles που λάβαμε από το http://fimi.ua.ac.be/data/ . Τα datafiles αυτά θα έχουν 100 και 88000 εγγραφές αντίστοιχα. Θέλουμε με αυτό τον έλεγχο να διαπιστώσουμε τη συμπεριφορά του συστήματος, τόσο στη χρήση πόρων, όσο – κυρίως – χρονικά στην αύξηση του μεγέθους των εγγραφών.

β. Θα χρησιμοποιήσουμε διαφορετικό αριθμό Mappers και Reducers στο σύστημα, με σκοπό να παρατηρήσουμε τη χρονική διαφοροποίηση της εκτέλεσης του προγράμματος, αλλά και να επιβεβαιώσουμε το θεωρητικό κομμάτι της μελέτης.

Για τις παραπάνω περιπτώσεις θα δοκιμάζουμε διαφορετικά Thresholds, όσο και διαφορετικό αριθμό Hash Functions, για να ελέγξουμε την αλλαγή, εάν αυτή υπάρχει.

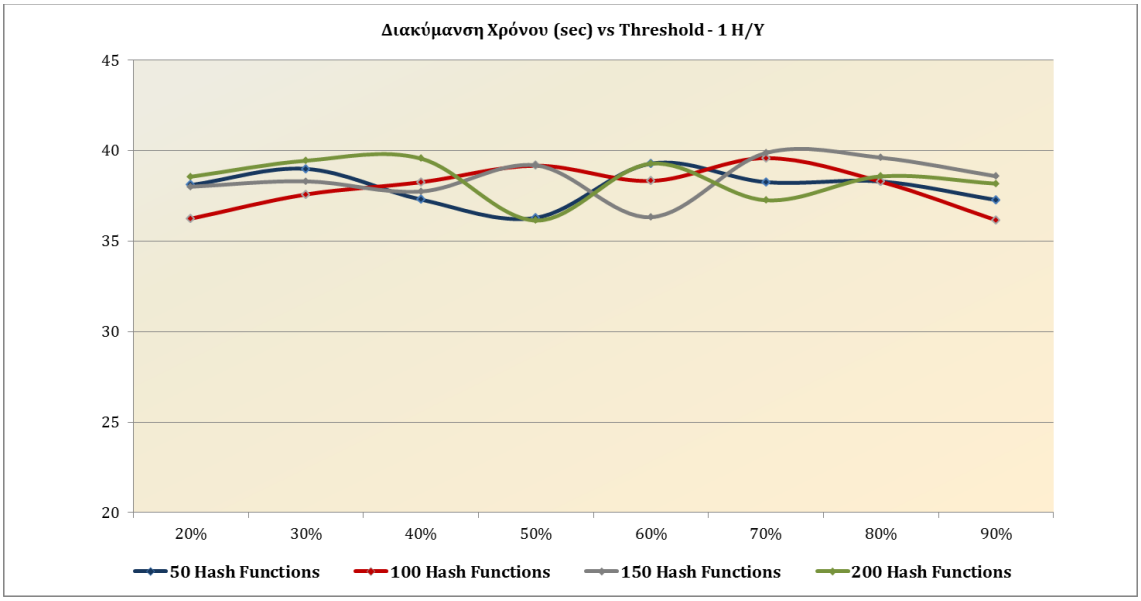
ΚΕΦΑΛΑΙΟ 5
ΑΠΟΤΕΛΕΣΜΑΤΑ

5.1 ΤΟ ΠΕΙΡΑΜΑ ΣΕ Η/Υ.

Το πρώτο πείραμα πραγματοποιήθηκε με τη χρήση ενός απλού οικιακού ηλεκτρονικού υπολογιστή. Στον υπολογιστή αυτό εγκαταστάθηκε περιβάλλον Linux, προκειμένου να είναι πιο εύκολη και λειτουργική η διαχείριση του Hadoop. Σε πρώτη φάση χρησιμοποιήθηκε ένα datafile με μέγεθος 100 εγγραφών. Στη συνέχεια έγινε απόπειρα χρήσης datafile 88000 εγγραφών, που όμως δεν ήταν επιτυχής, λόγω έλλειψης υπολογιστικών πόρων. Αρκεί να αναλογιστούμε, ότι μια και πραγματοποιείται σύγκριση ζευγών, στο αρχείο των 100 εγγραφών, πραγματοποιούνται συγκρίσεις ανά δύο σε 100 εγγραφές, ενώ στο αρχείο των 88.000 εγγραφών, σε 88.000 αντίστοιχα, κάτι που αυξάνει την ανάγκη για υπολογιστική ισχύ γεωμετρικά.

Από την εκτέλεση του προγράμματος δημιουργήθηκε ένα αρχείο (με όνομα time.txt), στο οποίο κάθε φορά καταγραφόταν, σε νέα εγγραφή, ο χρόνος που απαιτούταν για την ολοκλήρωση της διαδικασίας. Το πείραμα έγινε για το ίδιο datafile (100 εγγραφές), με χρήση διαφορετικών

παραμέτρων threshold και αριθμού hash functions (ποικίλα αρχεία parameters.txt). Το εύρος του threshold που δοκιμάστηκε ήταν από 20% - 90% και σε αυτό εφαρμόστηκαν 50, 100, 150 και 200 hash functions. Ο πίνακας που προέκυψε από τη διαδικασία αυτή, παρατίθεται στο Παράρτημα 3 (πίνακας Π3.1). Από τον πίνακα προέκυψε το γράφημα του σχήματος 5.1, στο οποίο απεικονίζεται η διακύμανση του χρόνου με τη χρήση threshold για κάθε set από hash functions που χρησιμοποιούμε.



Σχήμα 5.1. Η διακύμανση του χρόνου ολοκλήρωσης σε οικιακό Η/Υ, για αρχείο 100 εγγραφών.

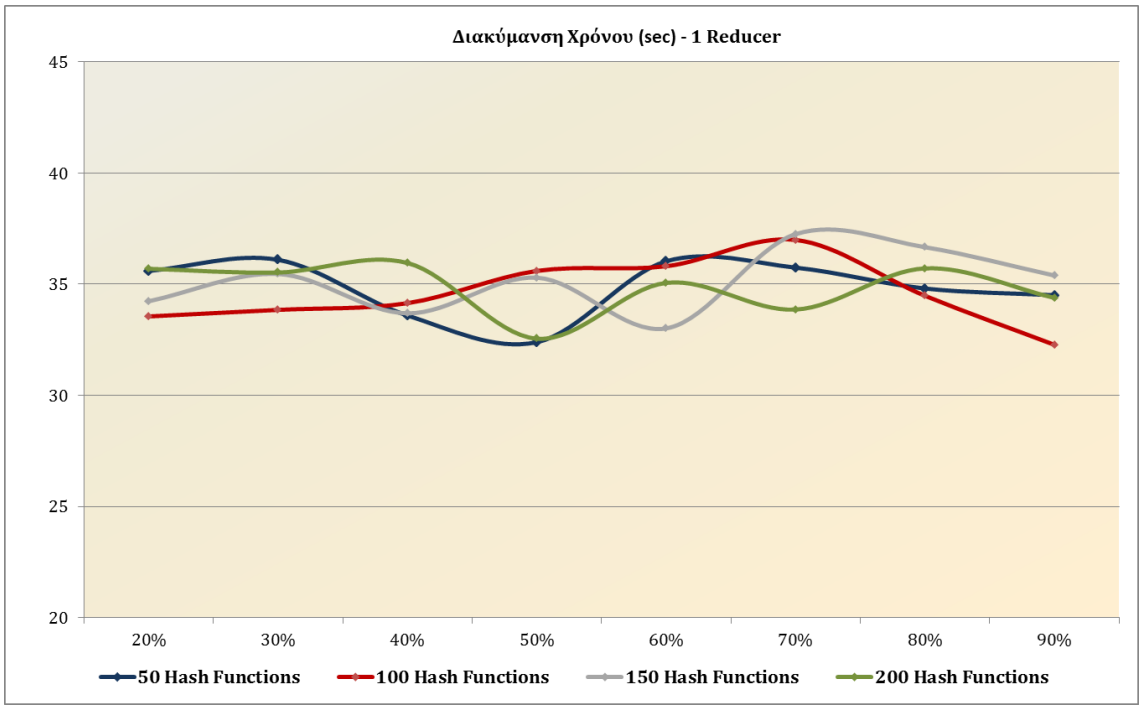
5.2 ΤΟ ΠΕΙΡΑΜΑ ΣΕ ΣΥΣΤΟΙΧΙΑ 4 ΚΟΜΒΩΝ.

Στη συνέχεια, το ίδιο πρόγραμμα εκτελέστηκε σε σύστημα πολλών κόμβων, συγκεκριμένα τεσσάρων, στο Μετσόβειο Πολυτεχνείο. Εκεί υπήρχε το περιθώριο να γίνουν πειραματισμοί, τόσο με μεγαλύτερο datafile (88000 εγγραφών), όσο και με χρήση διαφορετικού αριθμού Mappers και Reducers, προκειμένου να παρατηρηθεί και η αντίστοιχη διαφοροποίηση, εάν υπήρχε, στα αποτελέσματα.

5.2.1 Χρήση datafiles 100 και 88000 εγγραφών σε ένα κόμβο (με ένα Reducer).

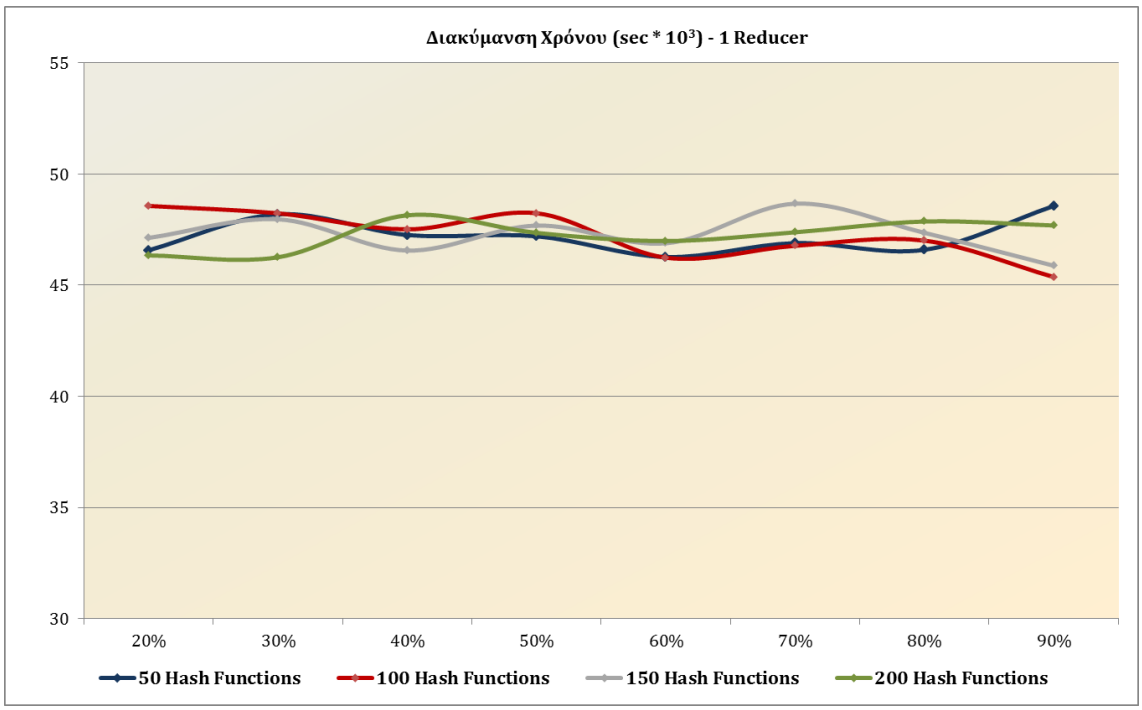
Αντίστοιχα με την εκτέλεση σε υπολογιστή, από την εκτέλεση του προγράμματος δημιουργήθηκε ένα αρχείο (με όνομα time.txt), στο οποίο κάθε φορά καταγραφόταν ο χρόνος

που απαιτούταν για την ολοκλήρωση της διαδικασίας. Το πείραμα έγινε με χρήση διαφορετικών παραμέτρων threshold και αριθμού hash functions (ποικίλα αρχεία parameters.txt). Το εύρος του threshold που δοκιμάστηκε ήταν και πάλι από 20% - 90% και σε αυτό εφαρμόστηκαν επίσης 50, 100, 150 και 200 hash functions. Ο πίνακας που προέκυψε από τη διαδικασία αυτή, παρατίθεται στο Παράρτημα 3 (πίνακας Π3.2). Από το πείραμα, για το αρχείο 100 εγγραφών, προέκυψε το γράφημα του σχήματος 5.2, στο οποίο απεικονίζεται η διακύμανση του χρόνου με τη χρήση threshold για κάθε set από hash functions που χρησιμοποιούμε.



