ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

# ΕΡΓΑΣΙΑ ΕΞΑΜΗΝΟΥ

Θέμα: «Κατανεμημένη Ανάκτηση Πληροφορίας»


## Ομάδα:

Κυριακού Ανδρόνικος 5806
Ντενέζος Παναγιώτης 5853

# Περιεχόμενα

# Εισαγωγή

Στα πλαίσια της εργασίας για το μάθημα “Ανάκτηση Πληροφορίας”, θα γίνει μια προσέγγιση πάνω στην επεξεργασία κειμένων μεγάλου όγκου δεδομένων (big data) με χρήση του MapReduce. Το MapReduce είναι ένα μοντέλο προγραμματισμού για την έκφραση κατανεμημένων υπολογισμών τεραστίου όγκου δεδομένων σε ένα πλαίσιο συστοιχιών διακομιστών. Αρχικά, αναπτύχθηκε από τη Google και βασίστηκε σε γνωστές αρχές παράλληλης και κατανεμημένης επεξεργασίας. Από τότε το MapReduce γνώρισε μεγάλη ανταπόκριση από την κοινότητα μέσω μιας εφαρμογής ανοιχτού κώδικα με το όνομα Hadoop. Η ανάπτυξη του Hadoop έγινε πρώτα από τη Yahoo και στη συνέχεια μετατράπηκε σε project του Apache. Σήμερα, όλο αυτό έχει μετατραπεί σε ένα ζωντανό οικοσύστημα λογισμικού με σημαντικο ενδιαφέρον και απόκριση στον κλάδο της Πληροφορικής, τόσο σε επιχειρήσεις όσο και στον ακαδημαϊκό κόσμο.

Στις μέρες μας, υπάρχει ένας μεγάλος αριθμός εταιριών, ερευνητικών κέντρων κλπ, που καλούνται καθημερινά να αντιμετωπίσουν το ζήτημα της επεξεργασίας μεγάλου όγκου δεδομένων που προέρχονται από τα συστήματα του πραγματικού κόσμου. Επιπλέον, σε ένα ευρύ φάσμα εφαρμογών επεξεργασίας κειμένου, περισσότερα δεδομένα μεταφράζονται σε πιο αποτελεσματικούς αλγόριθμους, και συνεπώς έχει νόημα να εκμεταλλευτούμε αυτά τα άφθονα δεδομένα που μας περιβάλλουν. Κάθε οργανισμός που ασχολείται γύρω από τη συλλογή, την ανάλυση, την παρακολούθηση, το φιλτράρισμα, την αναζήτηση δεδομένων καλείται να αντιμετωπίσει το πρόβλημα του μεγάλου όγκου δεδομένων.

Ένα παράδειγμα, ώστε να γίνει κατανοητός ο όγκος των δεδομένων στον οποίο αναφερόμαστε είναι αυτό της Google. Πιο συγκεκριμένα, ενώ η εταιρία επεξεργαζότανε 100 TB δεδομένων την ημέρα με το MapReduce το 2004, μόλις 4 χρόνια μετά, δηλαδή το 2008 η εταιρία επεξεργαζόταν την ημέρα 20 PB δεδομένων. Με λίγα λόγια, μέσα σε 4 χρόνια το μέγεθος των δεδομένων διακοσαπλασιάστηκε. Από αυτό καταλαβαίνουμε πόσο επιτακτική είναι η ανάγκη για αποτελεσματική και γρήγορη ανάλυση των δεδομένων.

Ακόμα ένα παράδειγμα αφορά την μοντελοποίηση των γλωσσών του ιστού. Ένα γλωσσικό μοντέλο είναι μια κατανομή πιθανότητας που χαρακτηρίζει την πιθανότητα παρατήρησης μιας συγκεκριμένης ακολουθίας λέξεων, η οποία εκτιμάται από ένα μεγάλο σύνολο κειμένων. Είναι χρήσιμα σε μια ποικιλία εφαρμογών, όπως η αναγνώριση ομιλίας και η αυτόματη μετάφραση. Δεδομένου ότι υπάρχουν πολλές πιθανές συμβολοσειρές και οι πιθανότητες πρέπει να ανατεθούν σε όλες, η μοντελοποίηση γλωσσών είναι πιο περίπλοκη υπόθεση από ότι απλά να παρακολου-

θείται ο αριθμός εμφανίσεων μιας συμβολοσειράς. Είναι πολύ δύσκολο να βρεθεί κάποιος αριθμός πιθανών συμβολοσειρών, ακόμη και με πολλά στοιχεία εκπαίδευσης. Συνεπώς για να αποδίδουν καλύτερα τα γλωσσικά μοντέλα και να μπορούν να εκπαιδευτούν όσο καλύτερα γίνεται, χρειάζονται περισσότερα δεδομένα.

Τώρα που παρουσιάστηκε ο λόγος, θα ασχοληθούμε και με τον τρόπο που επιτυγχάνεται αυτό. Η επεξεργασία δεδομένων μεγάλου όγκου είναι πέρα από την ικανότητα οποιασδήποτε μηχανής και απαιτεί συστοιχίες, πράγμα που σημαίνει ότι τα μεγάλα προβλήματα δεδομένων βασικά αφορούν την οργάνωση υπολογισμών σε δεκάδες, εκατοντάδες ή και χιλιάδες μηχανές. Αυτό είναι το πρόβλημα που λύνει το MapReduce, και στη συνέχεια θα εξερευνήσουμε τον τρόπο που το επιτυγχάνει.

## Υπολογισμοί στο Cloud

Σε αυτήν την υποενότητα θα προσπαθήσουμε να εξηγήσουμε τι είναι το cloud computing και πώς σχετίζεται με το MapReduce και την επεξεργασία μεγάλου όγκου δεδομένων.

Σε πιο επιφανειακό επίπεδο, μπορούμε να πούμε πως, οτιδήποτε κάποτε ονομαζόταν εφαρμογή ιστού έχει αναδιαμορφωθεί για να γίνει εφαρμογή cloud. Στην πραγματικότητα, ό,τι εκτελείται μέσα σε ένα πρόγραμμα περιήγησης που συλλέγει και αποθηκεύει περιεχόμενο, το οποίο δημιουργείται από το χρήστη, είναι πλέον μια εφαρμογή του cloud computing. Η συσσώρευση τεράστιων ποσοτήτων δεδομένων δημιουργεί μεγάλα προβλήματα. Για παράδειγμα, ένας ιστότοπος κοινωνικής δικτύωσης αναλύει το τεράστιο γράφημα των χρηστών του για να συστήσει νέες συνδέσεις. Επιπλέον, μια υπηρεσία ηλεκτρονικού ταχυδρομείου αναλύει τα μηνύματα και τη συμπεριφορά των χρηστών για να βελτιστοποιήσει την επιλογή και την τοποθέτηση διαφημίσεων.

Μια άλλη σημαντική πτυχή του cloud computing είναι η ενοικίαση υπολογιστικών πόρων (utility computing). Η ιδέα φτάνει πίσω στις μέρες των μηχανημάτων καταμερισμού χρόνου, και στην πραγματικότητα δεν διαφέρει πολύ από αυτή την αρχική μορφή πληροφορικής. Σύμφωνα με αυτό το μοντέλο, σε ένα χρήστη cloud μπορεί να παρέχεται δυναμικά ποσότητα υπολογιστικών πόρων από έναν πάροχο κατόπιν αιτήματος και να πληρώνει μόνο για αυτό που καταναλώνει. Η κατανάλωση πόρων μετριέται σε κάποια ισοδύναμη ώρα μηχανής και οι χρήστες χρεώνονται αντίστοιχα.

Το μοντέλο προγραμματισμού MapReduce είναι μια ισχυρή αφαίρεση που χωρίζει το πως από το τι κατά την επεξεργασία δεδομένων μεγάλου όγκου.

## Μεγάλα Προβλήματα – Μεγάλες Λύσεις

Η αντιμετώπιση μεγάλων προβλημάτων δεδομένων απαιτεί μια ξεχωριστή προσέγγιση, η οποία ενίοτε αντιβαίνει τα παραδοσιακά μοντέλα υπολογιστών. Σε αυτή την ενότητα, συζητούμε μια σειρά από ιδέες πίσω από το MapReduce. Όλες αυτές οι ιδέες έχουν συζητηθεί εδώ και αρκετό καιρό στη βιβλιογραφία των υπολογιστών (μερικές εδώ και δεκαετίες) και το MapReduce σίγουρα δεν είναι το πρώτο που τις υιοθετεί.

Για φόρτο εργασίας μεγάλης κλίμακας δεδομένων, προτιμάται ένας μεγάλος αριθμός low-end διακομιστών (δηλαδή η προσέγγιση κλιμάκωσης "προς την ποσότητα") έναντι ενός μικρού αριθμού high-end διακομιστών (δηλαδή της προσέγγισης κλιμάκωσης "προς την ποιότητα"). Η τελευταία προσέγγιση της αγοράς συμμετρικών μηχανών πολλαπλής επεξεργασίας (SMP) με μεγάλο αριθμό υποδοχών επεξεργαστών (δεκάδες, ακόμη και εκατοντάδες) και μεγάλης ποσότητας μνήμης (εκατοντάδες ή και χιλιάδες gigabytes) δεν είναι οικονομικά αποδοτική. Για παράδειγμα, μια μηχανή με διπλάσιο αριθμό επεξεργαστών είναι συχνά σημαντικά περισσότερο από δύο φορές πιο ακριβή, συνεπώς η αύξηση της τιμής δεν είναι γραμμική.

Δεδομένου ότι ο φόρτος εργασίας σήμερα είναι πέρα από την ικανότητα οποιασδήποτε μηχανής (ανεξάρτητα από το πόσο ισχυρή είναι), η σύγκριση είναι ακριβέστερη μεταξύ ενός μικρότερου συνόλου μηχανημάτων υψηλών προδιαγραφών και ενός μεγαλύτερου συνόλου μηχανημάτων χαμηλών προδιαγραφών. Οι Barroso και Hölzle συνέκριναν αυτές τις δύο προσεγγίσεις και κατέληξαν στο συμπέρασμα ότι μια ομάδα διακομιστών χαμηλών προδιαγραφών προσεγγίζει την απόδοση ενός ισοδύναμου συνόλου διακομιστών υψηλού επιπέδου, οπότε το μικρό χάσμα απόδοσης δεν επαρκεί για να δικαιολογήσει την διαφορά της τιμής των διακομιστών υψηλού επιπέδου. Συνεπώς οι περισσότερες υπάρχουσες υλοποιήσεις του μοντέλου προγραμματισμού MapReduce σχεδιάζονται γύρω από σύνολα διακομιστών προϊόντων χαμηλού επιπέδου.

Η αποδοτικότητα του κέντρου δεδομένων συνήθως διαχωρίζεται σε τρία χωριστά στοιχεία, τα οποία μπορούν να μετρούνται και να βελτιστοποιούνται ανεξάρτητα. Το πρώτο στοιχείο υπολογίζει κατά πόσο η ισχύς ενός κτιρίου φτάνει πραγματικά στον εξοπλισμό πληροφορικής και αντίστοιχα, πόση ενέργεια χάνεται από τα μηχανικά συστήματα του κτιρίου (π.χ. ψύξη, αερισμός) και την ηλεκτρική υποδομή (π.χ., αναποτελεσματικότητα κατανομής ισχύος). Το δεύτερο στοιχείο μετρά πόση απώλεια εισερχόμενης ισχύος ενός διακομιστή χάνεται από την τροφοδοσία, τους ανεμιστήρες ψύξης κλπ. Το τρίτο στοιχείο καταγράφει πόση ισχύς που παραδίδεται στα εξαρτή-

ματα υπολογιστών (επεξεργαστής, μνήμη RAM, δίσκος κ.λπ.) χρησιμοποιείται πραγματικά για να πραγματοποιήσει χρήσιμους υπολογισμούς.

Παράλληλα, οι αποτυχίες στα κέντρα δεδομένων δεν είναι μόνο αναπόφευκτες, αλλά συνηθισμένες. Ένας απλός υπολογισμός αρκεί για να αποδειχθεί. Ας υποθέσουμε ότι ένα σύμπλεγμα κατασκευάζεται από αξιόπιστες μηχανές με μέση χρονική περίοδο μεταξύ αποτυχιών (MTBF) 1000 ημερών (περίπου τρία χρόνια). Ακόμη και με αυτούς τους αξιόπιστους διακομιστές, ένα σύμπλεγμα 10.000 εξυπηρετητών θα εξακολουθούσε να αντιμετωπίζει περίπου 10 αποτυχίες την ημέρα. Έστω ότι ένα MTBF 10.000 ημερών (περίπου τριάντα χρόνια) ήταν εφικτό σε ρεαλιστικά κόστη (κάτι που είναι απίθανο). Ακόμα και τότε, ένα σύμπλεγμα 10.000 εξυπηρετητών θα αντιμετώπιζε ακόμα μία αποτυχία καθημερινά. Αυτό σημαίνει ότι οποιαδήποτε υπηρεσία μεγάλης κλίμακας που διανέμεται σε ένα μεγάλο σύνολο διακομιστών πρέπει να αντιμετωπίσει τις αποτυχίες υλικού ως εγγενή πτυχή της λειτουργίας της.

