ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Ομαδοποίηση εγγράφων του παγκόσμιου ιστού σε κατανεμημένα περιβάλλοντα με MapReduce και Hadoop

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Παπαδόπουλος Ανδρέας

Βόλος, Ιούνιος 2010

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Ομαδοποίηση εγγράφων του παγκόσμιου ιστού

σε κατανεμημένα περιβάλλοντα με

MapReduce και Hadoop

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Παπαδόπουλος Ανδρέας

Επιβλέποντας: Κατσαρός Δημήτριος

Βόλος, Ιούνιος 2010

Περίληψη

Το προγραμματιστικό μοντέλο Map Reduce προτάθηκε από την Google το 2004. Σκοπός του είναι να επεξεργάζεται μεγάλα δεδομένα εισόδου και να παράγει τα δεδομένα εξόδου χρησιμοποιώντας ένα σύνολο από διασυνδεδεμένους υπολογιστές (cluster) παραλληλίζοντας αυτόματα την εκτέλεση της εργασίας πάνω σε αυτό το δίκτυο υπολογιστών.

Στο πλαίσιο αυτής της Διπλωματικής Εργασίας θα ασχοληθώ με μια συγκεκριμένη υλοποίηση του προγραμματιστικού μοντέλου MapReduce, το Hadoop, μιας υλοποίησης ανοιχτού κώδικα γραμμένη σε Java. Χρησιμοποιώντας την προγραμματιστική διεπαφή (API) του Hadoop θα υλοποιήσω ένα αλγόριθμο για co-clustering εγγράφων.

Το co-clustering έχει μελετηθεί και χρησιμοποιηθεί σε διάφορα συστήματα όπως εξόρυξη γνώσης από κείμενο, βιοπληροφορική, συστήματα εισηγήσεων και εξόρυξης δεδομένων από γραφήματα. Λόγω του μεγάλου όγκου δεδομένων που χρειάζεται να επεξεργαστούμε πολλές φορές δεν είναι δυνατή η εφαρμογή του co-clustering σε μεγάλους πίνακες (μερικά εκατομμύρια γραμμές) για αυτό χρησιμοποιώντας το Hadoop μπορούμε να ομαδοποιήσουμε κατανεμημένα μεγάλο όγκο δεδομένων.

Πιο συγκεκριμένα στο στάδιο της αρχικοποίησης τα έγραφα αναλύονται με βάση τις λέξεις και την συχνότητα που εμφανίζονται σε αυτά. Στην συνέχεια με βάση αυτή την πληροφορία γίνεται ταυτόχρονα ομαδοποίηση τόσο για τις γραμμές (λέξεις) όσο και για τις στήλες (όνομα αρχείου ή διεύθυνση στον παγκόσμιο ιστό) (co-clustering) με βάση την ομοιότητα συνημίτονου (cosine similarity) ή κάποιο άλλο κριτήριο απόστασης. Σε αντίθεση με την απλή ομαδοποίηση (clustering), ομαδοποίηση μόνο με βάση τις γραμμές ή μόνο με βάση τις στήλες, βρίσκουμε υποπίνακες που έχουν κοινές ιδιότητες.

Στην συνέχεια θα κάνω κάποια πειράματα και θα παρουσιάσω γραφικά τα αποτελέσματα. Τέλος συγκρίνοντας τα αποτελέσματα αυτά παρουσιάζω τα συμπεράσματα μου και τους μελλοντικούς στόχους για την διπλωματική εργασία μου.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Εξόρυξη γνώσης

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Hadoop, MapReduce, co-clustering, ομαδοποίηση εγγράφων

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ

ΚΙΝΗΤΡΟ

Η ραγδαία ανάπτυξη του παγκόσμιου ιστού έχει δημιουργήσει την ανάγκη της γρήγορης και αποδοτικής επεξεργασίας μεγάλου όγκου δεδομένων (το Google επεξεργάζεται καθημερινά περίπου 20 petabytes δεδομένων). Για την απαραίτητη επεξεργασία δεν είναι πλέον δυνατή η χρήση ενός απλού υπολογιστή και για το λόγο αυτό έχουμε οδηγηθεί στην αναπόφευκτη χρήση συστημάτων πολυεπεξεργαστών (clusters). Αυτά τα συστήματα συνήθως αποτελούνται από αρκετούς διασυνδεδεμένους υπολογιστές.