Σχήμα 5.2. Η διακύμανση του χρόνου ολοκλήρωσης με χρήση ενός Reducer για αρχείο 100 εγγραφών.

Στη συνέχεια δοκιμάστηκε με τις ίδιες συνθήκες που αναφέρονται παραπάνω το datafile των 88000 εγγραφών. Το γράφημα που λάβαμε απεικονίζεται στη συνέχεια (Σχήμα 5.3) και ο πίνακας από τον οποίο προέκυψε, είναι αντίστοιχα ο Π3.3.



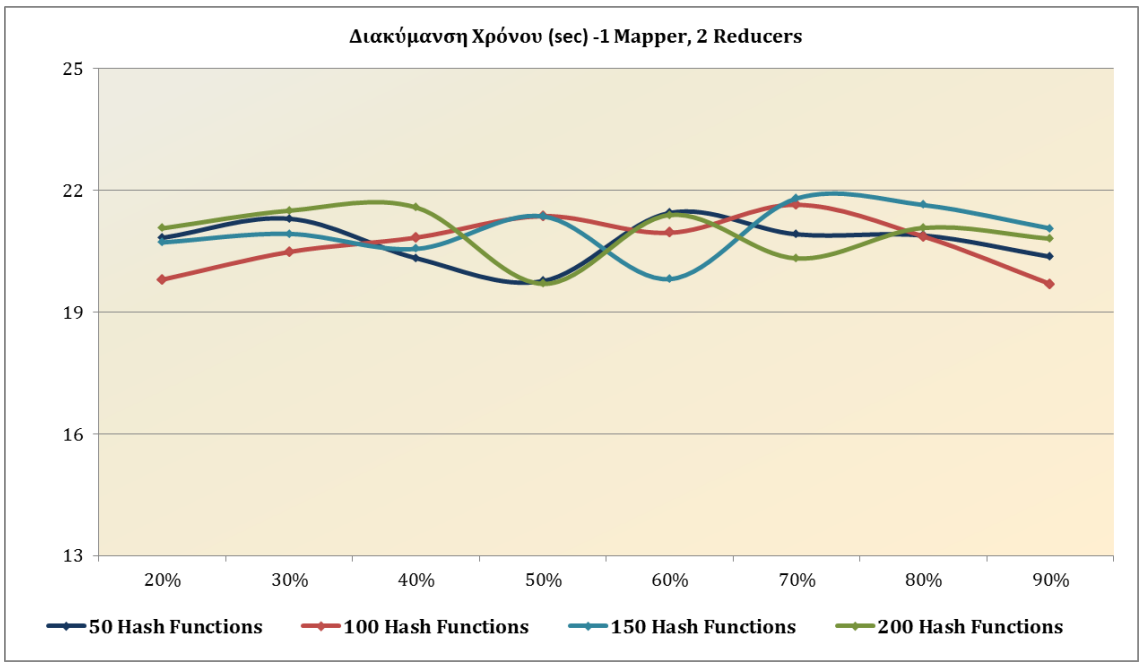
Σχήμα 5.3. Η διακύμανση του χρόνου ολοκλήρωσης με χρήση ενός Reducer για αρχείο 88.000 εγγραφών.

5.2.2. Χρήση datafiles 100 και 88000 εγγραφών με δύο Reducers.

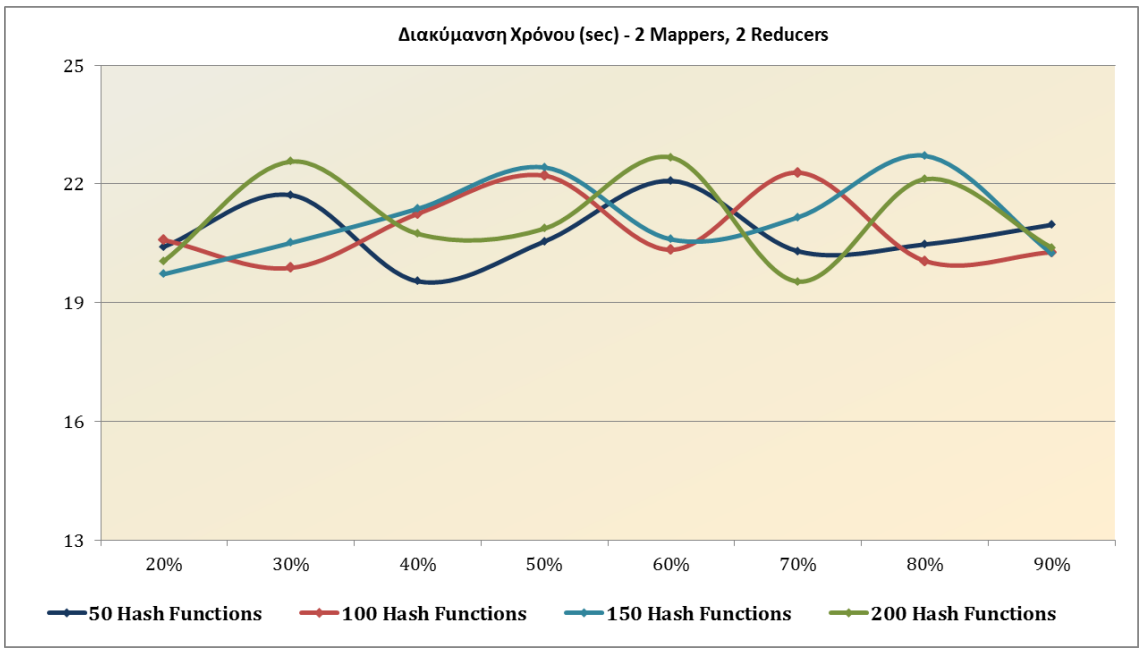
Ενεργώντας αντίστοιχα, προσθέσαμε ένα ακόμα Reducer στο πρόγραμμα. Στη συνέχεια μεταβάλλαμε τον αριθμό των Mappers από 1 μέχρι και 3, προκειμένου να διαπιστώσουμε τη συμπεριφορά του συστήματος. Οι συνθήκες παρέμειναν οι ίδιες, δηλαδή datafiles 100 και 88000 εγγραφών, parameters file με ποικίλα threshold από 20% έως και 90% και hash functions 50, 100, 150 και 200.

Οι πίνακες από τους οποίους προέκυψαν τα γραφήματα, για το αρχείο 100 εγγραφών, παρατίθενται στο Παράρτημα 3 (πίνακες Π3.4, Π3.5 και Π3.6, για ένα, δύο και τρεις Mappers αντίστοιχα).

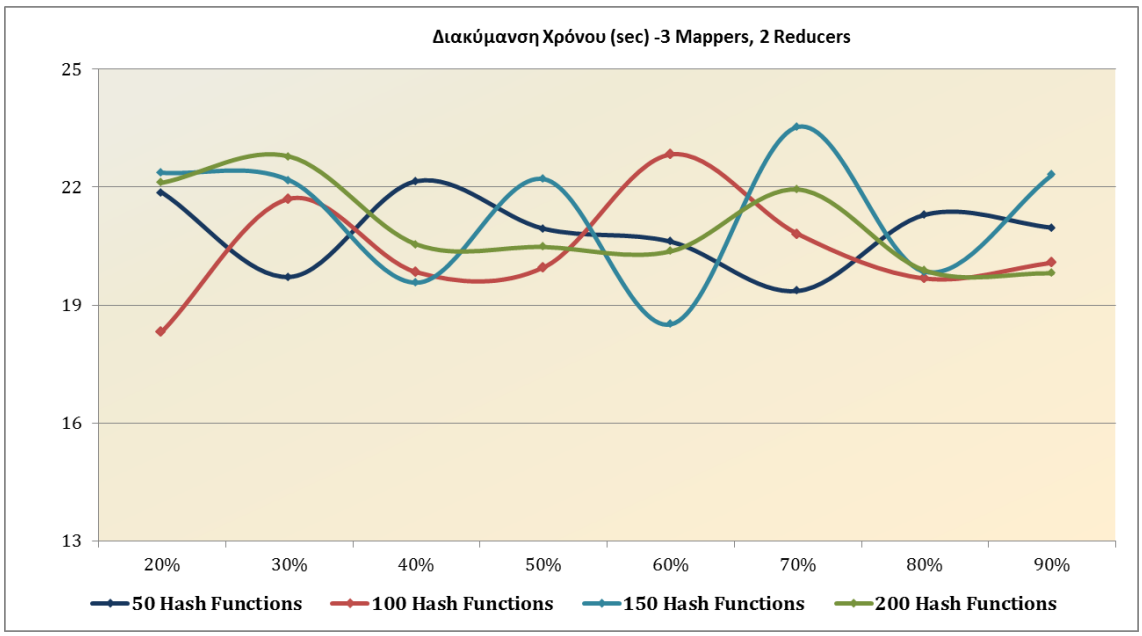
Ανάλογα με τον αριθμό των Mappers που χρησιμοποιήθηκαν, για το αρχείο 100 εγγραφών, προέκυψαν και τα παρακάτω γραφήματα (σχήματα 5.4, 5.5 και 5.6).



Σχήμα 5.4. Αποτελέσματα με τη χρήση ενός Mapper.