Μια καλά σχεδιασμένη και ανεκτική σε σφάλματα υπηρεσία πρέπει να αντιμετωπίζει τις αποτυχίες μέχρι ένα σημείο χωρίς να επηρεάσει την ποιότητα της υπηρεσίας, δηλαδή οι αποτυχίες δεν πρέπει να οδηγήσουν σε ασυνέπειες. Καθώς οι διακομιστές βγαίνουν εκτός λειτουργίας, άλλοι κόμβοι θα πρέπει να εισέρχονται απρόσκοπτα στο χειρισμό του φορτίου και η συνολική απόδοση θα πρέπει να υποβαθμίζεται με μικρό ρυθμό. Εξίσου σημαντικό, ένας διακομιστής αφού επιδιορθωθεί, θα πρέπει να είναι σε θέση να επανασυνδεθεί άψογα με την υπηρεσία χωρίς χειροκίνητη αναδιαμόρφωση από τον διαχειριστή. Οι υλοποιήσεις του μοντέλου προγραμματισμού MapReduce είναι σε θέση να αντιμετωπίσουν με επιτυχία τις αποτυχίες μέσω ενός αριθμού μηχανισμών όπως η αυτόματη επανεκκίνηση των εργασιών σε διαφορετικούς κόμβους.

Στις παραδοσιακές εφαρμογές υπολογιστών υψηλής απόδοσης (HPC) (π.χ. για κλιματολογικές ή πυρηνικές προσομοιώσεις), είναι συνηθισμένο για έναν υπερυπολογιστή να έχει ξεχωριστούς κόμβους επεξεργασίας και κόμβους αποθήκευσης συνδεδεμένους μεταξύ τους με διασύνδεση υψηλής χωρητικότητας. Πολλά φορτία δεδομένων μεγάλου όγκου δεν απαιτούν πολύ επεξεργαστική ισχύ, πράγμα που σημαίνει ότι ο διαχωρισμός των υπολογισμών και της αποθήκευσης δημιουργεί ένα εμπόδιο στο δίκτυο. Ως εναλλακτική λύση για τη μετακίνηση δεδομένων, είναι πιο αποτελεσματικό να μετακινηθεί η επεξεργασία. Δηλαδή, το MapReduce ορίζει μια αρχιτεκτονική όπου οι επεξεργαστές και ο αποθηκευτικός χώρος (δίσκος) βρίσκονται στο ίδιο σημείο χωροχρονικά. Σε μια τέτοια εγκατάσταση, μπορούμε να επωφεληθούμε από την τοποθεσία δεδομένων εκτελώντας κώδικα στον επεξεργαστή που είναι άμεσα συνδεδεμένος με το

μπλοκ δεδομένων που χρειαζόμαστε. Το κατανεμημένο περιβάλλον είναι υπεύθυνο για τη διαχείριση των δεδομένων με τα οποία λειτουργεί το MapReduce.

Η επεξεργασία δεδομένων μεγάλου όγκου σημαίνει ότι τα σχετικά σύνολα δεδομένων είναι πολύ μεγάλα για να χωρέσουν στη κύρια μνήμη και πρέπει να χρησιμοποιηθεί η δευτερεύουσα. Οι χρόνοι αναζήτησης για τυχαία πρόσβαση στο δίσκο περιορίζονται βασικά από τη μηχανική φύση των συσκευών (πχ κεφαλές ανάγνωσης και εγγραφής). Ως αποτέλεσμα, είναι επιθυμητό να αποφεύγεται η τυχαία πρόσβαση σε δεδομένα και, αντίθετα, να οργανώνονται υπολογισμοί έτσι ώστε τα δεδομένα να επεξεργάζονται διαδοχικά.

Το MapReduce διατηρεί ένα διαχωρισμό των υπολογισμών που πρέπει να εκτελεστούν και του πώς αυτοί οι υπολογισμοί εκτελούνται στην πραγματικότητα σε ένα σύμπλεγμα μηχανών. Το πρώτο είναι υπό τον έλεγχο του προγραμματιστή, ενώ το δεύτερο είναι αποκλειστικά η ευθύνη του πλαισίου εκτέλεσης. Το πλεονέκτημα είναι ότι το πλαίσιο εκτέλεσης πρέπει να σχεδιαστεί μόνο μία φορά και να επαληθευτεί για την ορθότητα του, ενώ στη συνέχεια, ο προγραμματιστής εκφράζοντας υπολογισμούς στο μοντέλο προγραμματισμού, έχει την εγγύηση ότι ο κώδικας θα συμπεριφέρεται όπως αναμένεται. Το αποτέλεσμα είναι ότι ο προγραμματιστής δεν χρειάζεται να ανησυχεί για τις λεπτομέρειες του συστήματος (π.χ. σε αλγόριθμο ή σχεδιασμό εφαρμογών).

## Τα βασικά της μεθόδου MapReduce

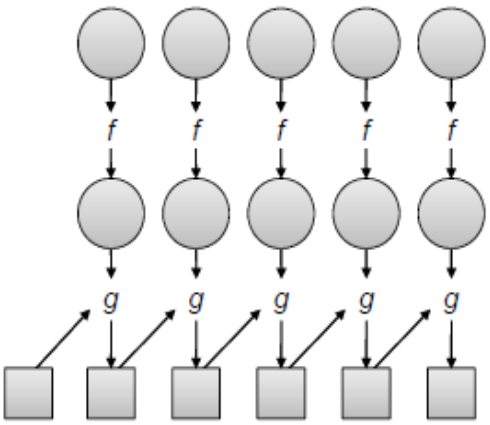
Η μόνη εφικτή προσέγγιση για την αντιμετώπιση μεγάλων προβλημάτων δεδομένων σήμερα είναι η τεχνική του «Διαίρει και βασίλευε» μια θεμελιώδη ιδέα στην επιστήμη των υπολογιστών. Η βασική ιδέα είναι να χωριστεί ένα μεγάλο πρόβλημα σε μικρότερα δευτερεύοντα προβλήματα. Η λύση των επιμέρους προβλημάτων συνδυάζεται για να αποδώσει το αποτέλεσμα.

Το MapReduce παρέχει μια αφαίρεση που κρύβει πολλές λεπτομέρειες σε επίπεδο συστήματος από τον προγραμματιστή. Επομένως, ένας προγραμματιστής μπορεί να επικεντρωθεί σε τι υπολογισμοί πρέπει να εκτελεστούν, σε αντίθεση με τον τρόπο με τον οποίο οι υπολογισμοί αυτοί πραγματοποιούνται ή πώς να αποκτήσουν τα δεδομένα οι διαδικασίες που εξαρτώνται από αυτά. Ωστόσο, η οργάνωση και ο συντονισμός μεγάλων ποσοτήτων υπολογισμού είναι μόνο μέρος της πρόκλησης. Η επεξεργασία μεγάλων δεδομένων από τον ορισμό της απαιτεί τη μεταφορά δεδομένων και κώδικα για υπολογισμό. Όπως αναφέραμε παραπάνω, αντί να μεταφέρουμε μεγάλα ποσά δεδομένων, είναι πολύ πιο αποτελεσματικό, εάν είναι δυνατόν, να μεταφέρουμε τον

κώδικα στα δεδομένα. Αυτό επιτυγχάνεται λειτουργικά με τη διάδοση δεδομένων σε όλους τους τοπικούς δίσκους κόμβων σε ένα σύμπλεγμα και την εκτέλεση διαδικασιών σε κόμβους που κατέχουν τα δεδομένα. Το πολύπλοκο έργο διαχείρισης της αποθήκευσης σε ένα τέτοιο περιβάλλον επεξεργασίας συνήθως αντιμετωπίζεται από ένα κατανεμημένο σύστημα που βρίσκεται κάτω από το MapReduce.

## Οι ρίζες του MapReduce

Το MapReduce έχει τις ρίζες του στον λειτουργικό προγραμματισμό. Ένα βασικό χαρακτηριστικό των γλωσσών αυτών είναι η έννοια των ανώτερων λειτουργιών ή λειτουργιών που μπορούν να δεχθούν άλλες λειτουργίες ως ορίσματα. Δύο κοινές ενσωματωμένες λειτουργίες υψηλότερης τάξης είναι η map και η fold, που απεικονίζονται στο Σχήμα 1. Με δεδομένη μια λίστα, η map λαμβάνει ως παράμετρο μια συνάρτηση f και την εφαρμόζει σε όλα τα στοιχεία της λίστας. Λαμβάνοντας μια λίστα, η fold παίρνει ως παράμετρο μια συνάρτηση g και μια αρχική τιμή. Η g εφαρμόζεται αρχικά στην αρχική τιμή και στο πρώτο στοιχείο της λίστας, το αποτέλεσμα της οποίας αποθηκεύεται σε μια ενδιάμεση μεταβλητή. Αυτή η ενδιάμεση μεταβλητή και το επόμενο στοιχείο στη λίστα χρησιμεύουν ως οι παράμετροι σε μια δεύτερη εφαρμογή του g, τα αποτελέσματα της οποίας αποθηκεύονται σε επόμενη ενδιάμεση μεταβλητή. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να καταναλωθούν όλα τα στοιχεία της λίστας. Η fold τότε επιστρέφει την τιμής της τελικής ενδιάμεσης μεταβλητής. Συνήθως, η map και η fold χρησιμοποιούνται σε συνδυασμό.




Σχήμα 1: Εικονογράφηση map και fold, δύο λειτουργίες υψηλότερης τάξης που χρησιμοποιούνται συνήθως μαζί στον λειτουργικό προγραμματισμό: η map παίρνει μια συνάρτηση f και την εφαρμόζει σε κάθε στοιχείο μιας λίστας, ενώ η fold εφαρμόζει επαναληπτικά μια λειτουργία g για να συγκεντρώσει τα αποτελέσματα.

Το παραπάνω αποτελεί μια αφηρημένη προσέγγιση του MapReduce. Η φάση του map στο MapReduce αντιστοιχεί στη λειτουργία της map σε λειτουργικό προγραμματισμό, ενώ η φάση reduce στο MapReduce αντιστοιχεί στην λειτουργία της fold στον λειτουργικό προγραμματισμό.

Το MapReduce κωδικοποιεί μια γενική προσέγγιση για την επεξεργασία μεγάλων συνόλων δεδομένων που αποτελείται από δύο στάδια: Στο πρώτο στάδιο, ένας υπολογισμός που καθορίζεται από το χρήστη εφαρμόζεται σε όλα τα αρχεία δεδομένων του συνόλου δεδομένων, ενώ παράλληλα αποδίδει την ενδιάμεσο αποτέλεσμα, το οποίο στη συνέχεια συγκεντρώνεται με άλλο υπολογισμό που έχει καθοριστεί από τον χρήστη. Ο προγραμματιστής σχεδιάζει αυτούς τους δύο τύπους υπολογισμών και το πλαίσιο εκτέλεσης ασχολείται με την πραγματική επεξεργασία τους.

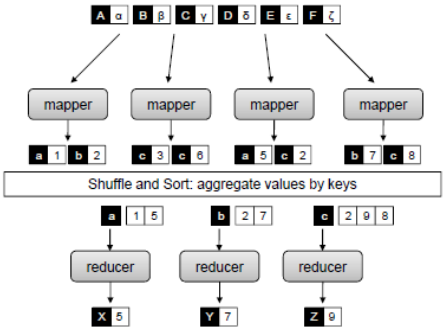
Το MapReduce μπορεί να αναφερθεί ως τρεις ξεχωριστές αλλά σχετικές έννοιες. Πρώτον, το MapReduce είναι ένα μοντέλο προγραμματισμού, του οποίου η έννοια συζητήθηκε παραπάνω. Δεύτερον, το MapReduce μπορεί να αναφερθεί στο πλαίσιο εκτέλεσης (δηλαδή το runtime) που συντονίζει την εκτέλεση των προγραμμάτων που γράφονται με αυτό το συγκεκριμένο τρόπο. Τέλος, το MapReduce μπορεί να αναφερθεί στην υλοποίηση λογισμικού του μοντέλου προγραμματισμού και του πλαισίου εκτέλεσης: παράδειγμα, την ιδιόκτητη υλοποίηση της Google έναντι της εφαρμογής Hadoop ανοιχτού κώδικα στην Java.

## Mappers και Reducers

Τα ζεύγη κλειδιών-τιμών (key-values) αποτελούν τη βασική δομή δεδομένων στο MapReduce. Τα κλειδιά και οι τιμές μπορεί να είναι ακέραιοι, αριθμοί κινητής υποδιαστολής, συμβολοσειρές και ακατέργαστα bytes ή μπορούν να είναι αυθαίρετα σύνθετες δομές (λίστες, πλειάδες κλπ.). Στο MapReduce, ο προ-γραμματιστής ορίζει έναν Mapper και έναν Reducer ως εξής $map\colon (k_1, u_1) \rightarrow [(k_2, u_2)]$ και $reduce\colon (k_2, [u_2]) \rightarrow [(k_3, u_3)]$

Η είσοδος σε μια εργασία MapReduce ξεκινά με δεδομένα αποθηκευμένα στο κατανεμημένο σύστημα αρχείων. Ο mapper εφαρμόζεται σε κάθε ζεύγος κλειδιού-τιμής εισόδου για να παράγει έναν αυθαίρετο αριθμό ζευγών ενδιάμεσων κλειδιών-τιμών. Ο reducer εφαρμόζεται σε όλες τις τιμές που σχετίζονται με το ίδιο ενδιάμεσο κλειδί για τη δημιουργία ζευγών εξόδου κλειδιού-τιμής. Οι ενδιάμεσες τιμές φθάνουν σε κάθε reducer κατά σειρά, ταξινομημένες με βάση το κλειδί. Δεν υπάρχει εγγύηση για τη διασύνδεση των τιμών μεταξύ διαφορετικών reducer. Τα ζεύγη από κάθε reducer γράφονται πίσω στο κατανεμημένο σύστημα αρχείων, ενώ τα ενδιάμεσα

ζευγάρια κλειδιού-τιμής είναι παροδικά και δεν διατηρούνται. Η έξοδος καταλήγει σε r αρχεία στο κατανεμημένο σύστημα αρχείων, όπου r είναι ο αριθμός των reducers. Κατά το μεγαλύτερο μέρος, δεν υπάρχει λόγος να αποθηκευτούν τα αποτελέσματα των reducers, δεδομένου ότι τα r αρχεία συχνά χρησιμεύουν ως είσοδος σε μια επόμενη εργασία MapReduce. Το Σχήμα 2 απεικονίζει αυτή τη δομή επεξεργασίας δύο σταδίων.