Η ισχύς και αποδοτικότητα των συστημάτων αυτών σίγουρα είναι ένα θετικό στοιχείο αλλά δεν είναι επαρκής. Πολλές φορές είμαστε αναγκασμένοι να μειώσουμε των όγκο δεδομένων και να υλοποιήσουμε αποδοτικούς αλγορίθμους για την επίλυση των διάφορων προβλημάτων που παρουσιάζονται καθημερινά. Ένας τρόπος να κάνουμε την επεξεργασία που θέλουμε πιο γρήγορα, παίρνοντας τα ίδια αποτελέσματα, είναι η ομαδοποίηση (clustering).

Η ομαδοποίηση είναι να τοποθετήσουμε τα δεδομένα που θέλουμε να επεξεργαστούμε σε ομάδες και ακολούθως να επεξεργαστούμε αυτές τις ομάδες μειώνοντας έτσι κατά μεγάλο βαθμό τον χρόνο που χρειάζεται για την επεξεργασία τεράστιων συνόλων δεδομένων. Η ομαδοποίηση (clustering) γίνεται με επεξεργασία μόνο της μία διάστασης ενός πίνακα (μόνο γραμμές ή μόνο στήλες) με αλγόριθμους όπως ο K-means, QT κλπ. Στο clustering έρχεται να προστεθεί το co-clustering. Το co-clustering είναι η ταυτόχρονη ομαδοποίηση τόσο των γραμμών όσο και των στηλών ενός πίνακα (ομαδοποίηση σε δύο διαστάσεις) με αποτέλεσμα να έχουμε πολύ μικρότερου όγκου δεδομένα να επεξεργαστούμε. Το co-clustering έχει μελετηθεί και χρησιμοποιηθεί σε διάφορα συστήματα όπως εξόρυξη γνώσης από κείμενο, βιοπληροφορική, συστήματα εισηγήσεων και εξόρυξης δεδομένων από γραφήματα κλπ.

Στα πλαίσια αυτής της εργασίας θα ασχοληθώ με την υλοποίηση και αξιολόγηση ενός αλγόριθμου co-clustering χρησιμοποιώντας το Hadoop, ένα λογισμικό ανοιχτού κώδικα που παρέχει μια υλοποίηση του προγραμματιστικού μοντέλου Map Reduce.

ΤΟ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΟ ΜΟΝΤΕΛΟ MAP REDUCE

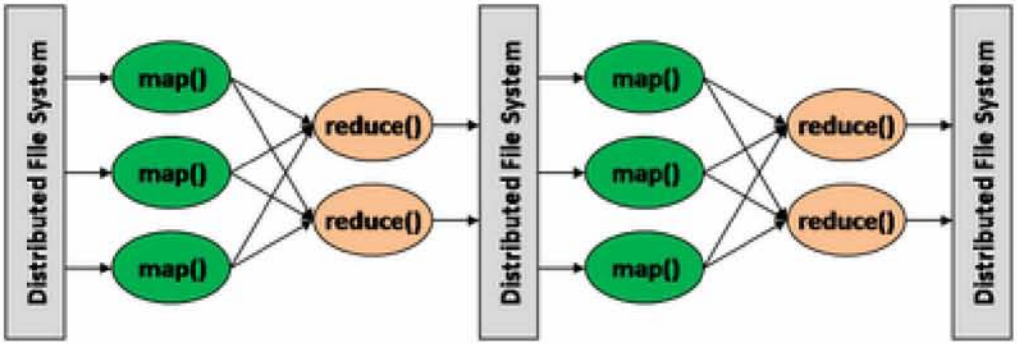
Το Map Reduce είναι ένα μοντέλο προγραμματισμού για την επεξεργασία και παραγωγή μεγάλων συνόλων δεδομένων πάνω σε ένα δίκτυο υπολογιστών (cluster) που αρχικά προτάθηκε από το Google το 2004. Το μοντέλο είναι αρκετά απλό στη χρήση και ευρέως διαδεδομένο. Το Google έχει υλοποιήσει πάνω από 10000 προγράμματα και κατά μέσο όρο κάθε μέρα τρέχουν στα clusters του Google 1000 ξεχωριστές εργασίες Map Reduce που συνολικά επεξεργάζονται πάνω από 20 petabytes δεδομένων.