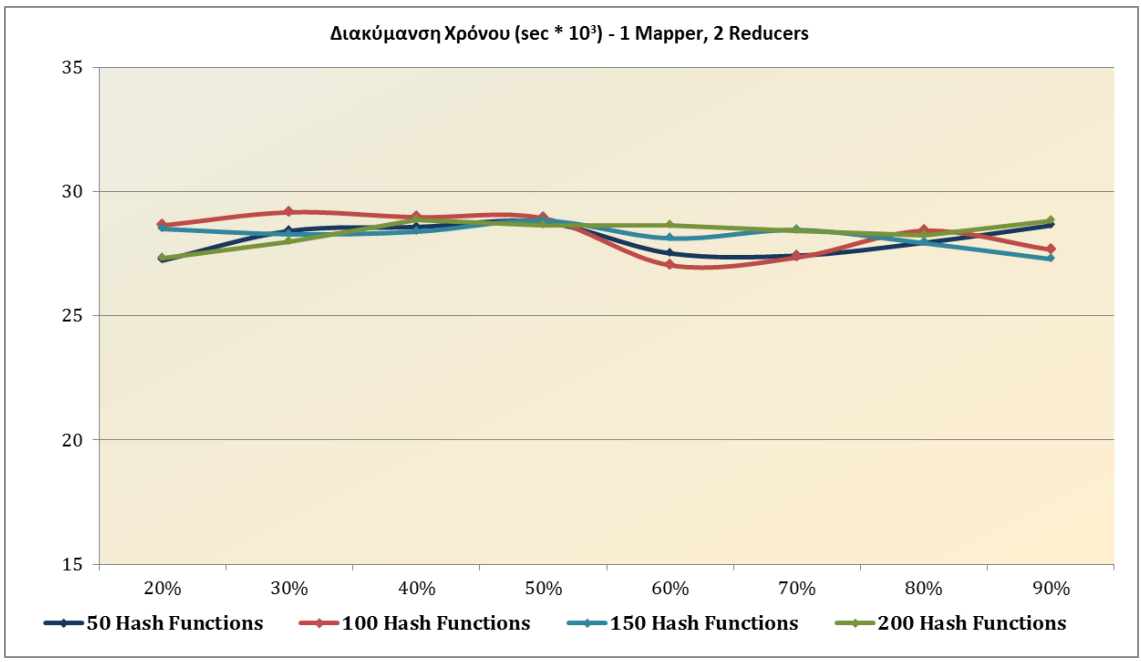


Σχήμα 5.5. Αποτελέσματα με τη χρήση δύο Mappers.

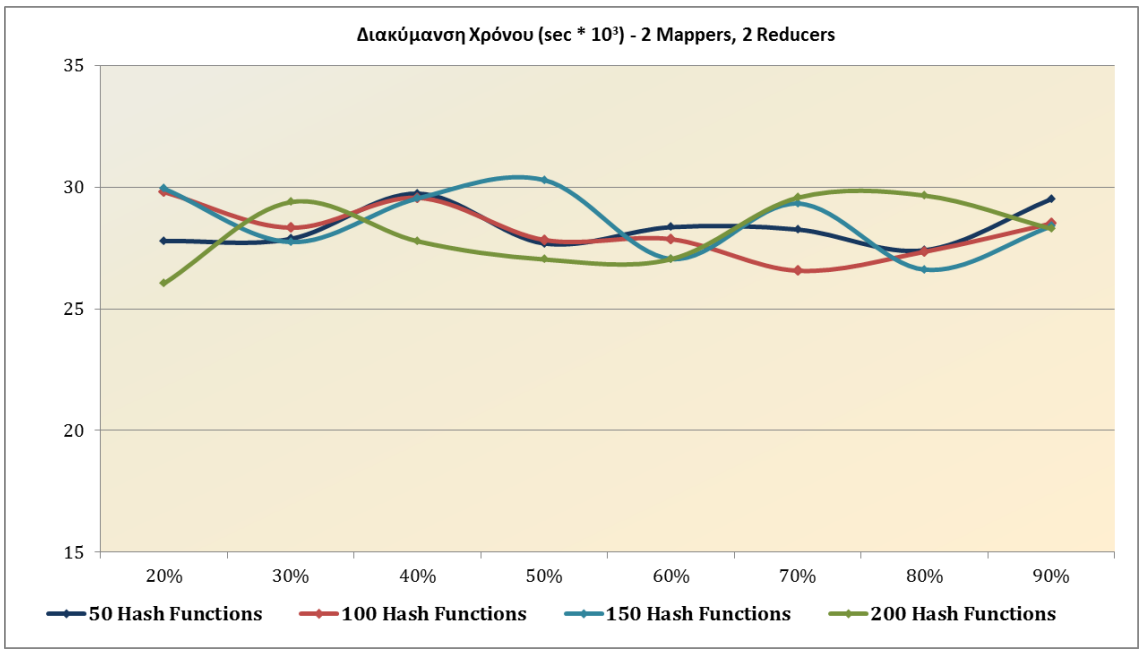


Σχήμα 5.6. Αποτελέσματα με τη χρήση τριών Mappers.

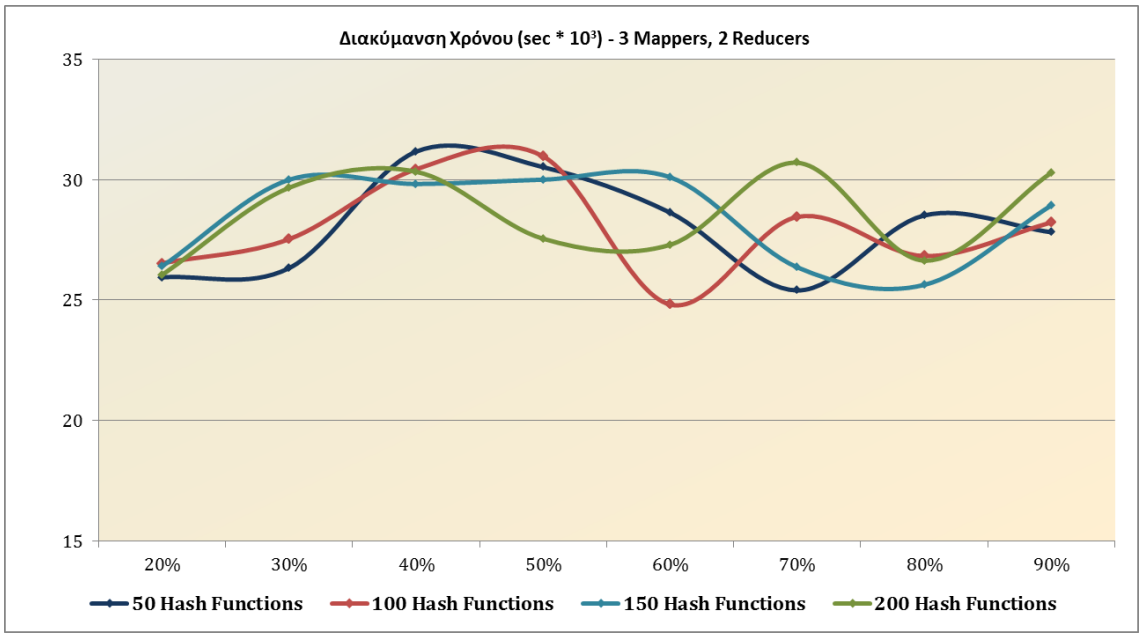
Αντίστοιχα για το αρχείο 88000 εγγραφών, οι πίνακες από τους οποίους προέκυψαν τα γραφήματα, παρατίθενται στο Παράρτημα 3 (πίνακες Π3.7, Π3.8 και Π3.9, για ένα, δύο και τρεις Mappers αντίστοιχα). Προέκυψαν τα παρακάτω γραφήματα (σχήματα 5.7, 5.8 και 5.9).



Σχήμα 5.7



Σχήμα 5.8



Σχήμα 5.9

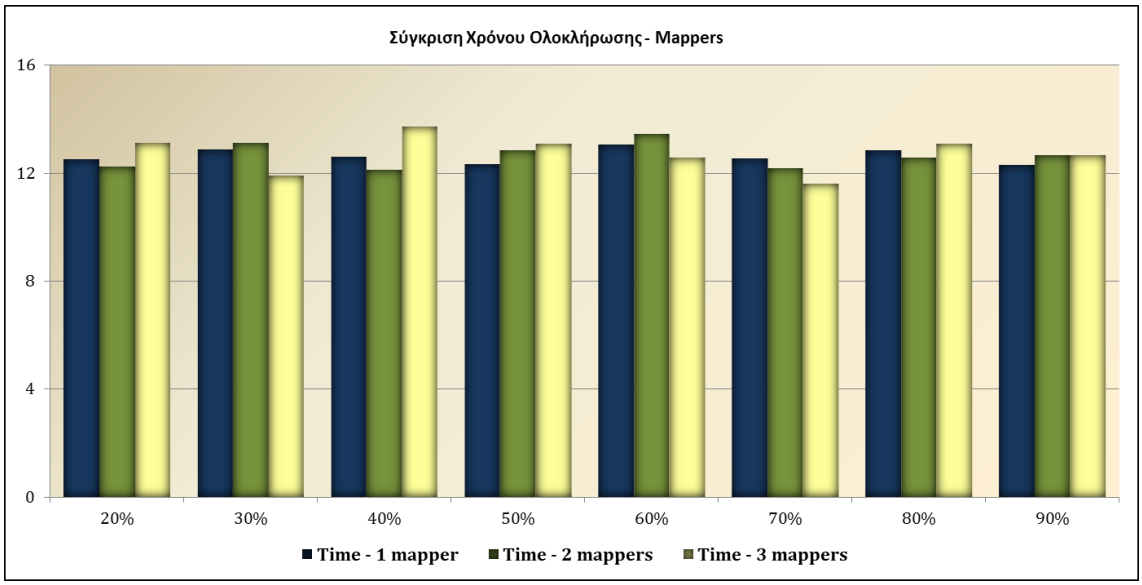
5.2.3. Χρήση datafiles 100 και 88000 εγγραφών με τέσσερεις Reducers.

Ενεργώντας αντίστοιχα, προσθέσαμε δύο ακόμα Reducers στο πρόγραμμα. Στη συνέχεια μεταβάλλαμε και πάλι τους Mappers από 1 μέχρι και 3. Οι συνθήκες παρέμειναν και πάλι οι ίδιες, δηλαδή datafiles 100 και 88000 εγγραφών, parameters file με ποικίλα threshold από 20% έως και 90% και hash functions 50, 100, 150 και 200.

Οι πίνακες για το αρχείο των 100 εγγραφών, παρατίθενται στο Παράρτημα 3 (πίνακες Π3.10, Π3.11 και Π3.12, για ένα, δύο και τρεις Mappers αντίστοιχα).