Σχήμα 2: Απλοποιημένη απεικόνιση του MapReduce. Οι mappers εφαρμόζονται σε όλα τα ζεύγη εισόδου κλειδιού-τιμής, τα οποία παράγουν έναν αυθαίρετο αριθμό ζευγών ενδιάμεσων κλειδιών-τιμών. Οι reducers εφαρμόζονται σε όλες τις τιμές που σχετίζονται με το ίδιο κλειδί. Μεταξύ των φάσεων του map και του reduce βρίσκεται ένα στάδιο που περιλαμβάνει ένα μεγάλο κατανεμημένο, ταξινομημένο και ομαδοποιημένο σύνολο κλειδιών-τιμών.

Υπάρχουν κάποιες διαφορές μεταξύ της υλοποίησης του MapReduce του Hadoop και της Google. Στο Hadoop, ο reducer παρουσιάζεται με ένα κλειδί και έναν iterator πάνω από όλες τις τιμές που σχετίζονται με το συγκεκριμένο κλειδί. Οι τιμές προσπελαύνονται με αυθαίρετη σειρά. Η υλοποίησή της Google επιτρέπει στον προγραμματιστή να καθορίζει ένα δευτερεύον κλειδί ταξινόμησης για την παραγγελία των τιμών, οπότε οι τιμές που σχετίζονται με κάθε τιμή θα εμφανίζονται στον κώδικα του reducer του προγραμματιστή με ταξινομημένη σειρά. Αργότερα θα δειχθεί τρόπος για το πώς να ξεπεραστεί αυτός ο περιορισμός στο Hadoop για να εκτελέσει δευτερεύουσα ταξινόμηση. Μια άλλη διαφορά είναι ότι στην υλοποίηση της Google ο προγραμματιστής δεν επιτρέπεται να αλλάξει το κλειδί του reducer. Δηλαδή, η τιμή της εξόδου του reducer πρέπει να είναι ακριβώς η ίδια με την τιμή της εισόδου του reducer.

Οι mappers και οι reducers μπορούν να εκτελέσουν αυθαίρετους υπολογισμούς στις εισόδους τους. Γενικά, οι mappers μπορούν να εκπέμπουν έναν αυθαίρετο αριθμό ζευγών ενδιάμεσων κλειδιών-τιμών και δεν χρειάζεται να είναι του ίδιου τύπου με τα ζεύγη εισόδου κλειδιού-τιμής. Ομοίως, οι reducers μπορούν να εκπέμπουν έναν αυθαίρετο αριθμό τελικών ζευγών κλειδιού-τιμής και μπορούν να διαφέρουν σε τύπο από τα ενδιάμεσα ζεύγη κλειδιών-τιμών.

Δεδομένου ότι πολλοί mappers και reducers λειτουργούν παράλληλα και το σύστημα κατανεμημένων αρχείων είναι ένας κοινόχρηστος πόρος, πρέπει να ληφθεί μέριμνα, ώστε οι εν λόγω λειτουργίες να αποφεύγουν τις ταυτόχρονες εγγραφές σε αυτό. Μια στρατηγική είναι να γραφτεί ένα προσωρινό αρχείο που μετονομάζεται μετά από την επιτυχή ολοκλήρωση του mapper ή του reducer.

Συνήθως, η είσοδος σε μια εργασία MapReduce προέρχεται από δεδομένα που είναι αποθηκευμένα στο κατανεμημένο σύστημα αρχείων και η έξοδος γράφεται σε αυτό. Βέβαια, υπάρχουν περιπτώσεις όπου οι εργασίες του MapReduce ενδέχεται να μην δέχονται καθόλου εισόδους (π.χ. ο υπολογισμός του π) ή μπορούν να δέχονται μόνο ένα μικρό όγκο δεδομένων.

## Το πλαίσιο εκτέλεσης

Μία από τις πιο σημαντικές ιδέες πίσω από το MapReduce είναι ο διαχωρισμός των κατανεμημένων επεξεργασιών. Ένα πρόγραμμα MapReduce, που αναφέρεται ως δουλειά, αποτελείται από κώδικα για τους mappers και τους reducers. Ο προγραμματιστής υποβάλλει την εργασία στον κόμβο ενός συμπλέγματος (στο Hadoop, αυτό ονομάζεται jobtracker) και το πλαίσιο εκτέλεσης (runtime) φροντίζει για όλα τα άλλα: διαχειρίζεται με διαφάνεια όλες τις άλλες πτυχές της κατανεμημένης εκτέλεσης κώδικα, σε ομάδες που κυμαίνονται από έναν μόνο κόμβο σε μερικές χιλιάδες κόμβους. Πιο συγκεκριμένα, κάθε δουλειά MapReduce χωρίζεται σε μικρότερες μονάδες που ονομάζονται εργασίες. Ο συνολικός αριθμός εργασιών μπορεί να υπερβεί τον αριθμό των εργασιών που μπορούν να εκτελεστούν ταυτόχρονα στο σύνολο των διακομιστών, καθιστώντας απαραίτητο για τον χρονοπρογραμματιστή να διατηρήσει κάποια ουρά προτεραιότητας και να παρακολουθεί την πρόοδο των εργασιών εκτέλεσης έτσι ώστε οι εργασίες αναμονής να μπορούν να εκχωρηθούν σε κόμβους όταν θα είναι διαθέσιμοι.

Η βασική ιδέα πίσω από το MapReduce είναι να μετακινηθεί ο κώδικας και όχι τα δεδομένα. Στο MapReduce, αυτό το ζήτημα συνδέεται με τον προγραμματισμό και εξαρτάται σε μεγάλο βαθμό από το σχεδιασμό του κατανεμημένου συστήματος αρχείων. Για να επιτευχθεί η παραπάνω ιδέα, ο χρονοπρογραμματιστής ξεκινά εργασίες στον κόμβο που περιέχει ένα συγκεκριμένο σύνολο δεδομένων που απαιτείται από την εργασία. Αυτό έχει ως αποτέλεσμα τη μεταφορά κώδικα στα δεδομένα. Αν αυτό δεν είναι δυνατό (π.χ. ένας κόμβος εκτελεί ήδη πάρα πολλές εργασίες), νέες εργασίες θα ξεκινήσουν αλλού και τα απαραίτητα δεδομένα θα μεταδοθούν μέσω του δικτύου. Μια σημαντική βελτιστοποίηση εδώ είναι να προτιμούνται οι κόμβοι που βρίσκονται

στο κέντρο δεδομένων με τον κόμβο που συγκρατεί το αντίστοιχο μπλοκ δεδομένων, καθώς το εύρος ζώνης μεταξύ τους είναι σημαντικά μικρότερο από το εύρος ζώνης εντός του rack.

Στο MapReduce, τα ενδιάμεσα ζεύγη κλειδιών-τιμών πρέπει να ομαδοποιηθούν με βάση το κλειδί, το οποίο βρίσκεται με μια κατανεμημένη ταξινόμηση που περιλαμβάνει όλους τους κόμβους που εκτέλεσαν τις εργασίες map και όλους τους κόμβους που θα εκτελέσουν εργασίες reduce. Αυτό συνεπάγεται αναγκαστικά την αντιγραφή των ενδιάμεσων δεδομένων μέσω του δικτύου και επομένως η διαδικασία είναι κοινώς γνωστή ως «shuffle and sort».

Το πλαίσιο εκτέλεσης του MapReduce πρέπει να ολοκληρώνει όλες τις παραπάνω εργασίες σε ένα περιβάλλον όπου τα σφάλματα είναι ο κανόνας και όχι η εξαίρεση. Δεδομένου ότι το MapReduce σχεδιάστηκε ρητά για διακομιστές χαμηλών δυνατοτήτων, το πλαίσιο εκτέλεσης πρέπει να είναι ιδιαίτερα ανθεκτικό. Σε μεγάλες ομάδες, οι δυσλειτουργίες δίσκων είναι κοινές και η RAM βιώνει περισσότερα σφάλματα από το αναμενόμενο. Τα δίκτυα δεδομένων υποφέρουν από προγραμματισμένες διακοπές και απροσδόκητες διακοπές. Το πλαίσιο εκτέλεσης του MapReduce πρέπει να μπορεί να χειρίζεται τα παραπάνω και να λειτουργεί σωστά και αποδοτικά σε ένα τέτοιο περιβάλλον.

## Partitioners και Combiners

Οι partitioners είναι υπεύθυνοι για τη διαίρεση του ενδιάμεσου χώρου κλειδιών και την ανάθεση των ζευγών ενδιάμεσων κλειδιών-τιμών στους reducers. Με άλλα λόγια, ο partitioner καθορίζει την εργασία στην οποία πρέπει να αντιγραφεί ένα ενδιάμεσο ζεύγος κλειδιού-τιμής. Μέσα σε κάθε reducer, τα κλειδιά προσπελάσσονται με ταξινομημένη σειρά. Ο απλούστερος partitioner αναλαμβάνει τον υπολογισμό της τιμής κατακερματισμού του κλειδιού και στη συνέχεια τη λήψη του υπολοίπου της διαίρεσης της τιμής αυτής με τον αριθμό των reducers.

Οι combiners είναι μια βελτιστοποίηση στο MapReduce που επιτρέπει την τοπική συνάθροιση (local aggregation) πριν από τη φάση shuffle and sort. Όλα τα ζεύγη κλειδιών-τιμών πρέπει να αντιγραφούν σε όλο το δίκτυο και έτσι η ποσότητα των ενδιάμεσων δεδομένων θα είναι μεγαλύτερη από την ίδια την είσοδο. Αυτό είναι σαφώς αναποτελεσματικό. Μία λύση είναι να πραγματοποιηθεί τοπική συνάθροιση στην έξοδο κάθε mapper. Με αυτήν την τροποποίηση, ο αριθμός των ενδιάμεσων ζευγών κλειδιών-τιμών θα είναι το πολύ ο αριθμός των μοναδικών λέξεων στη συλλογή επί τον αριθμό των mappers.

Ένας combiner στο MapReduce υποστηρίζει μια τέτοια βελτιστοποίηση. Μπορούμε να σκεφτούμε τους συνδυασμούς ως «mini-reducers» που λαμβάνουν χώρα στην έξοδο των mappers, πριν από τη φάση shuffle and sort. Κάθε combiner λειτουργεί μεμονωμένα και ως εκ τούτου δεν έχει πρόσβαση στην ενδιάμεση έξοδο άλλων mappers. Όλοι οι combiners παρέχουν κλειδιά και τιμές που συνδέονται με κάθε κλειδί. Ένας combiner δεν θα έχει την ευκαιρία να επεξεργαστεί όλες τις τιμές που σχετίζονται με το ίδιο κλειδί, ενώ τα κλειδιά και οι τιμές πρέπει να είναι του ίδιου τύπου με την έξοδο του mapper. Σε κάποιες περιπτώσεις, οι reducers μπορούν να χρησιμεύσουν άμεσα ως combiners. Γενικά, οι reducers και οι combiners δεν είναι εναλλάξιμοι .

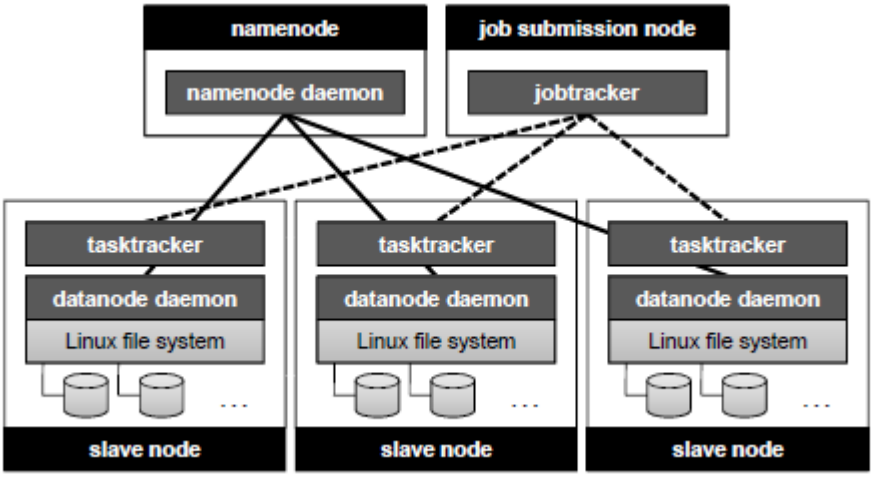
## Το κατανεμημένο σύστημα αρχείων

Καθώς αυξάνεται η τάξη του μεγέθους των δεδομένων, απαιτείται περισσότερη υπολογιστική ισχύς για την επεξεργασία τους. Όμως, καθώς η υπολογιστική χωρητικότητα αυξάνεται, η σύνδεση μεταξύ των υπολογιστικών κόμβων και της αποθήκευσης μπορεί να αποτελέσει τροχοπέδη. Τα κατανεμημένα συστήματα αρχείων δεν είναι καινούργια. Το σύστημα κατανομής αρχείων MapReduce βασίζεται σε προηγούμενες ιδέες, αλλά προσαρμόζεται ειδικά σε επεξεργασία μεγάλου όγκου δεδομένων και ως εκ τούτου αποκλίνει από ορισμένες αρχιτεκτονικές.