Η ιδέα του MapReduce είναι να παίρνει σαν είσοδο ένα σύνολο από ζευγάρια <κλειδί εισόδου – τιμή> και να παράγει στην έξοδο ένα σύνολο από ζευγάρια <κλειδί εξόδου – αποτέλεσμα>. Ο προγραμματιστής εκφράζει/υλοποιεί τον αλγόριθμο σαν δύο συναρτήσεις/ μεθόδους, την *map* και την *reduce*.

Η συνάρτηση *map* δέχεται σαν είσοδο ένα ζεύγος κλειδί-τιμή και παράγει σαν έξοδο ένα ζεύγος κλειδί-τιμή. Η έξοδος της συνάρτησης *map*, ταξινομημένη με βάση το κλειδί, είναι η είσοδος της συνάρτησης *reduce*.

Η συνάρτηση *reduce* παίρνει σαν είσοδο την έξοδο της συνάρτησης *map* στην μορφή κλειδί-ενδιάμεσες τιμές και την επεξεργάζεται. Συνήθως για κάθε κλειδί έχουμε μία τιμή στην έξοδο.

Για την επίλυση κάποιου προβλήματος με το Map Reduce, ο προγραμματιστής πρέπει να υλοποιήσει τουλάχιστον την συνάρτηση *map*. Κάποιες απλές εργασίες μπορούν να υλοποιηθούν μόνο με την χρήση της συνάρτησης *map*.



Σχήμα 1. Αναπαράσταση Map Reduce συναρτήσεων

ΠΑΡΑΔΕΙΓΜΑΤΑ

ΠΑΡΑΔΕΙΓΜΑ1: WORDCOUNT

Το παράδειγμα αυτό παίρνει σαν είσοδο διάφορα αρχεία κειμένου και μετρά της εμφανίσεις κάθε λέξης μέσα σε όλα τα αρχεία που έχει πάρει σαν είσοδο.

Η *map*, αφού πάρει τα αρχεία, δίνει σε κάθε εμφάνιση μιας λέξης μια προσωρινή τιμή ίση με ένα. Όταν η *map* τελειώσει την εκτέλεση της και δώσει τα αποτελέσματα της στην *reduce*, τότε η *reduce* θα αθροίσει της αυτές της προσωρινές μεταβλητές για κάθε μοναδική λέξη και θα της δώσει πίσω σαν αποτέλεσμα τον αριθμό εμφάνισης της κάθε μοναδικής λέξης μέσα στα αρχεία.

Map	*Reduce*
Input: a document Output: key=word value=1	Input: key=word values=list of values (1) Output: key=word value=occurrences (SumOfInputValues)

Map(void *input) {
for each word w in input
EmitIntermediate(w, 1);
}
	Reduce(String key, Iterator values) {
	int result = 0;
	for each v in values
	result += v;
	Emit(w , result);
	}

Πίνακας 1. Ψευδοκώδικας WordCount

Από τον ψευδοκώδικα βλέπουμε πόσο απλό είναι να λύσουμε παράλληλα το πιο πάνω πρόβλημα. Μπορούν παράλληλα να τρέχουν τόσες *map* συναρτήσεις όσα είναι και τα αρχεία κειμένου που έχουμε, ή ακόμα και όσες είναι οι γραμμές σε όλα τα αρχεία με μοναδικό περιορισμό το υλικό που διαθέτουμε. Παρόμοια μπορούν να τρέχουν παράλληλα τόσες *reduce* συναρτήσεις όσα και τα κλειδιά εξόδου των *map* συναρτήσεων (λέξεις). Το Hadoop παραληλλοποιεί την εργασία χωρίς ο προγραμματιστής να ανησυχεί πως θα γίνει η εκτέλεση.