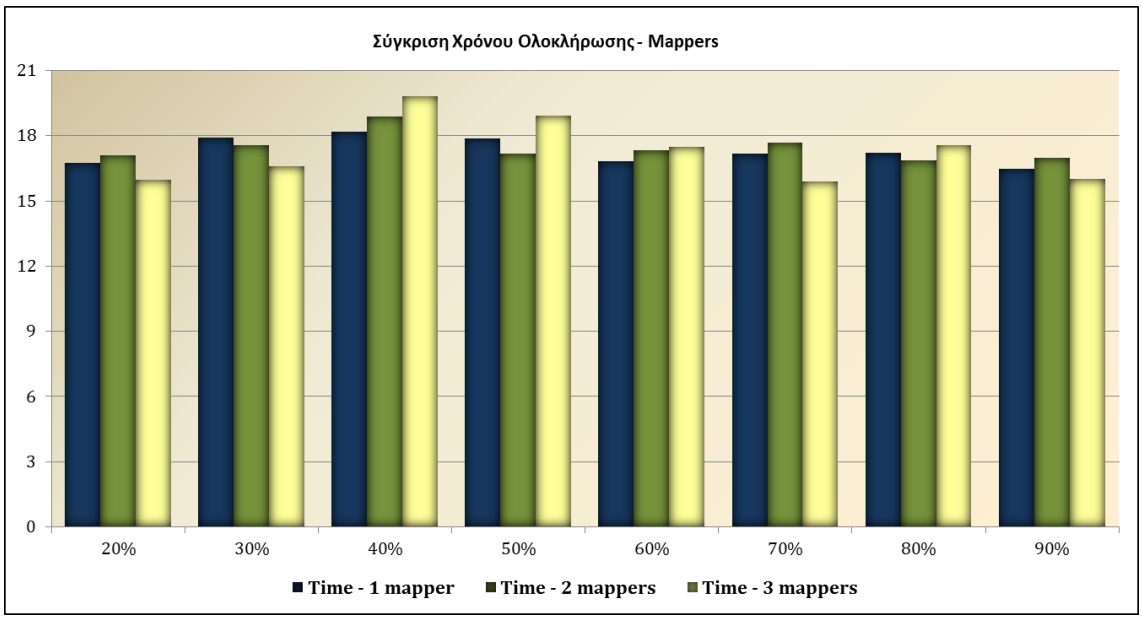
Αυτό που απεικονίζουμε γραφικά είναι ο χρόνος που απαιτείται με τη χρήση 1, 2 και 3 Mappers αντίστοιχα και για συγκεκριμένο αριθμό hash functions (50). Ο σκοπός είναι να δούμε τη διακύμανση του χρόνου και κατά πόσο αυτός εξαρτάται από τον αριθμό των hash functions που χρησιμοποιούνται.

Το γράφημα που προκύπτει είναι το παρακάτω (σχήμα 5.10) και ο πίνακας από τον οποίο δημιουργήθηκε, είναι ο Π3.13 του παραρτήματος Γ.



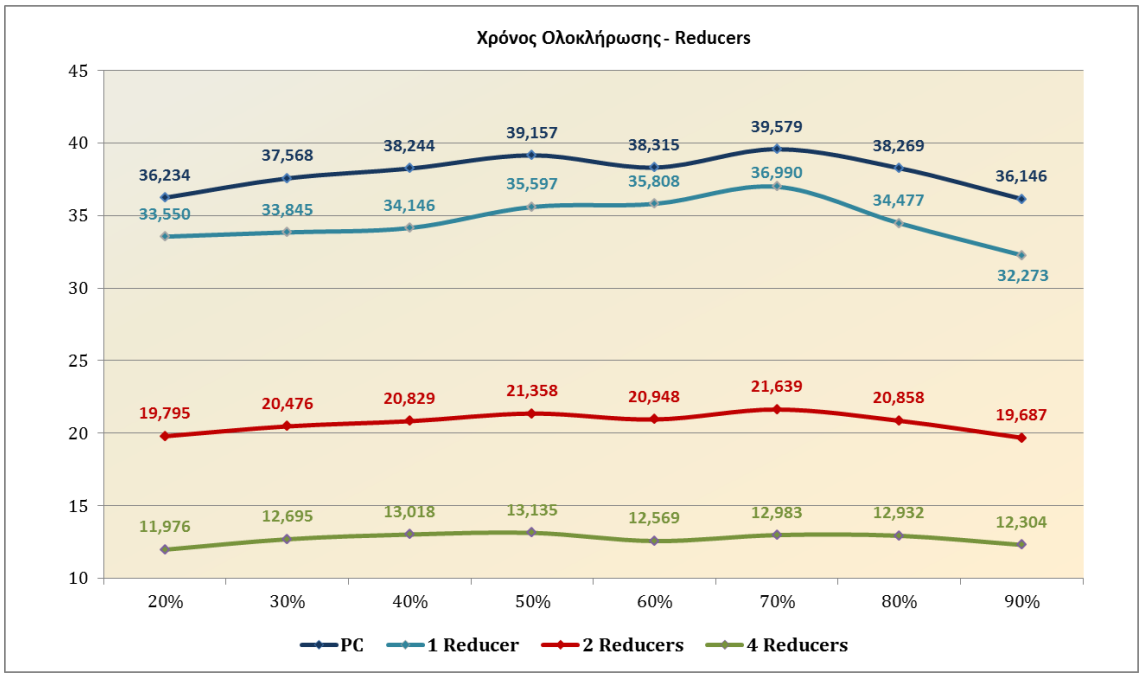
Σχήμα 5.10.

Αντίστοιχα για το αρχείο 88000 εγγραφών, θα έχουμε το γράφημα του σχήματος 5.11. Αντίστοιχα με τα προηγούμενα βήματα, δημιουργήθηκαν οι πίνακες Π3.14, Π3.15 και Π3.16, που παρατίθενται στο Παράρτημα Γ.

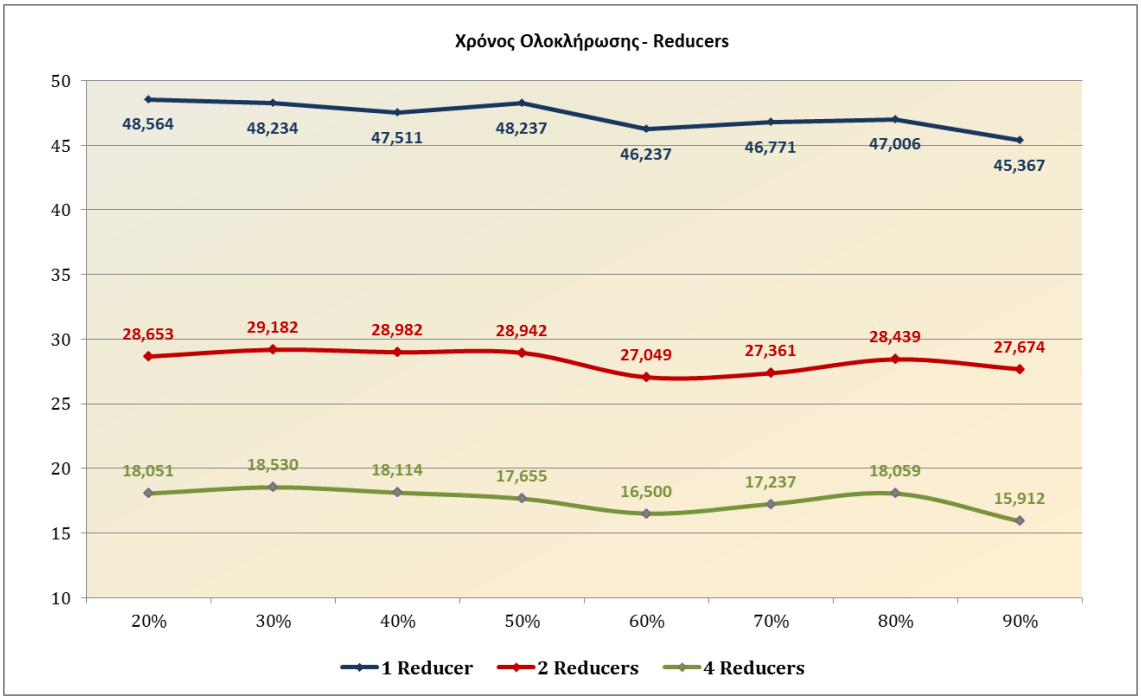


Σχήμα 5.11.

Τέλος δημιουργήθηκε ο πίνακας Π3.16, στον οποίο παριστάνεται ο χρόνος ολοκλήρωσης της διαδικασίας σε συνάρτηση με τον αριθμό των Reducers. Το αντίστοιχο γράφημα για τις 100 εγγραφές είναι το παρακάτω (σχήμα 5.12). Για τις 88.000 εγγραφές αντίστοιχα, ακολουθεί αυτό του σχήματος 5.13.



Σχήμα 5.12.



Σχήμα 5.13.

Στο σχήμα 5.12 έχουμε να παρατηρήσουμε ότι πρακτικά, σε ένα αρχείο δεδομένων μεγαλύτερο από 100 εγγραφές, η διαφορά στο χρόνο υλοποίησης θα ήταν μεγαλύτερη, από ότι είναι για το συγκεκριμένο αρχείο, που λόγω του μικρού του μεγέθους αμβλύνει κάπως την απόσταση. Είναι όμως καθαρό και αυτό εικονίζεται και στο σχήμα 5.13, ότι με την αύξηση του αριθμού των Reducers, μειώνεται σημαντικά ο χρόνος ολοκλήρωσης.

5.3 ΣΥΜΠΕΡΑΣΜΑΤΑ ΑΠΟ ΤΗΝ ΥΛΟΠΟΙΗΣΗ.

Ξεκινάμε την καταγραφή των συμπερασμάτων μας από το χρόνο που απαιτείται για την ολοκλήρωση της διαδικασίας.

1. Ο χρόνος ολοκλήρωσης της διαδικασίας μεταβάλλεται ελάχιστα καθώς μεταβάλλονται, τόσο το ζητούμενο threshold, όσο και οι hash functions που χρησιμοποιούνται, είτε η διαδικασία διεκπεραιώνεται σε ένα και μόνο απλό υπολογιστή, είτε αυτή διεκπεραιώνεται σε συστοιχία υπολογιστών. Αυτό πρακτικά σημαίνει ότι η διαδικασία μας έχει προκαθορισμένο χρόνο ολοκλήρωσης ανεξάρτητα από το ποιο θα είναι το ποσοστό της ομοιότητας που αναζητούμε (threshold) και του πόσες hash functions θα χρησιμοποιήσουμε για να την εντοπίσουμε.

Πρέπει ωστόσο να αναφερθεί, ότι ο αριθμός των hash functions επηρεάζει άμεσα τον πίνακα υπογραφών (signature table) και κατ' επέκταση των αριθμό των ζευγών εξόδου. Έτσι ο χρόνος ολοκλήρωσης της διαδικασίας θα ήταν κατά πολύ διαφορετικός εάν για παράδειγμα συγκρίναμε δύο αντικείμενα με 50 γραμμές (τιμές) και δύο άλλα με 500 αντίστοιχα.

Το γεγονός επίσης ότι η επιλογή του threshold επηρεάζει ελάχιστα το χρόνο εκτέλεσης της διαδικασίας, μπορεί μεν να εγγυάται μια σταθερότητα στη χρήση της μεθόδου, πρέπει όμως να ληφθεί υπ' όψιν και ο χρόνος αποθήκευσης στο HDFS του περιεχομένου της εξόδου, καθώς ο αριθμός των ομοιαζόντων ζευγών ποικίλει, το ίδιο επομένως και ο αντίστοιχος χρόνος αποθήκευσής τους, ο οποίος θα πρέπει να προσμετρηθεί στη συνολική εκτίμηση κόστους της διαδικασίας, μια και θα είναι διαφορετικός για ένα αποτέλεσμα 50 ζευγών και διαφορετικός για αποτέλεσμα 1000.

2. Ο χρόνος ολοκλήρωσης της διαδικασίας, είναι ανεξάρτητος του αριθμού των Mappers που θα χρησιμοποιηθούν σε αυτήν. Αυτό φαίνεται από την πολύ μικρή μεταβολή που παρατηρείται στο χρόνο, αλλάζοντας τον αριθμό των Mappers που χρησιμοποιούμε. Πρέπει όμως να ληφθεί υπ' όψη το γεγονός ότι για την εκκίνηση της λειτουργίας των Mappers, πραγματοποιείται μια σειρά ενεργειών, όπως η αρχικοποίηση μεταβλητών, η έναρξη συνδέσεων με το HDFS κλπ, όπου απαιτείται επιπλέον χρόνος.