Η κύρια ιδέα είναι να διαιρεθούν τα δεδομένα χρήστη σε μπλοκ και να αναπαρασταθούν αυτά τα μπλοκ σε όλους τους τοπικούς δίσκους των κόμβων του συνόλου. Το κατανεμημένο σύστημα αρχείων υιοθετεί αρχιτεκτονική master-slave, στην οποία ο master (namenode) διατηρεί το χώρο ονομάτων (μεταδεδομένα, δομή καταλόγου, θέση μπλοκ, δικαιώματα πρόσβασης κτλ) και οι slaves διαχειρίζονται τα πραγματικά μπλοκ δεδομένων (datanode).

Στο Hadoop Distributed File System (HDFS), ένα πρόγραμμα-πελάτης που επιθυμεί να διαβάσει ένα αρχείο (ή ένα τμήμα του) πρέπει πρώτα να επικοινωνήσει με το namenode για να προσδιορίσει πού αποθηκεύονται τα πραγματικά δεδομένα του (Σχήμα 3). Η απάντηση στο αίτημα περιέχει το αντίστοιχο αναγνωριστικό του μπλοκ και τη θέση στην οποία βρίσκεται. Στη συνέχεια ο πελάτης επικοινωνεί με το datanode για να ανακτήσει τα δεδομένα. Τα μπλοκ αποθηκεύονται τα ίδια σε συνηθισμένα συστήματα αρχείων ενός μηχανήματος, επομένως το HDFS βρίσκεται πάνω από την κανονική στοίβα του λειτουργικού συστήματος.. Ένα σημαντικό χαρακτηριστικό του σχεδιασμού είναι ότι τα δεδομένα δεν μετακινούνται ποτέ μέσω του namenode. Αντίθετα, όλα τα δεδομένα μεταφέρονται απευθείας μεταξύ των πελατών και των datanodes.

Το namenode του HDFS έχει τις ακόλουθες αρμοδιότητες: τη διαχείριση του namespace, το συντονισμό των λειτουργιών του αρχείου και τη διατήρηση της συνολικής υγείας του συστήματος αρχείων.




Σχήμα 3: Αρχιτεκτονική ενός συστήματος Hadoop

## Η αρχιτεκτονική του Hadoop σε συστάδες υπολογιστών

Αν και εννοιολογικά στο MapReduce κάποιος μπορεί να σκεφτεί ότι ο mapper εφαρμόζεται σε όλα τα ζεύγη εισόδου κλειδιού-τιμής και ο reducer εφαρμόζεται σε όλες τις τιμές που σχετίζονται με το ίδιο κλειδί, η πραγματική εκτέλεση της εργασίας είναι λίγο πιο περίπλοκη. Στο Hadoop, οι mappers είναι αντικείμενα της Java που περιέχουν μια μέθοδο map. Ένα τέτοιο αντικείμενο δημιουργείται για κάθε εργασία map από τον κατανεμητή εργασιών. Ο κύκλος ζωής αυτού του αντικειμένου ξεκινάει με πρόσβαση στο API για την εκτέλεση του κώδικα που έχει οριστεί από τον προγραμματιστή. Αυτό σημαίνει ότι οι mappers μπορούν να διαβάσουν εξωτερικά δεδομένα, παρέχοντας την ευκαιρία φόρτωσης στατικών πηγών δεδομένων, λεξικών κτλ. Έπειτα, η μέθοδος map καλείται σε όλα τα ζεύγη κλειδιού-τιμής της εισόδου. Μετά την επεξεργασία όλων των ζευγών κλειδιού-τιμής το αντικείμενο του mapper παρέχει την ευκαιρία να εκτελεστεί κώδικας τερματισμού που έχει οριστεί από τον προγραμματιστή, και αυτό θα είναι σημαντικό για το σχεδιασμό των αλγορίθμων MapReduce.

Η πραγματική εκτέλεση των reducers είναι παρόμοια με αυτή των mappers. Ένα αντικείμενο reducer δημιουργείται για κάθε εργασία reduce. Το Hadoop παρέχει API για την εκτέλεση του κώδικα που έχει οριστεί από τον προγραμματιστή. Μετά την αρχικοποίηση, για κάθε ενδιάμεσο κλειδί, το πλαίσιο εκτέλεσης καλεί επανειλημμένα τη μέθοδο reduce με ένα ενδιάμεσο

κλειδί και έναν iterator για όλες τις τιμές που σχετίζονται με αυτό το κλειδί. Το μοντέλο προγραμματισμού εγγυάται επίσης ότι τα ενδιάμεσα κλειδιά θα παρουσιαστούν στη μέθοδο reduce με ταξινομημένη σειρά. Εφόσον αυτό συμβαίνει στο πλαίσιο ενός μόνο αντικειμένου, είναι δυνατόν να διατηρηθεί η κατάσταση ανάμεσα σε διαδοχικές εργασίες reduce. Αυτή είναι μια ιδιότητα που είναι κρίσιμη για το σχεδιασμό των αλγορίθμων MapReduce.

## Σχεδιασμός αλγορίθμων με το MapReduce

Ο σχεδιασμός των αλγορίθμων στο MapReduce γίνεται σχεδιάζοντας τους mappers και τους reducers ή και τους partitioners και τους combiners στις περιπτώσεις που χρειάζονται. Αυτό πρακτικά σημαίνει ότι οποιοσδήποτε αλγόριθμος πρέπει να εκφρασθεί με ένα μικρό αριθμό από άρρηκτα συνδεδεμένα μέρη τα οποία πρέπει να συνεργάζονται με πολύ συγκεκριμένο τρόπο ενώ όλες οι υπόλοιπες παράμετροι της εκτέλεσης προκύπτουν με διαφανή τρόπο από το πλαίσιο εκτέλεσης.

Η μεγαλύτερη πρόκληση, όπως έχει παρατηρηθεί είναι το θέμα του συγχρονισμού. Καθώς το πλαίσιο εκτέλεσης επιλέγει σε ποιο σύνολο διακομιστών ένας mapper ή ένας reducer θα εκτελεστεί, αλλά και το πότε και ποιες τιμές θα πάρει σαν είσοδο, ο προγραμματιστής έχει μια μοναδική ευκαιρία για να κάνει συγχρονισμό σε επίπεδο συνόλου κατά την φάση του shuffle and sort. Υπάρχουν τεχνικές οι οποίες συχνά υιοθετούνται όπως:

- Η δυνατότητα δημιουργίας δομών δεδομένων σαν κλειδιά ή τιμές (keys ή values) για την αποθήκευση μερικών αποτελεσμάτων
- Η δυνατότητα εκτέλεσης κώδικα ορισμένου από τον χρήστη κατά την εκκίνηση ή τον τερματισμό μιας map ή reduce εργασίας
- Η δυνατότητα να επιλεγεί η σειρά ταξινόμησης των ενδιάμεσων κλειδιών και συνεπώς και η σειρά με την οποία ο reducer θα επεξεργαστεί τα συγκεκριμένα κλειδιά

Ο τρόπος με τον οποίο ένας προγραμματιστής μπορεί να εκμεταλλευτεί τα παραπάνω είναι η διάσπαση σύνθετων αλγορίθμων σε μικρότερα στάδια, στα οποία η έξοδος του ενός θα είναι η είσοδος του επόμενου και η χρήση ενός προγράμματος οδηγού, το οποίο δεν θα αποτελεί εργασία του MapReduce αλλά θα συγχρονίζει τα επιμέρους.

## Τοπική Συγκέντρωση

Στα πλαίσια των εργασιών, οι οποίες διαχειρίζονται μεγάλο όγκο δεδομένων, η σημαντικότερη παράμετρος την οποία πρέπει να λάβουμε υπόψιν μας για να μπορούμε να σχεδιάσουμε τον συγχρονισμό του συστήματος είναι η ανταλλαγή ενδιάμεσων αποτελεσμάτων από την διεργασία που τα παράγει στην διεργασία που τα χρησιμοποιεί. Ειδικότερα, στο Hadoop, τα ενδιάμεσα αποτελέσματα εγγράφονται στον δίσκο πριν σταλούν στο δίκτυο και δεδομένου ότι αυτό προκαλεί μεγάλες τιμές λανθάνοντα χρόνου (latency), ο περιορισμός αυτών θα επιφέρει μεγάλη βελτίωση στην αποτελεσματικότητα του αλγορίθμου. Για να επιτευχθεί αυτό, γίνεται χρήση του combiner και της δυνατότητας να διατηρείται η κατάσταση σε πολλαπλές εισόδους προκειμένου να μειωθεί το νούμερο αλλά και το μέγεθος των ζευγαριών κλειδιού – τιμής που πρέπει να γίνουν shuffle από τους mappers στους reducers.

Ένα παράδειγμα χρήσης των παραπάνω είναι η καταγραφή των συχνοτήτων των εμφανίσεων όρων σε μια συλλογή κειμένων. Μια απλοϊκή υλοποίηση θα μπορούσε να αναθέτει κείμενα σε mappers και αυτοί να δίνουν σαν είσοδο στους reducers ζευγάρια (όρος, 1) με τελικό σκοπό να γίνει η συνάθροιση στους reducers (Σχήμα 4). Το κόστος σε μεταφορές και αποθηκεύσεις αυτής της μεθόδου θα ήταν τεράστιο μιας και θα εκλυόταν πλήθος ζευγαριών ίσο με το συνολικό πλήθος των όρων του κειμένου. Αντίθετα, μια προσέγγιση κατά την οποία σε κάθε mapper θα υλοποιούταν ένας combiner και θα πρόκυπτε σαν έξοδος ένα ζευγάρι (όρος, πλήθους εμφανίσεων) για κάθε *μοναδικό* όρο ενός εγγράφου, θα περιόριζε την τάξη του αριθμού των μεταφορών και αποθηκεύσεων σε O(μεγέθους λεξικού) (Σχήμα 5). Βέβαια, υπάρχουν και δύο ανασταλτικοί παράγοντες για την νέα αυτή προσέγγιση. Αρχικά, ο νόμος του Zipf ορίζει ότι λόγω της κατανομής των όρων, πολλοί δεν θα εμφανιστούν σε κάποιο mapper και άρα στην περίπτωση αυτή τα αποτελέσματα θα είναι ίδια με την αρχική μας εκδοχή. Παράλληλα, η χρήση των combiners είναι στην κρίση του πλαισίου εκτέλεσης και άρα δεν θα μπορούσαμε ποτέ να είμαστε σίγουροι για την επιτυχή βέλτιστη εκτέλεση του αλγορίθμου αυτού.

```
class Mapper
 method Map (docid a, doc d)
  for all term t ϵ doc d do
    Emit (term t, count 1)
```

```
class Reducer
 method Reduce (term t, counts [c1, c2, …])
  sum=0
  for all count c ϵ counts [c1, c2, …] do
   sum = sum + c
  Emit (term t, count sum)
```

Σχήμα 4: Ψευδοκώδικας για την πρώτη προσέγγιση.

```
class Mapper
 method Map (docid a, doc d)
  H = new AssociativeArray
  for all term t ϵ doc d do
   H[t] = H[t] + 1
  for all term t ϵ H do
   Emit (term t, count H[t])
```

Σχήμα 5: Ψευδοκώδικας για την δεύτερη προσέγγιση.

Είναι σημαντικό να αναφέρουμε ότι μπορούμε να βελτιστοποιήσουμε ακόμα περισσότερο τον παραπάνω αλγόριθμο εκμεταλλευόμενοι την ικανότητα των mappers να διατηρούν την εσωτερική τους κατάσταση ανάμεσα σε διαδοχικές εκτελέσεις. Έτσι, είναι δυνατόν, ένας mapper να δώσει αποτέλεσμα στην έξοδο μόνο αφού έχει επεξεργαστεί όλο το σώμα των κειμένων που του αναλογεί και έτσι να μειωθεί ακόμα περισσότερο ο όγκος των διακινούμενων δεδομένων (Σχήμα 6).

```
 class Mapper
 method Initialize
  H = new AssociativeArray
 method Map (docid a, doc d)
  for all term t ϵ doc d do
   H[t] = H[t] + 1
 method Close
  for all term t ϵ H do
   Emit (term t, count H[t])
```

Σχήμα 6: Ψευδοκώδικας για την τρίτη προσέγγιση

Η χρήση αυτής της τεχνικής (“in mapper combining”) είναι αρκετά διαδεδομένη καθώς γίνεται εκμετάλλευση στο μέγιστο βαθμό της τοπικής συνάθροισης και άρα δεν υπάρχει λόγος να χρησιμοποιηθεί εξωτερικός combiner.

Ένας παράγοντας που πρέπει να καθορίσουμε κατά τη χρήση “in mapper combining” είναι το μέγεθος της μνήμης που χρειάζεται για να διατηρηθεί η κατάσταση. Σύμφωνα με τον Νόμο του Heap το μέγεθος του λεξιλογίου μεγαλώνει σχετικά με το μέγεθος της συλλογής (μέγεθος λεξικού $V = kT^b$ όπου το $T$ είναι ο αριθμός των όρων στην συλλογή και τα $k$ και $b$ είναι παράμετροι με τιμές $30 \leq k \leq 100$ και $b \sim 0.5$). Έτσι, είναι αναγκαίο να χρησιμοποιήσουμε ένα μετρητή και μετά από ένα προκαθορισμένο αριθμό επεξεργασίας κλειδιών – τιμών ζευγαριών να εκλύουμε τα μερικά αποτελέσματα και να αδειάζουμε την μνήμη. Με επιλογή μικρού ορίου για τον μετρητή μπορεί να χάσουμε ευκαιρίες για μερική συνάθροιση, που είναι και το ζητούμενο, αλλά για μεγάλες τιμές μπορεί η μνήμη του mapper να γεμίσει και να αδυνατεί να επεξεργαστεί άλλα στοιχεία.