ΠΑΡΑΔΕΙΓΜΑ2: INVERTED INDEX

Στο παράδειγμα αυτό θα δούμε πως μπορούμε εύκολα με χρήση του Map Reduce να κατασκευάσουμε ένα inverted index. Σαν είσοδο έχουμε και πάλι μια συλλογή εγγράφων στα οποία θέλουμε να κτίσουμε το ευρετήριο, δηλαδή σε ποια αρχεία βρίσκεται κάθε λέξη.

Map	*Reduce*
Input: a document Output: key=word value=filename	Input: key=word value=list of values(filenames) Output: key=word value=files

Map(void *input) {
for each word w in input
EmitIntermediate(w, filename);
}
	Reduce(String key, Iterator values) {
	String files =" ";
	for each v in values
	files += v+"-";
	Emit(w , files);
	}

Πίνακας 2. Ψευδοκώδικας Inverted Index

Το παράδειγμα αυτό μπορεί εύκολα να επεκταθεί ούτως ώστε στο ευρετήριο να συμπεριλάβουμε και πόσες φορές εμφανίζεται η κάθε λέξη στο κάθε αρχείο. Αυτό μπορεί να γίνει με ελάχιστη επιπλέον δουλειά στην συνάρτηση *reduce*. Στο παράδειγμα αυτό εάν μία λέξη εμφανίζεται δύο ή περισσότερες φορές στο ίδιο αρχείο τότε στην έξοδο θα έχουμε στην τιμή (files) δύο ή περισσότερες φορές την ίδια ενδιάμεση τιμή (ίδιο αρχείο). Άρα απλά μετρώντας πόσες φορές έχουμε το ίδιο αρχείο-ενδιάμεση τιμή μπορούμε να βρούμε και πόσες φορές η κάθε λέξη εμφανίζεται στο κάθε αρχείο.

ΠΕΡΙΣΣΟΤΕΡΑ ΠΑΡΑΔΕΙΓΜΑΤΑ

	Map	*Reduce*
Distributed Grep	Εάν μία γραμμή από την είσοδο ταιριάζει με το πρότυπο (pattern) τότε δίνει στην έξοδο την συγκεκριμένη γραμμή.	Αντιγράφει την έξοδο της map. Μπορεί να παραληφθεί εάν δεν θέλουμε ταξινομημένη έξοδο.
Count of URL Access Frequency	Για κάθε εγγραφή (πρόσβαση σε κάποια σελίδα) στα αρχεία ιστορικού(logs) δίνει έξοδο <URL, 1>	Προσθέτει όλες τις τιμές και δίνει έξοδο <URL, total count>
Reverse Web-Link Graph	Για κάθε σύνδεσμο από μία σελίδα (source URL) προς μια άλλη σελίδα (target URL) δίνει έξοδο <target, source>	Για κάθε κλειδί (target) δίνει στην έξοδο την λίστα από τις σελίδες που έχουν σύνδεσμο προς αυτή <target, list(source)>
Distributed Sort	Για κάθε εγγραφή βρίσκει το κλειδί και δίνει έξοδο <key, record>	Δίνει στην έξοδο όλα τα ζεύγη που πήρε σαν είσοδο.