3. Ο χρόνος ολοκλήρωσης της διαδικασίας εξαρτάται άμεσα από τον αριθμό των Reducers που χρησιμοποιούμε. Όσο περισσότεροι είναι οι Reducers, τόσο μικρότερος είναι ο χρόνος ολοκλήρωσης. Υπάρχει ωστόσο και ένα όριο σε αυτό, αφού δε μπορεί να χρησιμοποιηθούν περισσότεροι Reducers από τον αριθμό των υπό σύγκριση ζευγών.

4. Για την επίτευξη της διαδικασίας είναι απαραίτητη η χρήση της Distributed Cache, αφού χωρίς τη χρήση της δε θα μπορούσε να υπάρξει υλοποίηση.

5. Προκειμένου α αποφευχθούν αρχικοποιήσεις του συστήματος, οι οποίες θα είχαν επίπτωση στο χρόνο εκτέλεσης, μια και επιδιώκουμε να βρούμε τι επιρροή έχει στο σύστημά μας ο αριθμός των Mappers και ο αριθμός των Reducers, υπήρξε η προσπάθεια να εναρμονιστούν στον ίδιο αριθμό, τόσο οι Mappers με τα Map Tasks, όσο και οι Reducers με τα Reduce Tasks.

6. Η υλοποίηση έγινε εξετάζοντας τις γραμμές του πίνακα εισόδου. Όσο περισσότερο αυξάνονται οι Mappers, διαχωρίζονται μεν οι στήλες, με αποτέλεσμα να επιταχύνεται η διαδικασία, αλλά αυξάνονται και τα ζεύγη εξόδου της Map .

ΚΕΦΑΛΑΙΟ 6
ΣΥΜΠΕΡΑΣΜΑΤΑ

6.1 ΓΕΝΙΚΑ ΣΥΜΠΕΡΑΣΜΑΤΑ.

Η τεχνική του Map Reduce μας παρέχει τη δυνατότητα να αναζητήσουμε ομοιότητα ανάμεσα σε μεγάλα συστήματα, χωρίς να υπάρχει η ανάγκη συλλογής των δεδομένων κεντρικά, αλλά με άμεση επεξεργασία αυτών τοπικά.

Υπάρχει κέρδος στο χρόνο εκτέλεσης και αυτό μας δίνει τη δυνατότητα της επεξεργασίας δυναμικών δεδομένων, διαρκώς δηλαδή μεταβαλλόμενα.

Η εύρεση ομοιότητας κατά προσέγγιση, επιτυγχάνεται με τη χρήση της μεθόδου MinHash. Οι hash functions είναι εύκολο να χρησιμοποιηθούν, γιατί δημιουργούνται με μια απλή συνάρτηση και ο αριθμός αυτών που θα χρησιμοποιηθούν, δεν επηρεάζει καθοριστικά ούτε τους πόρους του συστήματος κατά τη διάρκεια εκτέλεσης της διαδικασίας, ούτε και το χρόνο εκτέλεσης. Ο αριθμός των hash functions δε χρειάζεται να πλησιάζει στον αριθμό των υπό επεξεργασία

εγγραφών. Ενδεικτικά στο πείραμα χρησιμοποιήσαμε τον ίδιο αριθμό τόσο για ένα πολύ μικρό αρχείο δεδομένων 100 εγγραφών, όσο και για ένα μεγαλύτερο, 88000 εγγραφών. Σίγουρα όμως η ακρίβεια του αποτελέσματος, είναι άμεσα εξαρτώμενη από τον αριθμό αυτό και όσο περισσότερες χρησιμοποιηθούν, τόσο ακριβέστερο θα είναι και το αποτέλεσμα. Βέβαια στο είδος της ομοιότητας που αναζητούμε, προσεγγιστική δηλαδή, η απαίτηση για ακριβή αποτελέσματα είναι μειωμένη. Στο επίπεδο αυτό, η κάλυψη που μας παρείχε ο MinHashingMR, ήταν επαρκής.

Η μέθοδος δύναται να χρησιμοποιηθεί και σε πολύ μεγάλα αρχεία δεδομένων, όπως είναι αυτά του Google News, ή του Amazon, όπου βέβαια και η υπολογιστική ισχύς είναι κατά πολύ μεγαλύτερη αυτής που χρησιμοποιήθηκε για τα πειράματα της παρούσης. Το δε αρχείο των 88.000 εγγραφών που χρησιμοποιήθηκε στο πείραμα, μπορούμε να πούμε ότι είναι αρκετό προκειμένου να ελεγχθεί η συμπεριφορά ενός «μεγάλου» αρχείου δεδομένων, εάν ληφθεί υπ' όψη ότι το σύστημα που χρησιμοποιήθηκε αποτελείται από τέσσερεις κόμβους και κυρίως ότι πραγματοποιούνται υπολογισμοί, κατά τους οποίους συγκρίνονται ανά δύο και όλα, 88.000 αντικείμενα, κάτι που ουσιαστικά καταδεικνύει την υπολογιστική δύναμη που απαιτεί η διαδικασία.

Η χρήση της Distributed Cache είναι απαραίτητη, αφού ο πίνακας εισόδου είναι διανεμημένος σε όλους τους κόμβους και με τον τρόπο αυτό ξεπερνάμε την ανάγκη προσπέλασης όλων των κόμβων κάθε φορά που χρειάζεται να διαβαστεί μια εγγραφή.

Τέλος, επειδή από τη διαδικασία επιθυμούμε να λάβουμε αποτελέσματα που δε θα δείχνουν την ομοιότητα με ακρίβεια, αλλά σε μια πιο αφηρημένη μορφή, μπορούμε να επιλέξουμε το βαθμό της ομοιότητας αυτής, γνωρίζοντας εκ των προτέρων ότι δε θα υπάρχουν ουσιαστικές διαφορές στη συμπεριφορά του συστήματος, κάτι που δημιουργεί ευελιξία στη χρήση του και διευρύνει το πεδίο εφαρμογής του.

Στην πορεία της έρευνας μελετήθηκε – παρατίθεται και στο κεφάλαιο 3 και ο αλγόριθμος LSH (Locality Sensitive Hashing). Ο συγκεκριμένος αλγόριθμος θα μπορούσε να αποτελέσει μια συνέχεια του Min Hashing, αφού είναι βασισμένος ο τρόπος λειτουργίας του στις hash functions και ταυτόχρονα θα μπορούσε να δώσει ακόμα πιο βελτιωμένα αποτελέσματα από αυτά του Min Hash, τόσο ως προς το χρόνο ολοκλήρωσης, όσο και ως προς τη δέσμευση υπολογιστικής ισχύος. Ο χρόνος όμως δεν ήταν επαρκής προκειμένου να πραγματοποιηθεί μια αντίστοιχη μελέτη και γι' αυτόν.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[01] Abhinandan Das, Mayur Datar, Ashutosh Garg, "Google News Personalization: Scalable Online Collaborative Filtering", WWW 2007 / Track: Industrial Practice and Experience.

[02] Anand Rajaraman, Jeffrey D. Ullman, "Mining of Massive Datasets".

[03] Ashish Goel, MS&E 317/CS 263: Algorithms for Modern Data Models, Aut 2011-12, <http://www.stanford.edu/~ashishg/amdm.>.

[04] Christopher Olston, Benjamin Reedy, Utkarsh Srivastavava, Ravi Kumar, Andrew Tomkins, "Pig Latin: A Not-So-Foreign Language for Data Processing", https://cwiki.apache.org/confluence/display/PIG/Index%3bjsessionid=76A40E651D2F6F0138CB6023B32870E8.

[05] Foto N. Afrati, Anish Das Sarma, David Menestrina, Aditya Parameswaran, Jeffrey D. Ullman, "Fuzzy Joins Using MapReduce".

[06] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters".

[07] Matthew Damigos, presentation "Google News Personalization", May 8-12, 2007.

[08] Michael Noll., «Hadoop tutorial», < http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/#running-a-mapreduce-job>, Ημερομηνία τελευταίας πρόσβασης: 10/4/2012.

[09] Naveenkumar Selvaraj, presentation "Google News Personalization: Scalable Online Collaborative Filtering", May 8-12, 2007.

[10] Serge Abitebul, Ioanna Manolescu, Philippe Rigaux, Marrie Christine Rousset, Pierre Senellart, "Distributed Computing with MAP REDUCE and PIG", < http://webdam.inria.fr/Jorge/>.

[11] Serge Abitebul, Ioanna Manolescu, Philippe Rigaux, Marrie Christine Rousset, Pierre Senellart, "Introduction to recommendation systems", < http://webdam.inria.fr/Jorge/>.

[12] Serge Abitebul, Ioanna Manolescu, Philippe Rigaux, Marrie Christine Rousset, Pierre Senellart, "Putting into practice: HADOOP", < http://webdam.inria.fr/Jorge/>.

[13] Serge Abitebul, Ioanna Manolescu, Philippe Rigaux, Marrie Christine Rousset, Pierre Senellart, "WEB Data Management", < http://webdam.inria.fr/Jorge/>.

[14] Tom White, "Hadoop: The Definitive Guide", 2nd edition, Yahoo! press, ISBN: 978-1-449-38973-4.

[15] U Kang, "Mining Tera-Scale Graphs with MapReduce: Theory, Engineering and Discoveries", October 2011 CMU-CS-11-XXX.

[16] Yaser S Abu – Mostafa, presentation "Learning From Data", "Error and Noise", Caltech's Provost Office E & AS Division and IST, April 12 2012.

[17] Yaser S Abu – Mostafa, presentation "Learning From Data", "Is learning fisible?", April 5 2012.

[18] Yaser S Abu – Mostafa, presentation "Learning From Data", "Linear models I", April 10 2012.

[19] Yaser S Abu – Mostafa, presentation "Learning From Data", "The learning problem", April 3 2012.

[20] Yaser S Abu – Mostafa, presentation "Learning From Data", "Training vs Testing", April 17 2012.

[21] NCDC, *http://www.ncdc.noaa.gov/*.

[22] «Hadoop MapReduce Tutorial», <http://hadoop.apache.org/common/docs/current/mapred_tutorial.html>, Ημερομηνία τελευταίας πρόσβασης: 10/3/2012.

[23] «Hadoop MapReduce Tutorial», http://hadoop.apache.org/common/docs/current/mapred_tutorial.html#DistributedCache, Ημερομηνία τελευταίας πρόσβασης: 12/4/2012.

[24] «Hadoop: υλοποίηση ανοιχτού κώδικα».

ΠΑΡΑΡΤΗΜΑ Α
ΕΓΚΑΤΑΣΤΑΣΗ Hadoop

Α.1 ΕΓΚΑΤΑΣΤΑΣΗ ΠΡΟΑΠΑΙΤΟΥΜΕΝΩΝ

Ο κώδικας που τρέξαμε προκειμένου να πάρουμε τα αποτελέσματα, απαίτησε τη δημιουργία μιας πλατφόρμας, την οποία περιγράφουμε βηματικά παρακάτω.

1. Το λειτουργικό που χρησιμοποιήσαμε ως τη βάση για την εγκατάσταση της πλατφόρμας, είναι το Ubuntu. Κατεβάζουμε το iso του Ubuntu από το http://www.ubuntu.com/start-download?distro=desktop&bits=32&release=latest .