## Η τεχνική των Pairs και Stripes

Μια συνηθισμένη τεχνική που υλοποιείται στο Hadoop για να επιτευχθεί συγχρονισμός είναι η δημιουργία σύνθετων κλειδιών και τιμών προκειμένου τα δεδομένα που χρειάζονται για έναν υπολογισμό να συγκεντρώνονται από το πλαίσιο εκτέλεσης.

Για την περαιτέρω ανάλυση της τεχνικής αυτής θα χρησιμοποιηθεί το παράδειγμα της δημιουργίας μητρώων συνύπαρξης όρων από μεγάλες συλλογές δεδομένων. Ένα μητρώο συνύπαρξης είναι ένα τετραγωνικό μητρώο $n \times n$ όπου το $n$ είναι το πλήθος των διαφορετικών όρων που εμφανίζονται στην συλλογή. Κάθε κελί $(i, j)$ του μητρώου αυτού περιέχει τον αριθμό των φορών που η λέξη $w_i$ συνυπάρχει με την λέξη $w_j$ και συνήθως το μητρώο είναι συμμετρικό. Ο χώρος που χρειάζεται για την αποθήκευση του μητρώου είναι $O(n^2)$ και εξαρτάται τόσο από το μέγεθος του λεξικού όσο και από την προ επεξεργασία που γίνεται πριν εισαχθούν νέες λέξεις, όπως ληματοποίηση ή αποκατάληξη, καθώς και αντικατάσταση σπάνιων λέξεων με σταθερές συμβολοσειρές. Προφανώς, αν το μητρώο χωράει στην κύρια μνήμη ο υπολογισμός του είναι απλός, ενώ όταν χρησιμοποιείται και δευτερεύουσα μνήμη η υλοποίηση μπορεί να καθυστερήσει πολύ λόγω του paging. Παρακάτω παρουσιάζονται οι τεχνικές που είναι γνωστές ως pairs και stripes.

Στην τεχνική του pairs ο mapper επεξεργάζεται κάθε κείμενο στην είσοδο του και βγάζει στην έξοδο ζευγάρια λέξεων που συνυπάρχουν σαν κλειδί, και τον αριθμό 1 σαν τιμή. Έτσι, το

MapReduce εγγυάται ότι όλα τα ζευγάρια που θεωρούνται κλειδί θα σταλούν στους ίδιους reducers και ο κάθε ένας θα προσθέσει τα ξεχωριστά ζευγάρια για να προκύψει το τελικό αποτέλεσμα για όλα τα κείμενα της συλλογής, το οποίο και θα καταχωρηθεί στο μητρώο.

Στην τεχνική των stripes αντί για να εκλύεται από ένα mapper σαν κλειδί το ζευγάρι των λέξεων, προκύπτει η κάθε λέξη του κειμένου και σαν τιμή ένα associative array, το οποίο περιέχει τον αριθμό των εμφανίσεων της κάθε γειτονικής λέξης. Ο reducer προσθέτει όλες τις τιμές των associative arrays στοιχείο προς στοιχείο που έχουν το ίδιο κλειδί και ουσιαστικά συγκεντρώνει τα αθροίσματα που αντιστοιχούν στο κάθε κελί του μητρώου.

Συγκριτικά, το μοντέλο pairs δημιουργεί ένα τεράστιο αριθμό ζευγαριών κλειδί-τιμής, ενώ το μοντέλο stripes είναι πιο συμπαγές καθώς δημιουργεί λίστες. Παράλληλα, στο stripes το μήκος των κλειδιών αλλά και ο αριθμός τους είναι μικρότερος και άρα το πλαίσιο εκτέλεσης εκτελεί μικρότερο αριθμό ταξινομήσεων. Το μειονέκτημα του μοντέλου αυτού είναι ότι υποθέτει πως η κύρια μνήμη είναι αρκετή για να χωρέσουν τα associative arrays, ενώ σε διαφορετική περίπτωση θα χρησιμοποιηθεί η δευτερεύουσα, γεγονός το οποίο θα έχει μεγάλο αντίκτυπο στην απόδοση.

Τέλος, σημαντικό είναι να αναφερθεί ότι η προσέγγιση του stripes έχει περισσότερες ευκαιρίες για να υλοποιήσει τοπική συνάθροιση μιας και ο χώρος των κλειδιών είναι το μέγεθος του λεξικού, ενώ στην προσέγγιση του pairs αυτό δεν είναι δυνατό καθώς ο χώρος είναι το γινόμενο του λεξικού με τον εαυτό του.

## Δευτερεύουσα Ταξινόμηση

Το MapReduce προσφέρει την δυνατότητα της ταξινόμησης με βάση το κλειδί (key) αλλά όχι με βάση την τιμή (value). Αυτό μπορεί να είναι πρόβλημα αν, για παράδειγμα, ένα δίκτυο αισθητήρων δίνει μετρήσεις ανά τακτά χρονικά διαστήματα τις οποίες θέλουμε να επεξεργαστούμε με βάση τον κάθε ξεχωριστό αισθητήρα. Με την δεδομένη υλοποίηση του MapReduce θα καταλήξουν σε ένα reducer με βάση το μοναδικό αναγνωριστικό του αισθητήρα αλλά όχι με χρονολογική σειρά.

Υπάρχουν δύο τεχνικές για την επίλυση αυτού του ζητήματος. Η πρώτη είναι να χρησιμοποιηθεί ένας buffer σε κάθε reducer και να γίνει η ταξινόμηση μέσα στην μνήμη, τεχνική στην οποία μπορεί να προκύψει πρόβλημα επεκτασιμότητας λόγω του μεγέθους της απαιτούμενης μνήμης. Η δεύτερη τεχνική είναι η "value-to-key μετατροπή" κατά την οποία ένα μέρος της τιμής μεταφέρεται στο κλειδί για να δημιουργηθεί ένα συμπαγές κλειδί και γίνει η επεξεργασία από το

πλαίσιο εκτέλεσης. Στο παραπάνω παράδειγμα, η λύση είναι να χρησιμοποιούταν η χρονική στιγμή σαν ένα μέρος του κλειδιού.

Η τεχνική που έχει τα περισσότερα πλεονεκτήματα είναι η δεύτερη μιας και παρά τον μεγάλο όγκο των κλειδιών που προκύπτουν, η ταξινόμηση έγκειται βαθιά στην καρδιά του προγραμματιστικού μοντέλου του MapReduce και άρα έχουν γίνει όλες οι απαραίτητες βελτιστοποιήσεις.

## Ανεστραμμένα Ευρετήρια για Ανάκτηση Κειμένου

Στη σημερινή εποχή, σχεδόν όλες οι μηχανές αναζήτησης βασίζονται στην δομή δεδομένων που ονομάζεται ανεστραμμένο ευρετήριο. Η δομή αυτή δοθέντος ενός όρους αναζήτησης επιστρέφει όλα τα κείμενα στα οποία ο όρος περιέχεται. Γενικότερα, μια μηχανή αναζήτησης θα μπορούσε να διασπαστεί σε 3 διακριτά μέρη: συγκομιδή του περιεχομένου του web (crawling), δημιουργία ανεστραμμένου ευρετηρίου (indexing) και βαθμολογημένη επιστροφή κειμένων βάση ενός ερωτήματος (retrieval). Τα πρώτα δύο στάδια έχουν αρκετά κοινά χαρακτηριστικά με το σημαντικότερο να είναι το γεγονός ότι δεν χρειάζεται να υλοποιούνται σε πραγματικό χρόνο αλλά ανά τακτά χρονικά διαστήματα. Από την άλλη, η ανάκτηση των κειμένων πρέπει να συμπεριφέρεται ταχύτατα και να εξυπηρετεί ταυτόχρονα μεγάλο αριθμό χρηστών. Παρακάτω θα παρουσιάσουμε τον τρόπο με τον οποίο μπορούμε να εκμεταλλευτούμε το MapReduce προκειμένου να υλοποιήσουμε το δεύτερο στάδιο της διαδικασίας, δηλαδή την δημιουργία του ανεστραμμένου ευρετηρίου.

### Ανεστραμμένα Ευρετήρια

Στην βασικότερη μορφή του ένα ανεστραμμένο ευρετήριο αποτελείται από λίστες καταχωρήσεων (posting lists), μια για κάθε όρο που εμφανίζεται στην συλλογή των κειμένων. Μια λίστα καταχωρήσεων αποτελείται από ένα σύνολο εγγραφών, όπου η κάθε μια περιέχει ένα μοναδικό αναγνωριστικό (id) κειμένου και μια καταχώρηση (payload) που περιέχει πληροφορία για την εμφάνιση του όρου στο κείμενο. Η πιο συνηθισμένη τιμή της καταχώρησης είναι το term frequency ($tf$), δηλαδή ο αριθμός των εμφανίσεων του όρου στο κείμενο .

Παρακάτω θα θεωρηθεί ότι κάθε ξεχωριστό κείμενο έχει ένα μοναδικό αναγνωριστικό το οποίο είναι ένας αριθμός από το 1 εώς $n$, όπου $n$ ο αριθμός των κειμένων. Τα έγγραφα μέσα στις

λίστες καταχώρησης είναι ταξινομημένα με βάση αυτό το μοναδικό χαρακτηριστικό και συνεπώς είναι εμφανές ότι η τιμή του δεν δίνεται τυχαία (παρότι μπορεί να γίνει και αυτό σε κάποιες περιπτώσεις) αλλά αξιοποιείται μετα-πληροφορία για κάθε κείμενο προκειμένου να υπολογιστεί. Παραδείγματα αυτού του υπολογισμού είναι τα κείμενα από τον ίδιο ιστότοπο που μπορούν να πάρουν συνεχόμενα id ή αν γίνεται χρήση κάποιου αλγορίθμου για αξιολόγηση των σελίδων (πχ PageRank), οι τιμές των πιο ποιοτικών κειμένων να είναι μικρές για να είναι στις πρώτες θέσεις της κάθε λίστας καταχωρήσεων. Σε κάθε περίπτωση χρειάζεται μια δευτερεύουσα δομή δεδομένων, η οποία θα αντιστοιχεί κάθε μοναδικό αναγνωριστικό σε μια πληροφορία που θα είναι χρήσιμη για την ανάκτηση του εγγράφου πχ URL.

Όταν ένας χρήστης δώσει κάποιους όρους προς αναζήτηση, η μηχανή αναζήτησης πρέπει να επιστρέψει όλες τις λίστες καταχωρήσεων που περιέχουν τους όρους αυτούς με κάποια επεξεργασία η οποία μπορεί να είναι ένωση για λογικό Η ή τομή για λογικό ΚΑΙ. Είναι λοιπόν προφανές ότι το μέγεθος του ανεστραμμένου ευρετηρίου έχει τεράστια σημασία μιας και, εκτός από την Google, η οποία κρατάει την δομή στην κύρια μνήμη, οι τυχαίες προσπελάσεις δίσκου είναι αναπόφευκτες και συνεπώς η οργάνωση των προσπελάσεων αυτών αποτελεί στόχο προς επίτευξη. Βέβαια, είναι σημαντικό να αναφερθεί ότι υλοποιούνται τεχνικές συμπίεσης, οι οποίες είναι ικανές να μετατρέψουν ένα ευρετήριο μόλις στο ένα δέκατο του αρχικού τους μεγέθους, αλλά όσο αυξάνεται η μετα-πληροφορία που αποθηκεύεται τόσο δυσκολότερο είναι να επιτευχθούν τέτοια αποτελέσματα.

## Ανεστραμμένη Ευρετηριοποίηση: Μια αρχική προσέγγιση

Το MapReduce σχεδιάστηκε από την απαρχή του προκειμένου να μπορεί να δημιουργεί τις δομές δεδομένων που χρησιμοποιούνται στο Web συμπεριλαμβανομένων των ανεστραμμένων ευρετηρίων και του γραφήματος του Web. Παρακάτω θα παρουσιάσουμε έναν πρώτο αλγόριθμο για την δημιουργία του ανεστραμμένου ευρετηρίου.

Ο αλγόριθμος δέχεται σαν είσοδο στον mapper ένα αναγνωριστικό κειμένου και το αντίστοιχο κείμενο. Η προ επεξεργασία του κειμένου, όπως η αφαίρεση των HTML tags, η λημματοποίηση της κάθε λέξης και η αφαίρεση των λέξεων χωρίς χρήσιμο περιεχόμενο (stop words), δεν παρουσιάζεται αλλά θεωρείται δεδομένη.

Αρχικά, στον mapper δημιουργείται ένα associative array Η για κάθε κείμενο και για κάθε όρο σε αυτό υπολογίζεται η συχνότητα του η οποία και αποθηκεύεται. Έπειτα, για κάθε όρο ως

κλειδί δημιουργείται μια τιμή που είναι το ζευγάρι <αναγνωριστικό κειμένου, Η {όρου}> και εκλύονται στον reducer ως ζεύγος κλειδιού-τιμής.