Πίνακας 3. Περισσότερα Παραδείγματα

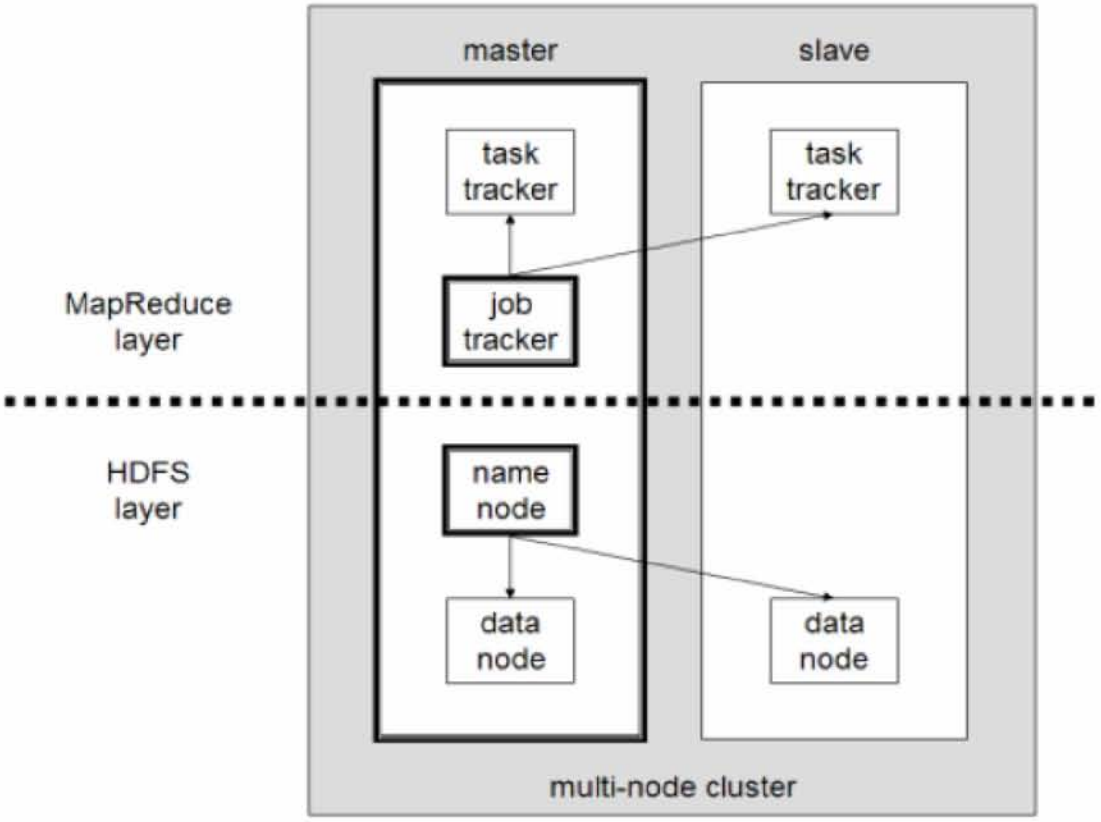
ΕΠΙΣΚΟΠΗΣΗ HADOOP

Το Hadoop είναι ένα λογισμικό ανοιχτού κώδικα που υποστηρίζει κατανεμημένη επεξεργασία μεγάλου όγκου δεδομένων (petabytes) και παρέχει μια υλοποίηση του MapReduce. Το Hadoop βασίστηκε στο Google Map Reduce framework και το Google File System (GFS). Είναι ένα έργο του Apache Software Foundation που αναπτύσσετε και χρησιμοποιείτε από ανθρώπους από όλο τον κόσμο και κυρίως την Yahoo!.