2. Κατεβάζουμε από το https://my.vmware.com/web/vmware/evalcenter?p=player το VMPlayer, που είναι η εικονική μηχανή στην οποία θα εγκαταστήσουμε το Ubuntu και το εγκαθιστούμε στον υπολογιστή εργασίας μας.

3. Εγκαθιστούμε το ubuntu μέσα στο VMPlayer.

3a. Δίνουμε προσοχή στο username και password.

3b. Εγκαθιστούμε και τα vmware tools όταν μας το προτείνει.

Α.2 ΕΓΚΑΤΑΣΤΑΣΗ Hadoop

Στη συνέχεια προχωράμε στην εγκατάσταση του Hadoop. Ξεκινάμε ανοίγοντας terminal.

1. Εγκαθιστούμε το jdk 1.6 της Sun για τη Java (λειτουργεί καλύτερα), εκτελώντας τα παρακάτω βήματα.

α. sudo add-apt-repository "deb http://archive.canonical.com/ lucid partner"
b. sudo apt-get update
c. sudo apt-get install sun-java6-jdk
Αποδεχόμαστε licences.
d. Ελέγχουμε αν εγκαταστάθηκε σωστά η java:
sudo update-java-alternatives -s java-6-sun.

2. Εγκαθιστούμε ssh-server:

sudo apt-get install openssh-server

3. Δημιουργούμε κενά κλειδιά για το ssh:

ssh-keygen -t rsa -P ""

Στο σημείο αυτό, για να σώσουμε το κλειδί, απλά πατάμε Enter.

4. Αποθηκεύουμε το κλειδί:

cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys

5. Δοκιμάζουμε για πρόσβαση: ssh localhost

Εάν εγκατασταθεί, ουσιαστικά έχουμε «μπει» στο εικονικό μηχάνημα μας, και με exit επιστρέφουμε.

6. Απενεργοποιούμε το ipv6:

```
sudo nano /etc/sysctl.conf
```

Στον editor, στο τέλος του κειμένου προσθέτουμε τα παρακάτω:

```
#disable ipv6
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

7. Αποθηκεύουμε με Ctrl+X και μετά κάνουμε reboot.

8. Κάνουμε download το Hadoop (συμπιεσμένο) και το εγκαθιστούμε στο /usr/local/hadoop:

```
wget -c http://apache.tsl.gr/hadoop/common/hadoop-0.20.2/hadoop-0.20.2.tar.gz
sudo mv hadoop-0.20.2.tar.gz /usr/local/
cd /usr/local
sudo tar xzf hadoop-0.20.2.tar.gz
sudo mv hadoop-0.20.2 hadoop
sudo chown -R name:name hadoop
```

9. Δημιουργούμε κάποιες συντομεύσεις που θα διευκολύνουν κάποιες από τις εργασίες που θα πραγματοποιήσουμε:

```
sudo nano ~/.bashrc
```

και προσθέτουμε στο τέλος του κειμένου:

```
##################
# Set Hadoop-related environment variables
export HADOOP_HOME=/usr/local/hadoop
# Set JAVA_HOME (we will also configure JAVA_HOME directly for Hadoop later on)
export JAVA_HOME=/usr/lib/jvm/java-6-sun
# Some convenient aliases and functions for running Hadoop-related commands
unalias fs &> /dev/null
alias fs="hadoop fs"
unalias hls &> /dev/null
alias hls="fs -ls"
# If you have LZO compression enabled in your Hadoop cluster and
# compress job outputs with LZOP (not covered in this tutorial):
# Conveniently inspect an LZOP compressed file from the command
# line; run via:
#
# $ lzohead /hdfs/path/to/lzop/compressed/file.lzo
#
# Requires installed 'lzop' command.
#
lzohead () {
   hadoop fs -cat $1 | lzop -dc | head -1000 | less
}
# Add Hadoop bin/ directory to PATH
export PATH=$PATH:$HADOOP_HOME/bin
############################
```

10. Αλλαγή default java jdk:

sudo nano /usr/local/hadoop/conf/hadoop-env.sh

Εδώ θα αλλάξουμε μόνο τη γραμμή 8 σε:

export JAVA_HOME=/usr/lib/jvm/java-6-sun

11. sudo mkdir -p /app/hadoop/tmp

sudo chown -R name:name /app/hadoop/tmp

12. sudo nano /usr/local/hadoop/conf/core-site.xml

και μέσα στο tag <configuration> κάνουμε επικόλληση:

```
<!-- In: conf/core-site.xml -->
<property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
  <description>A base for other temporary directories.</description>
</property>
<property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:54310</value>
  <description>The name of the default file system.  A URI whose  scheme and authority determine the FileSystem implementation.  The uri's scheme determines the config property (fs.SCHEME.impl) naming  the FileSystem implementation class.  The uri's authority is used to determine the host, port, etc. for a filesystem.</description>
</property>
```

13. Όμοια,

```
sudo nano /usr/local/hadoop/conf/mapred-site.xml
<!-- In: conf/mapred-site.xml -->
<property>
  <name>mapred.job.tracker</name>
  <value>localhost:54311</value>
  <description>The host and port that the MapReduce job tracker runs  at.  If "local", then jobs are run in-process as a single map  and reduce task.
  </description>
</property>
```

14. Ακόμα,

```
sudo nano /usr/local/hadoop/conf/hdfs-site.xml
<!-- In: conf/hdfs-site.xml -->
<property>
 <name>dfs.replication</name>
 <value>1</value>
 <description>Default block replication.
 The actual number of replications can be specified when the file is created.
 The default is used if replication is not specified in create time.
 </description>
</property>
```

15. Στη συνέχεια κάνουμε format τον εικονικό διανεμημένο δίσκο:

```
hadoop namenode -format
```

16. Ενεργοποιούμε το Hadoop:

```
start-all.sh
```

17. Ελέγχουμε αν έχει ενεργοποιηθεί με jps και αν έχει ενεργοποιηθεί θα πρέπει να εμφανίζει αυτό το αποτέλεσμα:

```
7364 TaskTracker
6652 NameNode
7092 SecondaryNameNode
7156 JobTracker
6884 DataNode
7523 Jps
```

18. Κλείνουμε αντίστοιχα το Hadoop με την εντολη stop-all.sh

19. Για να γράψουμε κώδικα για το Hadoop, χρησιμοποιήσαμε το Eclipse. Κατεβάζουμε λοιπόν μέσα από ubuntu, Eclipse και εγκαθιστούμε στο /home/name/

wget -c ftp://ftp.ntua.gr/pub/devel/eclipse/eclipse/downloads/drops/R-3.6.2-201102101200/eclipse-SDK-3.6.2-linux-gtk.tar.gz

tar xzf eclipse-SDK-3.6.2-linux-gtk.tar.gz

20. Εγκαθιστούμε το Karmasphere Plugin για το Eclipse. Το συγκεκριμένο βοήθησε στην εξοικείωση με το Map Reduce.

21.Ανοίγουμε το Eclipse και δηλώνουμε ένα workspace πχ /home/name/hadoopworkspace/

22. Από το μενού πάμε στο Help>Install New Software

23. Και επιλέγουμε Add και στο Name "Karmasphere Plugin", δίνοντας Location:

http://updates.karmasphere.com/dist/KwraJ9CkQpEJIvkHqywX_I2B0Jc/eclipse/

http://updates.karmasphere.com/dist/7Tk0p_M2L8rwyiCLToPxgk6pra4/eclipse/

24. Επιλέγουμε Karmasphere > Next > Next > Accept > Finish και περιμένουμε να γίνουν download τα αιτούμενα πακέτα. Μετά κάνουμε restart.

25. Στη συνέχεια κάνουμε gedit ~/eclipse/eclipse.ini και προσθέτουμε/αλλάζουμε παραμέτρους .

-vmargs
-Xms512m
-Xmx1024m
-XX:MaxPermSize=512m
-Dosgi.classloader.lock=classname
-Dosgi.requiredJavaVersion=1.6

ΠΑΡΑΡΤΗΜΑ Β

Ο ΚΩΔΙΚΑΣ

Β.1 ΔΗΜΙΟΥΡΓΙΑ parameters.txt

```
package minhashinghashfunctiongen;

import java.io.*;
import java.util.ArrayList;


public class MinHashingHashFunctionGen {

    /**
     * @param args the command line arguments
     * @param arg[0]: Give the number of Hash Functions
     * @param arg[1]: Give the full-path of the parameters file (ATTENTION: the
     * filename must be included in the path).
     * @param arg[2]: threshold value
     */
    public static void main(String[] args) {

/*Form of parameters file:
 * 1st line: Number of table rows
 * 2nd line: Number of table columns
 * 3rd line: Number of hash functions
 * 4th line: threshold value (percent number, e.g., 50 which means 50%)
 * 5rd line: The separator phrase: Hash Values
 * 6th-...: each line has (k+1)-values separated by the char ";", where k is the number of hash functions.
```

```
 * The first value indicates the number of input value (of the hash functions, i.e.,
 * the number of row).
 * The values appearing in the (j+1)place of the line indicates the output value of the j-th hash function.
 */


//Determine the Number of Hash-Functions:
	int num_of_hashF = Integer.parseInt(args[0]);
//Determine the full-path of the file in which the parameters will be stored.
	File par_file = new File(args[1]);
	try {

//Initialize the Output and Print streams
		FileOutputStream out;
		PrintStream p;
		out = new FileOutputStream(par_file, false);
		p = new PrintStream(out);
		/*
		 * The file containing the data must be named as datafile.txt. Also
		 * the file must be placed in the same directory to that containing
		 * the parameters file.
		 */
		LineNumberReader reader = new LineNumberReader(new FileReader(new
File(par_file.getParent(), "datafile.txt")));
		String lineRead = "";
		// Attention: The number of lines of the file identifies the number of users, i.e., the number of
columns of data table.

		int numberOfColumns = 0;
		int number_of_rows = 0;
		while ((lineRead = reader.readLine()) != null) {
		   String[] items = lineRead.trim().split(" ");
		   for (int i = 1; i < items.length; i++) {
		      if (Integer.parseInt(items[i]) > number_of_rows) {
		         number_of_rows = Integer.parseInt(items[i]);
		      }
		   }
		}

		   numberOfColumns = reader.getLineNumber();

		reader.close();

		p.println(number_of_rows);
		p.println(numberOfColumns);
		p.println(num_of_hashF);
		p.println(args[2]);
		p.println("Hash Values");
		RandomPermutation perm = new RandomPermutation(number_of_rows);

		ArrayList<String> hash_values = new ArrayList(number_of_rows);
// Attention: in each data line (i.e., hash values) of the parameters file,
//we store the number of row and the output value for each hash function and row number.
		for (int i = 0; i < num_of_hashF; i++) {
		   ArrayList<Integer> temp_perm = perm.next();
		   for (int k = 0; k < number_of_rows; k++) {
		      if (i == 0) {
		         String line = k + ";" + temp_perm.get(k);
		         hash_values.add(line);
		      } else {
		         String line = hash_values.get(k) + ";" + temp_perm.get(k);
		         hash_values.set(k, line);
		      }
```

```
            }
          }

          for (int k = 0; k < number_of_rows; k++) {
            p.println(hash_values.remove(0));
          }

          p.close();
          out.close();

        } catch (Exception e) {
          System.out.println(e);
          System.exit(0);
        }


        System.exit(0);
    }
}
```

B.2 ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ.