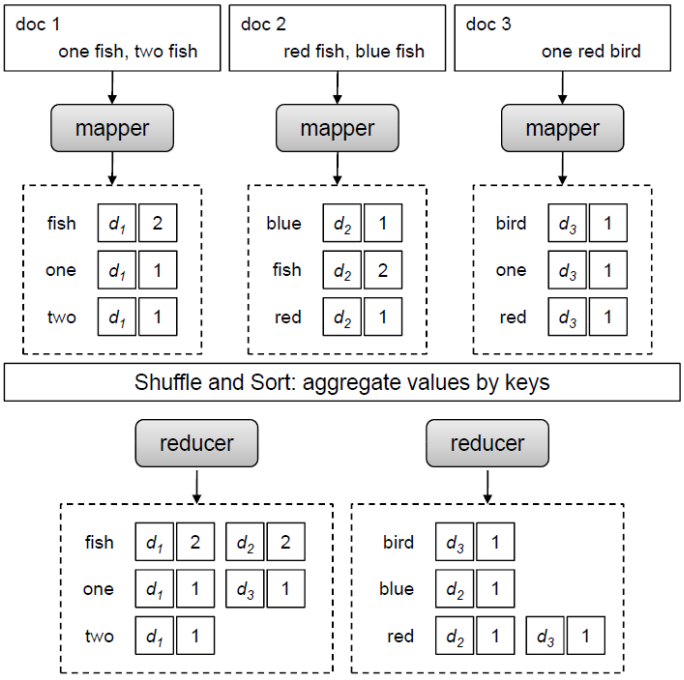
Το πλαίσιο εκτέλεσης του MapReduce συγκεντρώνει όλες τις λίστες καταχώρησης για κάθε όρο και τις αναθέτει σε ένα reducer. Εκεί αρχικοποιείται μια λίστα και συγκεντρώνονται όλες οι επιμέρους καταχωρήσεις οι οποίες και ταξινομούνται πριν εγγραφούν σε ένα αρχείο.

Λόγω της κατανεμημένης δημιουργίας αρχείων, στο τέλος θα υπάρχουν τόσα ανεστραμμένα ευρετήρια όσοι και οι reducers που χρησιμοποιήθηκαν. Έτσι είναι σημαντικό να δημιουργηθεί και ένα επιπλέον ευρετήριο το οποίο θα περιέχει πληροφορία για την τοποθεσία της κάθε λίστας καταχωρήσεων. Είναι συνηθισμένο να γίνεται μια χαρτογράφηση σε ζευγάρια (αρχείου, μετατόπιση [offset] byte) προκειμένου η μηχανή αναζήτησης να μπορεί να προσπελάσει απευθείας την πληροφορία που χρειάζεται. Παρακάτω παρουσιάζεται ο αλγόριθμος που περιγράφηκε και ένα παράδειγμα χρήσης του.

```
class Mapper
 procedure Map (docid n, doc d)
  H = new AssociativeArray
  for all term t ϵ doc d do
   H[t] = H[t] + 1
  for all term t ϵ H do
   Emit (term t; posting (n, H[t]))


class Reducer
 procedure Reduce (term t, postings [(n1, f1), (n2, f2), …])
  P = new List
 for all posting (a, f) ϵ postings [(n1, f1), (n2, f2), …] do
  Append (P, (a, f))
 Sort(P)
 Emit (term t, postings P)
```

Σχήμα 7: Ψευδοκώδικας αρχικής προσέγγισης για ευρετηριοποίηση




Σχήμα 8: Παράδειγμα διαδικασίας εκτέλεσης

## Ανεστραμμένη Ευρετηριοποίηση: Μια καλύτερη προσέγγιση

Το πρόβλημα που ανακύπτει από καλύτερη ανάλυση του παραπάνω αλγορίθμου είναι η θεώρηση ότι υπάρχει αρκετή μνήμη για να κρατηθούν όλα τα μοναδικά αναγνωριστικά που αφορούν έναν όρο. Ειδικότερα, στις γραμμές 4 και 5 όπου οι λίστες συνενώνονται και έπειτα ταξινομούνται, οι αλγόριθμοι που εκτελούνται βασίζονται σε δεδομένα που βρίσκονται στην κύρια μνήμη. Αυτό είναι πιθανό τροχοπέδη στην επεκτασιμότητα καθώς όσο οι λίστες καταχωρήσεων μεγαλώνουν, κάποια στιγμή θα ξεπεραστεί η διαθέσιμη μνήμη.

Ο δεύτερος αλγόριθμος που θα παρουσιαστεί ξεπερνά το πρόβλημα αυτό αφήνοντας το πλαίσιο εκτέλεσης να κάνει την ταξινόμηση, δεδομένου ότι τα κλειδιά φτάνουν σε κάθε reducer ταξινομημένα. Έτσι, ο κάθε mapper αντί να εκλύει τα ζευγάρια (όρος, <αναγνωριστικό κειμένου, Η {όρου}) κάνει την value-to-key μετατροπή που αναλύθηκε παραπάνω και βγάζει στην έξοδο του (<όρος, αναγνωριστικό κειμένου>, Η {όρου}). Με αυτή την τροποποίηση το προγραμματιστικό μοντέλο εγγυάται ότι τα ενδιάμεσα αποτελέσματα θα φτάνουν στους reducers ταξινομημένα, όπου και διατηρώντας την κατάσταση για πολλαπλά κλειδιά, μπορεί να κωδικοποιηθεί η ακολουθία των μοναδικών αναγνωριστικών και έτσι να χρειαστεί πολύ λιγότερη ποσότητα μνήμης. Ο αλγόριθμος παρουσιάζεται παρακάτω:

```
class Mapper
 method Map (docid n; doc d)
  H = new AssociativeArray
 for all term t ϵ doc d do
  H[t]= H[t] + 1
 for all term t ϵ H do
  Emit (tuple (t, n), tf H[t])

class Reducer
 method Initialize
  tprev = 0
  P = new PostingsList
 method Reduce (tuple (t, n), tf[f])
  if t != tprev ^ tprev != 0 then
   Emit (term t, postings P)
   P.Reset()
  P.Add((n, f))
  Tprev = t
method Close
 Emit (term t; postings P)
```

Σχήμα 9: Βελτιωμένος ψευδοκώδικας ευρετηριοποίησης

Σημαντικό, τέλος, είναι να αναφερθεί ότι όταν δεν έρθει κάποια νέα είσοδος στον reducer ($t \neq t_{prev}$) , ο αλγόριθμος θα τερματιστεί καλώντας την μέθοδο CLOSE, η οποία θα δώσει στην έξοδο την τελική καταχώρηση του ευρετηρίου για τον κάθε όρο.

## Τεχνικές Συμπίεσης

Στην ενότητα αυτή θα γίνει μια εισαγωγή σε βασικές τεχνικές συμπίεσης προκειμένου να μπορούν να χρησιμοποιηθούν για συμπίεση των λιστών καταχώρησης και αποδοτική αποθήκευση τους στον δίσκο.

Έστω ότι θα θέλαμε να αποθηκεύσουμε μια λίστα ζευγαριών (id, συχνότητα όρου). Η πρώτη προσέγγιση θα ήταν να αποθηκεύσουμε τα δεδομένα της λίστας όπως ακριβώς δημιουργούνται, περίπτωση στην οποία θα χρειαζόμασταν 6 byte για κάθε καταχώρηση (4 bytes για το id και 2 bytes για τη συχνότητα). Πχ [(5,2), (7,3), (12,1), (49,1), (51,2),…]

Μια καλύτερη προσέγγιση θα ήταν να αποθηκεύαμε τις διαφορές δυο συνεχόμενων τιμών των id. Οι διαφορές αυτές, που ονομάζονται d-gaps, θα είναι θετικοί ακέραιοι αριθμοί. Η παραπάνω λίστα θα γινόταν: [(5,2), (2,3), (5,1), (37,1), (2,2),…]

Η αντίστροφη διαδικασία της επαναφοράς της αρχικής λίστας είναι πολύ εύκολο να υπολογιστεί ξανά απλά προσθέτοντας τους όρους μεταξύ τους. Ένα μειονέκτημα αυτής της τεχνικής είναι ότι στην χειρότερη περίπτωση το μεγαλύτερο d-gap θα είναι (αριθμός κειμένων της συλλογής – 1) και άρα δεν θα έχει επιτευχθεί καμία βελτίωση.

Πριν συνεχίσουμε την ανάλυση τεχνικών συμπίεσης θα έπρεπε να αναφερθεί ότι όλες οι τεχνικές είναι ένα αντιστάθμισμα χρόνου και χώρου. Αυτό, πρακτικά, σημαίνει ότι μπορεί να επιτευχθεί μείωση του χώρου στον οποίο αποθηκεύονται τα δεδομένα, αλλά για να γίνει η κωδικοποίηση και η αποκωδικοποίηση τους θα πρέπει να επενδυθούν επιπλέον υπολογιστικοί κύκλοι. Παρακάτω θα διαφοροποιήσουμε τις τεχνικές που χρησιμοποιούνται σε αυτές που συμπιέζουν bytes ή words πληροφορίας και σε αυτές που δουλεύουν με bits.

## Συμπίεση σε επίπεδο Bytes και Words

Μια απλή προσέγγιση για την συμπίεση ενός ακεραίου είναι να χρησιμοποιηθούν μόνο όσα bytes χρειάζονται για την αναπαράσταση του. Αυτή η τεχνική ονομάζεται variable-length integer κωδικοποίηση (ή varInt για συντομογραφία). Ο τρόπος που επιτυγχάνεται το επιθυμητό αποτέλεσμα είναι με το να χρησιμοποιείται το πιο σημαντικό ψηφίο κάθε byte σαν ψηφίο συνέχισης και να τίθεται σε 1 στο τελευταίο byte και 0 οπουδήποτε αλλού. Έτσι, υπάρχουν διαθέσιμα 7 bits ανά byte για την κωδικοποίηση του αριθμού που σημαίνει ότι $0 \leq n < 2^7$ εκφράζονται με 1 byte, ότι $2^7 \leq n < 2^{14}$ εκφράζονται με 2 bytes κ.ο.κ. Παράδειγμα χρήσης είναι τα 127 και 128 που εκφράζονται αντίστοιχα ως εξής: $1\ 1111111, 0\ 0000001\ 1\ 0000000$

Η κωδικοποίηση αυτού του τύπου αποθηκεύεται στην μνήμη ευθυγραμμισμένη ανά byte και δεν υπάρχει αμφισημία για το που ξεκινάει μια κωδικοποιημένη λέξη και που τερματίζει. Το μειονέκτημα είναι ότι η αποκωδικοποίηση χρειάζεται πολλές ολισθήσεις και καλύψεις (masking) bits.

Στις περισσότερες αρχιτεκτονικές η προσπέλαση ολόκληρων λέξεων είναι πιο αποδοτική από την προσπέλαση μεμονωμένων bytes. Έτσι, για να επιτευχθεί η αποθήκευση σε λέξεις των16-bits, 32-bits ή των 64-bits αναπτύχθηκαν πολλοί αλγόριθμοι. Ένας από αυτούς είναι από τους Anh και Moffat, ο Simple-9 που βασίζεται σε λέξεις των 32-bits.

Στον Simple-9 από τα 32 bits κάθε λέξης, τα 4 χρησιμοποιούνται σαν επιλογέας και τα υπόλοιπα 28 για την αναπαράσταση του ακεραίου. Υπάρχουν 9 τρόποι για να για να διασπαστούν τα 28 bits σε ίσα μέρη (εξ' ου και το όνομα του αλγορίθμου) όπως σε 28 ομάδες του 1 bit, σε 14 ομάδες των 2 bits κτλ. Η αποκωδικοποίηση είναι απλή καθώς διαβάζεται ο επιλογέας και με την χρήση ενός πίνακα αντιστοίχισης μπορεί να βρεθεί ο αριθμός των ομάδων που χρησιμοποιήθηκαν. Αντίστοιχα, η κωδικοποίηση δουλεύει με τον αντίστροφο τρόπο, υπολογίζοντας πόσοι αριθμοί μπορούν να κωδικοποιηθούν στα 28 bits και έπειτα θέτοντας τα bit του επιλογέα, Το μειονέκτημα της τεχνικής αυτής είναι ότι μπορεί να μείνουν αχρησιμοποίητα bits κατά την ομαδοποίηση.

## Συμπίεση σε επίπεδο bit

Το πλεονέκτημα των αλγορίθμων της προηγούμενης ενότητας είναι ότι μπορούν να κωδικοποιηθούν και να αποκωδικοποιηθούν γρήγορα. Το μειονέκτημα, από την άλλη, είναι ότι πρέπει να χρησιμοποιηθούν πολλαπλάσια των 8 bits, ενώ πολλές φορές λιγότερα θα ήταν αρκετά. Στις τεχνικές συμπίεσης για bits, η κωδικοποίηση μπορεί να γίνει για οποιοδήποτε αριθμό bits και άρα να μην υπάρχουν αχρησιμοποίητα. Δεδομένου ότι τα όρια δεν είναι διακριτά, χρειάζεται περισσότερη προσπάθεια και περισσότερες καλύψεις και ολισθήσεις για την επιτυχή κωδικοποίηση και αποκωδικοποίηση.