```
package minhashinghadoopv2;

import java.io.*;
import java.net.URI;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;


public class MinHashingHadoopV2 {

    /**
     * Loads the Driver
     *
     *
     * @param args
     * @param args[0]: local path of the folder you want to store statistics of the experiment.
     * @throws Exception
     */
//---------------------------------------------------------------------------
    public static void main(String[] args) {
        String localPath=args[0];
        run(localPath);
    }

    static void cacheItemList(JobConf conf) throws Exception {
//FileSystem fs = FileSystem.get(conf);
//Path hdfsPath = new Path(Parameters.HDFS_PARAMETERS_FILE);
// upload the file to hdfs. Overwrite any existing copy.
//fs.copyFromLocalFile(false, true, new Path(Parameters.LOCAL_PARAMETERS_FILE), hdfsPath);
DistributedCache.addCacheFile(new URI(Parameters.HDFS_PARAMETERS_FILE), conf);
//System.out.println(Parameters.HDFS_PARAMETERS_FILE);
    }
```

```
    public static void run(String localPath) {
        try {

File paramterers_file= new File(localPath, "parameters.txt");

if(!paramterers_file.exists()){
    System.out.println("Input Error!!!");
    System.out.println("The local path does not contain the file 'parameters.txt'");
    return;
}


            JobConf conf = new JobConf(MinHashingHadoopV2.class);
            conf.setJobName("MinHashingV2");
            conf.setOutputKeyClass(Text.class);
            conf.setOutputValueClass(Text.class);

            conf.setMapperClass(MinHashingMapper.class);
            conf.setReducerClass(MinHashingReducer.class);

            conf.setInputFormat(TextInputFormat.class);
            conf.setOutputFormat(TextOutputFormat.class);
//Specify the number of reducers/mappers.
//            int num_of_mappers= ;//Give an integer
//            int number_of_reducers= ;//Give an integer
//            conf.setNumMapTasks(num_of_mappers);
//            conf.setNumReduceTasks(number_of_reducers);

FileInputFormat.setInputPaths(conf, new Path(Parameters.INPUT_HDFS));
FileOutputFormat.setOutputPath(conf, new Path(Parameters.OUTPUT_HDFS));
 cacheItemList(conf);


//Start counting time
            double starttime = (double) System.currentTimeMillis();
//Run Job
            JobClient.runJob(conf);
//Stop counting time
            double endtime = (double) System.currentTimeMillis();


    FileSystem hdfs = FileSystem.get(conf);
    Path srcPath = new Path(Parameters.OUTPUT_HDFS);
    Path dstPath = new Path(localPath);
    hdfs.moveToLocalFile(srcPath, dstPath);


BufferedReader inFile = new BufferedReader(new FileReader(paramterers_file));
//Read the number of table rows.
String rows= inFile.readLine();
//Read the number of table columns.
String columns= inFile.readLine();
//Read the number of hash functions.
String num_hashF= inFile.readLine();
//Read the threshold.
String threshold= inFile.readLine();

//Print elapsed time.
//Initialize the file you print the time results
            File timeFile = new File(localPath, "time.txt");
//Initialize the Output and Print streams
            FileOutputStream out;
```

```
            PrintStream p;
            out = new FileOutputStream(timeFile, true);
            p = new PrintStream(out);
            p.println("Rows: "+rows+" ; "+"Columns: "+columns+" ; "+"Hash Functions: "+num_hashF+" ;
"+"Threshold: "+threshold+"% ; "+"Elapsed time: "+((endtime - starttime) * 0.001));
            p.close();
            out.close();



        } catch (Exception eio) {
            System.err.println(eio.getMessage());
        }
    }

//--------------------------------------------------------------------------


}
```

B.3 Η ΔΙΑΔΙΚΑΣΙΑ Mapper.

```
package minhashinghadoopv2;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.net.URI;
import java.util.ArrayList;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;


public class MinHashingMapper extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

    private Path[] localFiles;
    private URI[] localFilesURI;
    private ArrayList<Integer> a;
    private ArrayList<Integer> b;
    private static int num_of_rows;
    private static int num_of_columns;
    private static int num_of_hashFunctions;
    private static int threshold;

    public void configure(JobConf job) {
        try {
            //Cluster implementation.
//            localFilesURI = DistributedCache.getCacheFiles(job);
//            String temp=localFilesURI[0].toString();
```

```
//            Path str=new Path(temp);
//            hashTableGenerator(str);
            //Single-node Hadoop implementation.
            localFiles = DistributedCache.getLocalCacheFiles(job);
            hashTableGenerator(localFiles[0]);



        } catch (IOException ioe) {
            System.err.println("IOException reading from distributed cache");
            System.err.println(ioe.toString());
        }
    }

    void hashTableGenerator(Path cachePath) throws IOException {
        BufferedReader cacheReader = new BufferedReader(new FileReader(cachePath.toString()));

        try {
            String line;
            //Read number of table rows
            num_of_rows = Integer.parseInt(cacheReader.readLine());
            //Read number of table columns
            num_of_columns = Integer.parseInt(cacheReader.readLine());
            //Read number of hash functions
            num_of_hashFunctions = Integer.parseInt(cacheReader.readLine());
            //Read threshold
            threshold = Integer.parseInt(cacheReader.readLine());
            //Read the separator line
            line = cacheReader.readLine();
            a = new ArrayList();
            b = new ArrayList();
            while ((line = cacheReader.readLine()) != null) {
                String[] values = line.trim().split(";");
                a.add(new Integer(values[1].trim()));
                b.add(new Integer(values[2].trim()));
            }

            cacheReader.close();
        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }

    @Override
    public void map(LongWritable key, Text value,
            OutputCollector<Text, Text> output, Reporter reporter)
            throws IOException {
        String[] line = value.toString().trim().split(" ");

        int columnNumber = Integer.parseInt(line[0].trim());
        ArrayList<String> signValues = new ArrayList<String>();
        int row = Integer.parseInt(line[1].trim());

        String signString = "";
        if ((line.length - 1) == 1) {
            signString = "" + hashF(a.get(0), b.get(0), row, num_of_rows);
            for (int i = 1; i < a.size(); i++) {
                signString = signString + "," + hashF(a.get(i), b.get(i), row, num_of_rows);
            }
        } else {
            for (int i = 0; i < a.size(); i++) {
                signValues.add("" + hashF(a.get(i), b.get(i), row, num_of_rows));
            }
```

```
        }

        for (int i = 2; i < line.length; i++) {
            row = Integer.parseInt(line[i].trim());

            if (i == (line.length - 1)) {
                signString = "" + Math.min(Integer.parseInt(signValues.get(0).trim()), hashF(a.get(0), b.get(0),
row, num_of_rows));
                for (int k = 1; k < signValues.size(); k++) {
                    signString = signString + "," + Math.min(Integer.parseInt(signValues.get(k).trim()),
hashF(a.get(k), b.get(k), row, num_of_rows));
                }
            } else {
                for (int k = 0; k < signValues.size(); k++) {
                    if (Integer.parseInt(signValues.get(k).trim()) > hashF(a.get(k), b.get(k), row, num_of_rows))
{
                        signValues.set(k, "" + hashF(a.get(i), b.get(i), row, num_of_rows));
                    }
                }
            }


        }

        for (int i = 1; i <= num_of_columns; i++) {
            if (i < columnNumber) {
                String out_key = threshold + "," + i + ";" + columnNumber;
                output.collect(new Text(out_key), new Text(signString));
            } else if (i > columnNumber) {
                String out_key = threshold + "," + columnNumber + ";" + i;
                output.collect(new Text(out_key), new Text(signString));
            }
        }
    }

    public int hashF(int a, int b, int x, int N) {
        return ((a * x + b) % N);
    }
}
```

B.4 Η ΔΙΑΔΙΚΑΣΙΑ Reducer.

```
package minhashinghadoopv2;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
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 MinHashingReducer  extends MapReduceBase implements Reducer<Text, Text, Text, Text>
{

   @Override
   public void reduce(Text key, Iterator< Text > values, OutputCollector<Text, Text> output, Reporter
reporter)
          throws IOException {
       String[] item = new String[4];
       ArrayList<String> item_temp = new ArrayList<String>();
       String[] param=key.toString().trim().split(",");
       int threshold=Integer.parseInt(param[0]);
       int j=0;
       while (values.hasNext()) {
          Text temp=values.next();
          String str=temp.toString().trim();
          if((!str.equals(""))||str!=null){
          item[j] =str;
          item_temp.add(str);
          j++;
                }
       }
       String[] item1=item[0].trim().split(",");
       String[] item2=item[1].trim().split(",");
       int count_match=0;
       for(int i=0;i<item1.length;i++){
          if(item1[i].trim().equals(item2[i].trim())){
             count_match++;
          }
       }
       if((((double) count_match)/((double) item1.length))>=( ((double)threshold)/((double)100))){
          output.collect(key, new Text(""+(((double) count_match)/((double) item1.length))+" "+(
((double)threshold)/((double)100))));
       }


   }
}
```

B.5 ΟΔΕΥΣΗ ΑΡΧΕΙΩΝ.

```
package minhashinghadoopv2;