Η πρώτη προσέγγιση που θα παρουσιαστεί έρχεται να καλύψει το πρόβλημα της μη διακριτότητας των διαφορετικών λέξεων δημιουργώντας μια κωδικοποίηση στην οποία καμία έγκυρη λέξη δεν μπορεί να είναι πρόθεμα κάποιας άλλης έγκυρης λέξης. Η κωδικοποίηση ονομάζεται unary και κάθε ακέραιος x κωδικοποιείται από x-1 bits στην τιμή 1 ακολουθούμενα από ένα bit στην τιμή 0. Ο αλγόριθμος αυτός είναι αποδοτικός για μικρούς αριθμούς, αλλά για μεγαλύτερους δεν προτιμάται, καθώς μεγαλώνουν οι απαιτήσεις σε αποθηκευτικό χώρο. Αυτός είναι και ο λόγος που σπάνια χρησιμοποιείται μόνος του ενώ αποτελεί συστατικό στοιχείο πιο σύνθετων. Ένα παράδειγμα για τους πρώτους 10 αριθμούς παρουσιάζεται στην δεύτερη στήλη του Πίνακα 1

Μια άλλη τεχνική που βασίζεται στην unary κωδικοποίηση, είναι η γ κωδικοποίηση. Κάθε ακέραιος $x$ διασπάται σε δύο μέρη, στο μήκος που κωδικοποιείται με unary και στο υπόλοιπο το

οποίο κωδικοποιείται δυαδικά. Το πρώτο μέρος δίνει την πληροφορία για το πόσα bits χρειάζονται για να κωδικοποιηθεί ο αριθμός και το υπόλοιπο κωδικοποιείται στα bits που μένουν. Οι τύποι που χρησιμοποιούνται για τα δύο μέρη είναι $n = 1 + \lfloor log_2 x \rfloor$ και $x - 2^{\lfloor log_2 x \rfloor}$ αντίστοιχα. Για παράδειγμα αν θέλαμε να κωδικοποιήσουμε το 10 θα προέκυπτε $n = 1 + \lfloor log_2 10 \rfloor = 4$, το οποίο σε unary είναι $1110$ και $10 - 2^{\lfloor log_2 10 \rfloor} = 10 - 8 = 2$, το οποίο με τα εναπομείναντα 3 bits είναι $010$. Αντίστοιχα στην τρίτη στήλη του Πίνακα 1 εμφανίζεται η κωδικοποίηση για αριθμούς από 1 μέχρι το 10.

Οι προηγούμενες δυο κωδικοποιήσεις ήταν μη παραμετρικές. Ακόμα καλύτερα αποτελέσματα μπορούν να επιτευχθούν με παραμετρικές όπως είναι η κωδικοποίηση Golomb. Η κωδικοποίηση αυτή δέχεται μια παράμετρο $b$ και για ένα ακέραιο $x > 0$ υπολογίζει 2 μέρη. Το πρώτο μέρος είναι το $q = \left\lfloor \frac{x-1}{b} \right\rfloor$ όπου και κωδικοποιείται το $q + 1$ σε unary και το δεύτερο μέρος είναι το υπόλοιπο $r = x - qb - 1$ το οποίο και κωδικοποιείται σε truncated δυαδικό. Το truncated δυαδικό δουλεύει ως εξής: αν το b είναι δύναμη του 2, τότε η κωδικοποίηση είναι ίδια με το κανονικό δυαδικό, ενώ σε αντίθετη περίπτωση κωδικοποιούνται οι πρώτες $2^{\lfloor log_2 b \rfloor + 1} - b$ τιμές του $r$ σε $\lfloor log_2 b \rfloor \, bits$ και τα υπόλοιπα σε $\lfloor log_2 b \rfloor + 1$ bits. Παράδειγμα χρήσης για b=5 εμφανίζεται στην τέταρτη στήλη του Πίνακα 1. Η κωδικοποίηση αυτή είναι σημαντική γιατί έρευνες έχουν δείξει ότι δουλεύει καλά για την κωδικοποίηση d-gaps και ότι φτάνει στο μέγιστο της απόδοσης της για την ακόλουθη τιμή της παραμέτρου b: $b \approx 0.69 \times \frac{df}{N}$, όπου με $df$ συμβολίζεται η συχνότητα εμφάνισης ενός όρου στο κείμενο και με $N$ ο αριθμός των κειμένων στην συλλογή.

| x | unary | γ | Golomb (b = 5) |
|---|---|---|---|
| 1 | 0 | 0 | 0 00 |
| 2 | 10 | 10 0 | 0 01 |
| 3 | 110 | 10 1 | 0 10 |
| 4 | 1110 | 110 00 | 0 110 |
| 5 | 11110 | 110 01 | 0 111 |
| 6 | 111110 | 110 10 | 10 00 |
| 7 | 1111110 | 110 11 | 10 01 |
| 8 | 11111110 | 1110 000 | 10 10 |
| 9 | 111111110 | 1110 001 | 10 110 |
| 10 | 1111111110 | 1110 010 | 10 111 |

Πίνακας 1: Παραδείγματα Κωδικοποίησης

## Συμπίεση Λιστών Καταχωρήσεων

Οι παραπάνω αλγόριθμοι που παρουσιάστηκαν θα εφαρμοστούν σε αυτή την ενότητα για να συμπιεστούν λίστες καταχωρήσεων. Πιο συγκεκριμένα, θα χρησιμοποιηθεί ο αλγόριθμος του Golomb για την συμπίεση των d-gaps και η γ-κωδικοποίηση για την συμπίεση της συχνότητας των όρων. Η πραγματική απόδοση ενός σχεδιασμένου με τέτοιο τρόπο σύστημα εξαρτάται από τα χαρακτηριστικά της συλλογής όπως είναι η κατανομή των d-gaps.

Η κωδικοποίηση με γ-codes είναι εύκολη καθώς δεν δέχεται κάποια παράμετρο και μπορεί να υπολογιστεί άμεσα. Αντίθετα, η κωδικοποίηση των d-gaps με χρήση της Golomb χρειάζεται να γνωρίζει το μέγεθος της συλλογής και τον αριθμό των ids που υπάρχουν σε μια λίστα καταχωρήσεων. Το πρώτο μπορεί να υπολογιστεί αυτόνομα και να περαστεί σαν σταθερά στον αλγόριθμο. Το δεύτερο, χρειάζεται μια μικρή τροποποίηση στα δεδομένα που εκλύει ο κάθε mapper.

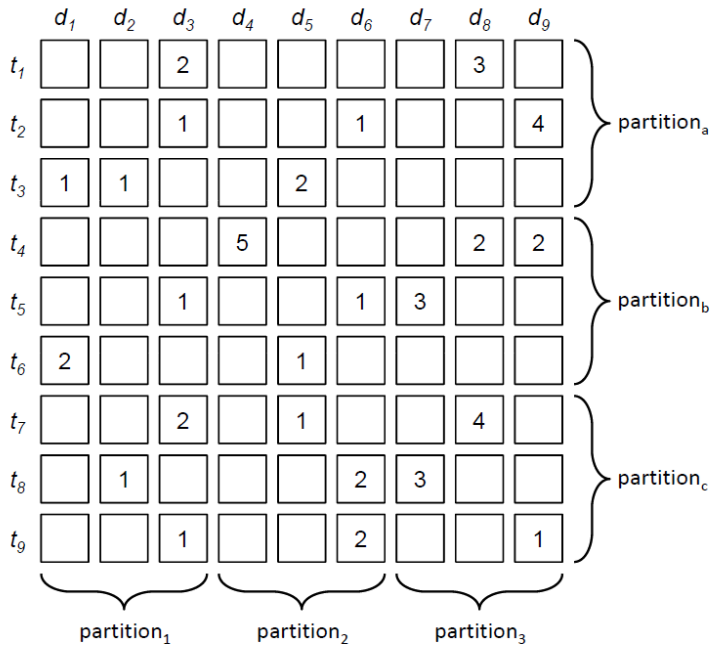
Η λύση που προτείνεται είναι ο κάθε mapper πέρα από τα ζευγάρια που δίνει στην έξοδο του, να υπολογίζει και μερικές συχνότητες και να τις εκλύει στην μορφή (<όρος,*>, μερική συχνότητα όρου). Ουσιαστικά, υλοποιείται το in–mapper combining στο οποίο ο κάθε mapper κρατάει στην μνήμη τον αριθμό των εγγράφων που έχει βρει ένα όρο και αφού τελειώσει με την επεξεργασία όλων των εγγράφων εκλύει αυτή την μερική συχνότητα. Το γεγονός ότι μέρος του κλειδιού είναι ο χαρακτήρας * είναι αρκετό για να έχει προτεραιότητα στην σειρά που θα φτάσει σε κάθε reducer όπου και θα αθροιστούν οι μερικές συχνότητες για να υπολογιστεί η τελική συχνότητα η οποία στην συνέχεια θα χρησιμοποιηθεί για να υπολογιστεί η παράμετρος b.

## Ανάκτηση Κειμένων στο MapReduce

Το MapReduce έχει σχεδιαστεί για εργασίες των οποίων η δομή είναι διαφορετική από τις ανάγκες ενός αλγορίθμου ανάκτησης. Η αναζήτηση των λιστών καταχώρησης χρειάζεται τυχαίες προσπελάσεις στον δίσκο, γεγονός το οποίο δεν καλύπτεται από τα σχεδιαστικά πρότυπα του συστήματος αρχείων που υποστηρίζει το MapReduce. Στο HDFS πρέπει να προσδιοριστεί η θέση του ζητούμενου μπλοκ δεδομένων από το namenode πριν βρεθούν τα αντίστοιχα δεδομένα.

Η ταυτόχρονη εξυπηρέτηση μεγάλου αριθμού χρηστών οι οποίοι ζητούν πολύ γρήγορη ανταπόκριση δεν είναι δυνατή οπότε η λύση έρχεται από την κατανομή του ευρετηρίου σε ένα μεγάλο πλήθος μηχανημάτων. Υπάρχουν δυο στρατηγικές για κατανεμημένη ανάκτηση: η διαμέριση κατά όρους και η διαμέριση κατά έγγραφα. Στην πρώτη περίπτωση, η συλλογή διασπάται σε υπο συλλογές από τις οποίες η κάθε μια αντιστοιχίζεται σε ένα εξυπηρετητή, ο οποίος είναι υπεύθυνος

για όλες τις εμφανίσεις ενός όρου (οριζόντια κατάτμηση). Στην δεύτερη περίπτωση ο κάθε εξυπηρετητής είναι υπεύθυνος για ένα υποσύνολο των όρων ή με άλλα λόγια, κατέχει όλες τις εμφανίσεις των όρων για ένα υποσύνολο των εγγράφων (κάθετη κατάτμηση).




Σχήμα 10: Διαμέριση κατά όρους (οριζόντια κατάτμηση) και Διαμέριση κατά έγγραφα (κάθετη κατάτμηση)

Η ανάκτηση με διαμέριση κατά έγγραφα χρειάζεται ένα ενδιάμεσο παράγοντα ο οποίος θα προωθήσει το ερώτημα του χρήστη σε όλους τους εξυπηρετητές, θα συνενώσει τα ενδιάμεσα αποτελέσματα και θα δώσει στην έξοδο το τελικό αποτέλεσμα. Με αυτή την αρχιτεκτονική για ένα ερώτημα πρέπει να χρησιμοποιηθούν όλοι οι εξυπηρετητές αλλά το γεγονός ότι δουλεύουν παράλληλα δίνει μικρότερους χρόνους αναζήτησης. Η ανάκτηση με διάσπαση κατά όρο διαχωρίζει το ερώτημα στους επιμέρους όρους και προωθεί τον πρώτο όρο στον κατάλληλο διακομιστή. Εκείνος επιστρέφει τα επιμέρους αποτελέσματα και τα προωθεί στον διακομιστή που περιέχεται ο δεύτερος όρος κοκ.

Έρευνες έχουν δείξει ότι η διαμέριση κατά έγγραφο είναι η καλύτερη στρατηγική λαμβάνοντας όλα τα κριτήρια υπόψιν και δεν είναι τυχαία το γεγονός ότι έχει υιοθετηθεί και από την Google.

Στα πλεονεκτήματα που αξίζει να αναφερθούν είναι ότι υπάρχει μεγάλος αριθμός τεχνικών για να διασπαστεί το web σε επιμέρους τμήματα αλλά και ότι μπορεί να υποστηριχθεί πολυεπίπεδη

αναζήτηση χρησιμοποιώντας διακομιστές οι οποίοι περιέχουν πιο ποιοτικά κείμενα και συνεχίζοντας έπειτα σε λιγότερο ποιοτικά.

Τέλος, λόγω της φύσης του MapReduce η αξιοπιστία της υπηρεσίας αναζήτησης επιτυγχάνεται με τον πλεονασμό των αποθηκευτικών μονάδων τόσο σε επίπεδο data center (λόγω σχεδιαστικού προτύπου) όσο και γεωγραφικά προκειμένου να εξυπηρετούνται πολλοί χρήστες από τον κοντινότερο σε αυτούς εξυπηρετητή και να μειώνεται ο χρόνος απόκρισης. Άλλωστε, η κατανομή του Zipf για τα ερωτήματα των χρηστών δίνει την δυνατότητα να αναπτύσσονται τεχνικές αποθήκευσης στην κρυφή μνήμη συχνών απαντήσεων και έτσι να δημιουργείται ακόμα μεγαλύτερο πλεονέκτημα για την κατανεμημένη αυτή προσέγγιση.

# Ο PageRank στο MapReduce

## Ο αλγόριθμος PageRank

Ο PageRank είναι ένας αλγόριθμος ο οποίος υπολογίζει την ποιότητα μιας ιστοσελίδας βασιζόμενος στην δομή του γραφήματος των συνδέσεων του Παγκοσμίου Ιστού.