public interface Parameters {
//    public static final String LOCAL_PARAMETERS_FILE = "/home/
hduser/hadoop/stelmoul/metadata/parameters.txt";
   public static final String HDFS_PARAMETERS_FILE =
"/home/hduser/hadoop/stelmoul/metadata/parameters.txt";
//    public static final String INPUT_HDFS = "/home/ hduser/hadoop/stelmoul/Input";
//    public static final String OUTPUT_HDFS = "/home/ hduser/hadoop/stelmoul/Output";
   public static final String LOCAL_PARAMETERS_FILE =
"/home/hduser/hadoop/stelmoul/parameters.txt";
//    public static final String HDFS_PARAMETERS_FILE = "/user/hduser/metadata/parameters.txt";
   public static final String INPUT_HDFS = "/user/hduser/input";
   public static final String OUTPUT_HDFS = "/user/hduser/output";
}
```

ΠΑΡΑΡΤΗΜΑ Γ

ΠΙΝΑΚΕΣ

Γ.1 ΑΡΧΕΙΟ ΔΕΔΟΜΕΝΩΝ 100 ΕΓΓΡΑΦΩΝ.

Γ.1.1 Πίνακας αποτελεσμάτων εκτέλεσης σε Η/Υ.

	Elapsed Time			
Threshold	50 Hash Functions	100 Hash Functions	150 Hash Functions	200 Hash Functions
20%	38,088	36,234	37,997	38,548
30%	38,982	37,568	38,296	39,432
40%	37,289	38,244	37,731	39,549
50%	36,279	39,157	39,176	36,128
60%	39,274	38,315	36,314	39,268
70%	38,247	39,579	39,861	37,246
80%	38,286	38,269	39,597	38,567
90%	37,264	36,146	38,573	38,165
AVG	**37,964**	**37,939**	**38,443**	**38,363**

Πίνακας Π3.1

Στον πίνακα Π3.1 παρατίθενται τα αποτελέσματα που λάβαμε, τρέχοντας το πρόγραμμα με χρήση αρχείου (datafile)100 εγγραφών, δοκιμάζοντας threshold 20 – 90% και hash functions 50,

100, 150 και 200, σε οικειακό υπολογιστή. Στο κάτω μέρος του πίνακα, αναγράφεται η μέση τιμή του χρόνου ολοκλήρωσης κατά περίπτωση. Ο χρόνος μετράται σε δευτερόλεπτα (sec).

Γ.1.2 Πίνακες αποτελεσμάτων εκτέλεσης με 1 Reducer.

Γ.1.2.1Αρχείο 100 εγγραφών

	Elapsed Time			
Threshold	50 Hash Functions	100 Hash Functions	150 Hash Functions	200 Hash Functions
20%	35,596	33,550	34,232	35,693
30%	36,094	33,845	35,459	35,524
40%	33,594	34,146	33,688	35,954
50%	32,392	35,597	35,294	32,548
60%	36,031	35,808	33,013	35,061
70%	35,745	36,990	37,253	33,860
80%	34,805	34,477	36,664	35,710
90%	34,504	32,273	35,388	34,383
AVG	34,845	34,586	35,124	34,842

min time	32,273
max time	37,253

Πίνακας Π3.2

Γ.1.2.2Αρχείο 88.000 εγγραφών

	Elapsed Time			
Threshold	50 Hash Functions	100 Hash Functions	150 Hash Functions	200 Hash Functions
20%	46,568	48,564	47,125	46,341
30%	48,189	48,234	47,964	46,256
40%	47,256	47,511	46,557	48,142
50%	47,200	48,237	47,681	47,354
60%	46,268	46,237	46,883	46,987
70%	46,895	46,771	48,667	47,379
80%	46,593	47,006	47,356	47,865
90%	48,562	45,367	45,875	47,687
AVG	47,191	47,241	47,264	47,251

min time	45,367
max time	48,667

Πίνακας Π3.3

Ο χρόνος μεταξύ των δύο πινάκων διαφέρει ως προς το μέτρο και στο αρχείο των 100 εγγραφών μετριέται σε sec, ενώ στο αρχείο των 88.000 εγγραφών σε sec επί 10^3. Αυτό ισχύει και σε όλους τους παρακάτω πίνακες.

Γ.1.3 Πίνακες αποτελεσμάτων εκτέλεσης με 2 Reducers.

Γ.1.3.1Αρχείο 100 εγγραφών

	Elapsed Time - 1 Mapper			
Threshold	50 Hash Functions	100 Hash Functions	150 Hash Functions	200 Hash Functions
20%	20,824	19,795	20,710	21,059
30%	21,296	20,476	20,921	21,492
40%	20,324	20,829	20,550	21,572
50%	19,759	21,358	21,353	19,691
60%	21,439	20,948	19,808	21,387
70%	20,911	21,639	21,793	20,316
80%	20,883	20,858	21,632	21,069
90%	20,357	19,687	21,056	20,802
AVG	**20,724**	**20,699**	**20,978**	**20,924**

min time	19,687
max time	21,793

Πίνακας Π3.4

	Elapsed Time - 2 Mappers			
Threshold	50 Hash Functions	100 Hash Functions	150 Hash Functions	200 Hash Functions
20%	20,416	20,586	19,724	20,056
30%	21,722	19,880	20,511	22,567
40%	19,542	21,246	21,372	20,742
50%	20,549	22,213	22,420	20,873
60%	22,082	20,338	20,600	22,670
70%	20,302	22,288	21,158	19,535
80%	20,474	20,056	22,713	22,122
90%	20,968	20,277	20,246	20,394
AVG	**20,757**	**20,861**	**21,093**	**21,120**

min time	19,535
max time	22,713

Πίνακας Π3.5

	Elapsed Time - 3 Mappers			
Threshold	50 Hash Functions	100 Hash Functions	150 Hash Functions	200 Hash Functions
20%	21,865	18,328	22,367	22,112
30%	19,718	21,705	22,176	22,782
40%	22,153	19,837	19,571	20,545
50%	20,945	19,961	22,207	20,479
60%	20,614	22,833	18,512	20,369
70%	19,362	20,807	23,537	21,941
80%	21,301	19,678	19,846	19,876
90%	20,968	20,080	22,319	19,811
AVG	**20,866**	**20,404**	**21,317**	**20,989**

min time	18,328
max time	23,537

Πίνακας Π3.6

Γ.1.3.2Αρχείο 88.000 εγγραφών

	Elapsed Time - 1 Mapper			
Threshold	50 Hash Functions	100 Hash Functions	150 Hash Functions	200 Hash Functions
20%	27,242	28,653	28,511	27,341
30%	28,432	29,182	28,299	27,985
40%	28,590	28,982	28,400	28,885
50%	28,792	28,942	28,847	28,649
60%	27,529	27,049	28,130	28,662
70%	27,434	27,361	28,470	28,427
80%	27,956	28,439	27,940	28,240
90%	28,652	27,674	27,296	28,851
AVG	**28,078**	**28,285**	**28,236**	**28,380**

min time	27,049
max time	29,182

Πίνακας Π3.7

	Elapsed Time - 2 Mappers			
Threshold	50 Hash Functions	100 Hash Functions	150 Hash Functions	200 Hash Functions
20%	27,787	29,799	29,936	26,039
30%	27,874	28,332	27,744	29,384
40%	29,733	29,561	29,536	27,774
50%	27,685	27,829	30,289	27,028
60%	28,355	27,860	27,048	27,040
70%	28,257	26,564	29,324	29,564
80%	27,408	27,345	26,610	29,652
90%	29,511	28,504	28,387	28,285
AVG	**28,326**	**28,224**	**28,609**	**28,096**

min time	26,039
max time	30,289

Πίνακας Π3.8

	Elapsed Time - 3 Mappers			
Threshold	50 Hash Functions	100 Hash Functions	150 Hash Functions	200 Hash Functions
20%	25,945	26,530	26,399	26,039
30%	26,325	27,530	29,997	29,664
40%	31,163	30,431	29,820	30,329
50%	30,520	30,968	30,001	27,547
60%	28,631	24,815	30,099	27,297
70%	25,401	28,455	26,361	30,702
80%	28,515	26,829	25,633	26,642
90%	27,817	28,227	28,933	30,293
AVG	**28,040**	**27,973**	**28,405**	**28,564**

min time	24,815
max time	31,163

Πίνακας Π3.9

Γ.1.4 Πίνακες αποτελεσμάτων εκτέλεσης με 4 Reducers.

Γ.1.4.1Αρχείο 100 εγγραφών.

	Elapsed Time - 1 Mapper - 100 data			
Threshold	50 Hash Functions	100 Hash Functions	150 Hash Functions	200 Hash Functions
20%	12,494	11,976	12,840	12,740
30%	12,884	12,695	12,657	13,325
40%	12,601	13,018	12,844	13,267
50%	12,349	13,135	13,239	12,209
60%	13,078	12,569	12,182	13,367
70%	12,546	12,983	13,076	12,494
80%	12,843	12,932	13,087	12,747
90%	12,316	12,304	12,844	12,897
AVG	**12,639**	**12,702**	**12,846**	**12,881**

min time	11,976
max time	13,367

Πίνακας Π3.10

	Elapsed Time - 2 Mappers - 100 data			
Threshold	50 Hash Functions	100 Hash Functions	150 Hash Functions	200 Hash Functions
20%	12,249	12,455	12,229	12,134
30%	13,142	12,326	12,409	13,991
40%	12,116	13,279	13,357	12,757
50%	12,843	13,661	13,901	12,941
60%	13,470	12,203	12,669	14,169
70%	12,181	13,373	12,695	12,014
80%	12,591	12,435	13,742	13,384
90%	12,686	12,673	12,350	12,644
AVG	**12,660**	**12,800**	**12,919**	**13,004**

min time	12,014
max time	14,169

Πίνακας Π3.11

	Elapsed Time - 3 Mappers - 100 data			
Threshold	50 Hash Functions	100 Hash Functions	150 Hash Functions	200 Hash Functions
20%	13,119	11,089	13,867	13,377
30%	11,930	13,457	13,417	14,125
40%	13,735	12,398	12,232	12,635
50%	13,090	12,276	13,768	12,697
60%	12,575	13,700	11,385	12,730
70%	11,617	12,484	14,122	13,494
80%	13,100	12,200	12,007	12,025
90%	12,686	12,550	13,615	12,283
AVG	**12,731**	**12,519**	**13,052**	**12,921**

Total Data Avg	12,806

min time	11,089
max time	14,125

Πίνακας Π3.12

	50 Hash Function - 100 data		
Threshold	Time - 1 mapper	Time - 2 mappers	Time - 3 mappers
20%	12,494	12,249	13,119
30%	12,884	13,142	11,930
40%	12,601	12,116	13,735
50%	12,349	12,843	13,090
60%	13,078	13,470	12,575
70%	12,546	12,181	11,617
80%	12,843	12,591	13,100
90%	12,316	12,686	12,686
AVG	**12,639**	**12,660**	**12,731**

Πίνακας Π3.13

Γ.1.4.2Αρχείο 88.000 εγγραφών.

	Elapsed Time - 1 Mapper - 88.000 data			
Threshold	50 Hash Functions	100 Hash Functions	150 Hash Functions	200 Hash Functions
20%	16,754	18,051	17,534	17,225
30%	17,912	18,530	17,970	17,491
40%	18,155	18,114	17,892	18,198
50%	17,851	17,655	18,029	18,192
60%	16,793	16,500	17,159	17,914
70%	17,146	17,237	17,509	17,483
80%	17,193	18,059	17,323	17,791
90%	16,475	15,912	15,695	16,589
AVG	**17,285**	**17,507**	**17,389**	**17,610**

min time	15,695
max time	18,530

Πίνακας Π3.14

	Elapsed Time - 2 Mappers - 88.000 data			
Threshold	50 Hash Functions	100 Hash Functions	150 Hash Functions	200 Hash Functions
20%	17,089	18,773	18,411	16,405
30%	17,561	17,991	17,617	18,365
40%	18,881	18,476	18,608	17,498
50%	17,164	16,976	18,931	17,162
60%	17,297	16,995	16,499	16,900
70%	17,660	16,735	18,034	18,182
80%	16,856	17,364	16,498	18,681
90%	16,969	16,390	16,323	16,264
AVG	**17,435**	**17,462**	**17,615**	**17,432**

min time	16,264
max time	18,931

Πίνακας Π3.15

	Elapsed Time - 3 Mappers - 88.000 data			
Threshold	50 Hash Functions	100 Hash Functions	150 Hash Functions	200 Hash Functions
20%	15,956	16,714	16,235	16,405
30%	16,585	17,481	19,048	18,540
40%	19,788	19,019	18,786	19,108
50%	18,922	18,891	18,751	17,493
60%	17,465	15,137	18,360	17,061
70%	15,876	17,927	16,212	18,881
80%	17,537	17,036	15,892	16,784
90%	15,995	16,231	16,637	17,419
AVG	**17,265**	**17,305**	**17,490**	**17,711**

min time	15,137
max time	19,788

Πίνακας Π3.16

	50 Hash Function - 88.000 data		
Threshold	Time - 1 mapper	Time - 2 mappers	Time - 3 mappers
20%	16,754	17,089	15,956
30%	17,912	17,561	16,585
40%	18,155	18,881	19,788
50%	17,851	17,164	18,922
60%	16,793	17,297	17,465
70%	17,146	17,660	15,876
80%	17,193	16,856	17,537
90%	16,475	16,969	15,995
AVG	**17,285**	**17,435**	**17,265**

Πίνακας Π3.17