Για την μοντελοποίηση του τρόπου λειτουργίας του αλγορίθμου θα χρησιμοποιηθεί ένας τυχαίος web surfer ο οποίος επισκέπτεται μια ιστοσελίδα και τυχαία ακολουθεί ένα σύνδεσμο. Αυτή η διαδικασία επαναλαμβάνεται επ' άπειρον. Πιο τυπικά, ο PageRank αποτελεί μια πιθανοτική κατανομή πάνω σε όλους τους κόμβους του γραφήματος που αναπαριστά την πιθανότητα ένας τυχαίος περίπατος πάνω στην δομή των υπερσυνδέσεων να καταλήξει σε ένα συγκεκριμένο κόμβο. Οι κόμβοι που έχουν μεγάλο αριθμό εισερχόμενων ακμών καθώς και αυτοί οι οποίοι έχουν εισερχόμενες ακμές από καλές ποιοτικά σελίδες, τείνουν να βαθμολογούνται με μεγάλες τιμές από τον αλγόριθμο.

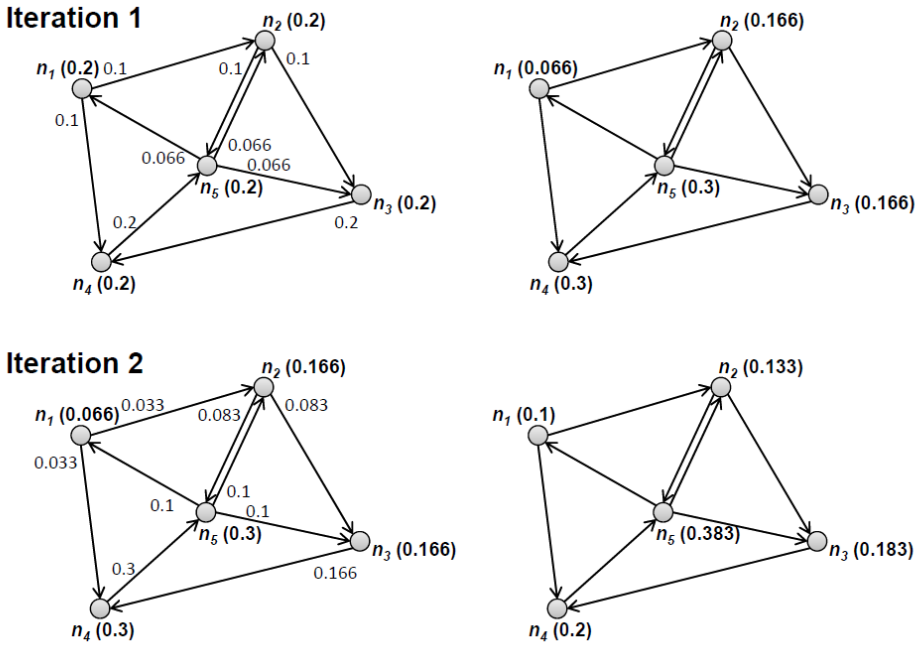
Η τυχαιότητα του web surfer στο να ακολουθήσει ένα υπερσύνδεσμο μπορεί να μοντελοποιηθεί από ένα μεροληπτικό νόμισμα το οποίο αν φέρει κορώνα, ο σύνδεσμος θα ακολουθηθεί ενώ διαφορετικά θα παραληφθεί και τυχαία ο surfer θα βρεθεί σε μια διαφορετική ιστοσελίδα.

Η τιμή του PageRank P για μια σελίδα n ορίζεται ως :

$$P(n) = a\left(\frac{1}{|G|}\right) + (1-a)\sum_{m \in L(n)} \frac{P(m)}{C(m)}$$

όπου $|G|$ είναι ο συνολικός αριθμός των κόμβων (σελίδων) στο γράφημα, $a$ είναι ο παράγοντας για την τυχαία ακολούθηση ενός συνδέσμου, $L(n)$ είναι το σύνολο των σελίδων που έχουν συνδέσμους προς την σελίδα n και το $C(m)$ είναι ο αριθμός εξερχόμενων ακμών από κάθε σελίδα $m$ (ο αριθμός των συνδέσμων που περιέχει η σελίδα αυτή).

Είναι προφανές ότι ο αλγόριθμος εκτελείται αναδρομικά. Στην αρχή της κάθε επανάληψης κάθε κόμβος αποστέλλει τις τιμές που έχει υπολογίσει στους κόμβους με τους οποίους είναι συνδεδεμένος. Από την στιγμή που ο PageRank είναι μια πιθανοτική κατανομή, αυτή η ανταλλαγή μηνυμάτων είναι παρεμφερής με το να διανέμεται η πιθανοτική μάζα (probability mass) στους γείτονες μέσω των εξερχόμενων ακμών. Για να ολοκληρωθεί ο γύρος, κάθε κόμβος αθροίζει όλες τις τιμές που έλαβε και υπολογίζει την ανανεωμένη PageRank βαθμολογία του. Ο αλγόριθμος επαναλαμβάνεται μέχρι οι βαθμολογίες των κόμβων να μην αλλάζουν. Στο παρακάτω σχήμα εμφανίζεται ένα παράδειγμα εκτέλεσης του αλγορίθμου.




Σχήμα 11: Παράδειγμα Εκτέλεσης του PageRank

Στο παράδειγμα αυτό παρουσιάζεται η εκτέλεση του αλγορίθμου για δυο γύρους. Για λόγους απλότητας δεν υπάρχουν κόμβοι που δεν έχουν εξερχόμενες ακμές καθώς και δεν συνυπολογίζεται ο παράγοντας τυχαίας ακολούθησης $\alpha$. Στον πρώτο γύρο ο κάθε κόμβος στέλνει σε όλους του γείτονες του ένα ισόποσο μερίδιο από την τιμή του PageRank. Δεδομένου ότι στον PageRank η τιμή των βαρών στους κόμβους αθροίζει στο 1 (πιθανοτικός αλγόριθμος) για 5 κόμ-

βους ξεκινάει ο καθένας με τιμή ίση με 0.2. Ένας κόμβος ο οποίος έχει δύο εξερχόμενες ακμές, όπως ο $n_1$, θα δώσει σε κάθε γείτονα του από 0.1 το οποίο αναπαριστά την πιθανότητα που έχει να επιλέξει ένα από τους δύο γείτονες. Στο τέλος της επανάληψης, κάθε κόμβος συλλέγει όλες τις εισερχόμενες τιμές και ανανεώνει την τιμή του. Ο κόμβος $n_1$ έχει παραπάνω εξερχόμενες ακμές από εισερχόμενες άρα σε κάθε επανάληψη θα δίνει παραπάνω βάρη από αυτά που δέχεται. Ο αλγόριθμος στην αρχή κάθε επανάληψης υπολογίζει το συνολικό άθροισμα των βαρών των κόμβων και ισοκατανέμει την διαφορά από το 1 για να συνεχίζει να υπάρχει έγκυρη πιθανοτική κατανομή των βαρών.

## Υλοποίηση του PageRank στο MapReduce

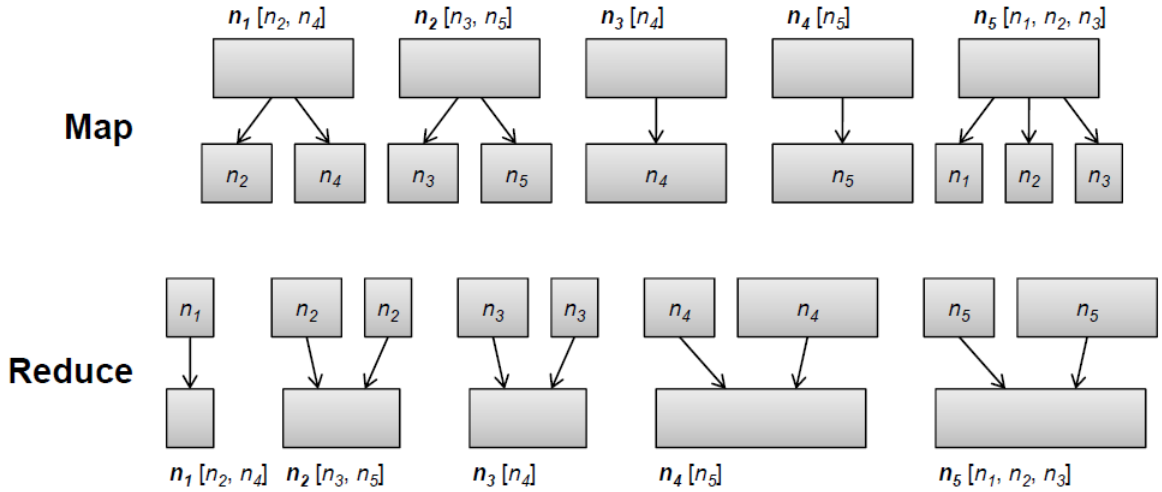
Στην ενότητα αυτή θα παρουσιαστεί μια υλοποίηση του αλγορίθμου PageRank κατάλληλα σχεδιασμένη για εκτέλεση στο MapReduce. Όπως και παραπάνω, στην αρχική του μορφή, δεν θεωρείται ότι υπάρχουν κόμβοι χωρίς εξερχόμενες ακμές καθώς και δεν λαμβάνεται υπόψιν ο συντελεστής τυχαίας μετάβασης α. Ο αλγόριθμος καταγράφεται στο Σχήμα 12.

```
class Mapper
 method Map (nid n, node N)
  p = N.PageRank / |N. AdjacencyList|
  Emit (nid n, N)
  for all nodeid m ϵ N.AdjacencyList do
   Emit (nid m, p)

 class Reducer
  method Reduce (nid m, [p1, p2, …])
   M = 0
   for all p ϵ counts [p1, p2, …] do
    if IsNode(p) then
     M = p
    else
     s = s + p
  M.PageRank = s
  Emit (nid m, node M)
```

Σχήμα 12: Ψευδοκώδικας για τον αλγόριθμο PageRank

Ο αλγόριθμος για κάθε κόμβο, προσπελαύνει τον αριθμό των γειτόνων του και εκλύει σαν ζευγάρι κλειδιού-τιμής το id του κάθε γείτονα μαζί με την τιμή του. Έπειτα, ο reducer αναλαμβάνει να συγκεντρώσει όλες τις τιμές που αντιστοιχούν σε ένα κόμβο και αφού τις αθροίσει, να ανανεώσει την τιμή του κόμβου αυτού. Μια εκτέλεση παρουσιάζεται στο παρακάτω σχήμα.




Σχήμα 13: Παράδειγμα εκτέλεσης PageRank στο MapReduce

Οι αλγόριθμοι που αφορούν γραφήματα και εκτελούνται στο MapReduce είναι αναγκαίο να περνούν σαν παράμετρο την δομή του γραφήματος. Έτσι, κάθε mapper πέρα από την πληροφορία που αφορά τον αλγόριθμο, δίνει στην έξοδο του και το γράφημα για κάθε κόμβο το οποίο συλλέγεται από τον reducer και γράφεται στον δίσκο για περαιτέρω χρήση.

Μια πιο σύνθετη υλοποίηση θα λάμβανε υπόψιν της και κόμβους οι οποίοι δεν έχουν εξερχόμενες ακμές καθώς και τον παράγοντα $\alpha$. Το πρόβλημα που ανακύπτει είναι το χαμένο βάρος σε τέτοιους κόμβους καθώς δεν εκλύεται κάποιο ζευγάρι από τον mapper. Μια πιθανή λύση είναι η χρήση μετρητών έτσι ώστε να αθροίζεται το βάρος που θα έπρεπε να κατανεμηθεί στους mappers. Μια άλλη λύση είναι να δημιουργηθεί ένα ειδικό κλειδί που θα εκλύει ο mapper μαζί με το βάρος ενός κόμβου και έπειτα να γίνεται ειδική επεξεργασία σε κάθε reducer για την ισοκατανομή. Τέλος, μια τρίτη προσέγγιση θα ήταν αρχικά να γράφονται οι χαμένες τιμές από κάθε mapper σαν δευτερεύοντα δεδομένα (χρησιμοποιώντας in-mapper combining) και ένα δεύτερο πέρασμα να τις συγκεντρώνει και να τις ισοκατανέμει. Η ισοκατανομή μπορεί να γίνει χρησιμοποιώντας τον εξής τύπο: $p' = a\left(\frac{1}{|G|}\right) + (1-a)(\frac{m}{|G|} + p)$, όπου $m$ είναι το συνολικό βάρος που έχει ανακτηθεί, $|G|$ ο συνολικός αριθμός των κόμβων και $\alpha$ ο τυχαίος παράγοντας μετάβασης.

Συνοψίζοντας, ένας γύρος του αλγορίθμου PageRank χρειάζεται δυο MapReduce εργασίες, μια για να κατανείμει το βάρος κάθε κόμβου στους γείτονες του και μια για να επιλυθεί το πρόβλημα των κόμβων χωρίς εξερχόμενες ακμές αλλά και για να ληφθεί υπόψιν η πιθανότητα τυχαίας μετάβασης με χρήση του παράγοντα $\alpha$. Ο αριθμός των γύρων που εκτελείται ο αλγόριθμος, τέλος, μπορεί είτε να είναι σταθερός και προκαθορισμένος, είτε να εκτελείται μέχρι να συγκλίνει, δηλαδή οι τιμές των βαρών να μην αλλάζουν.

## Βιβλιογραφία


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

[2] C. Manning, P. Raghavan και H. Schutze, Intoduction to information retrieval, Cambridge University Press, 2008.

[3] T. White, Hadoop: The Definitive Guide, O'Reilly Media, Inc, 2015.

[4] L. A. Barroso και U. Holzle, The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Morgan and Claypool Publishers, 2009.

[5] V. N. Anh και A. Moffat, «Inverted index compression using word-aligned binary codes» σε *Information Retrieval*, 2005.