HADOOP DISTRIBUTED FILE SYSTEM – HDFS

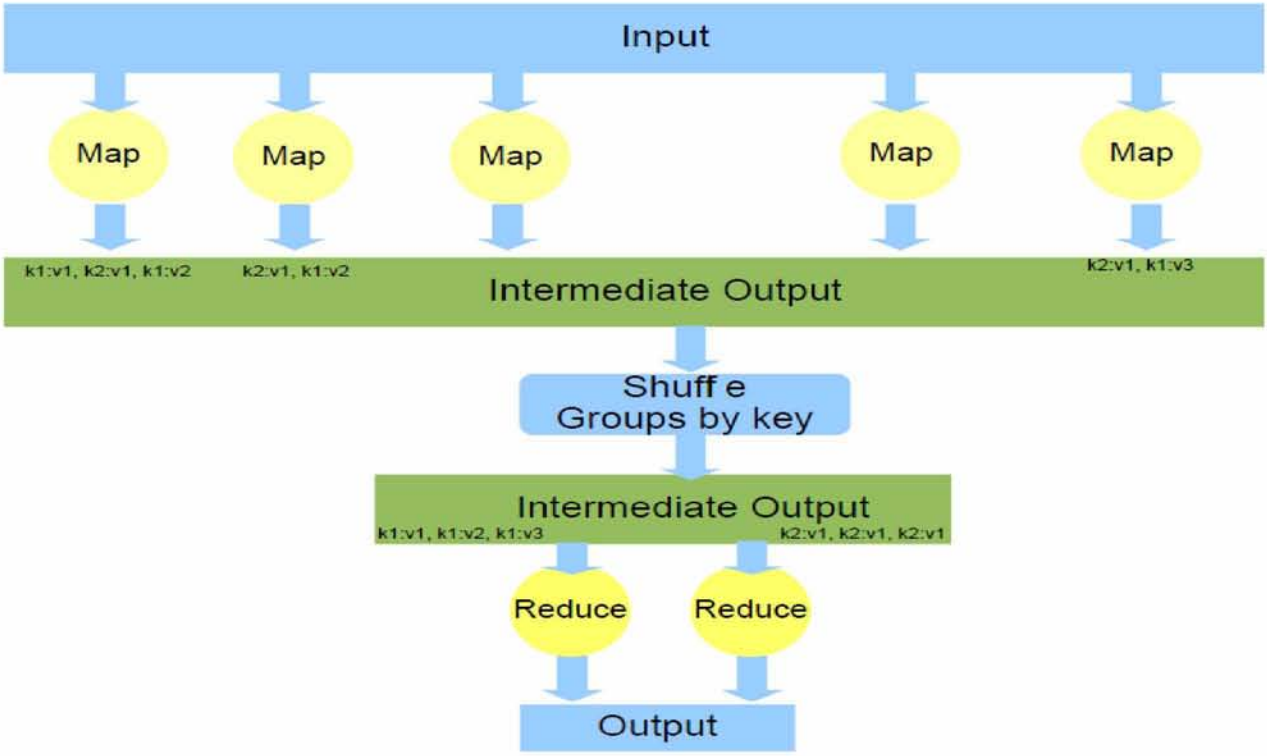
Το HDFS είναι ένα κατανεμημένο σύστημα αρχείων για αποθήκευση μεγάλων αρχείων (ιδανικά μεγέθους πολλαπλάσιο του 64Mb) παρόμοιο με το Google File System(GFS). Επιτυγχάνει να είναι αξιόπιστο αποθηκεύοντας τα δεδομένα σε περισσότερους από ένα κόμβους. Οι κόμβοι επικοινωνούν μεταξύ τους για εξισορρόπηση των δεδομένων, αντιγραφή ή μετακίνηση των δεδομένων, για να κρατηθεί ψηλά ο λόγος αντιγραφής (replication).

Το σύστημα αρχείων HDFS χρησιμοποιεί ένα κεντρικό κόμβο, τον *name node*, ο οποίος είναι και το μοναδικό σημείο αποτυχίας (single point of failure), που κρατά τις πληροφορίες για το που βρίσκεται κάθε δεδομένο στο HDFS. Αν αυτός δεν είναι διαθέσιμος τότε δεν υπάρχει πρόσβαση στο σύστημα αρχείων. Επιπλέον χρησιμοποιεί ακόμα ένα ακόμα κόμβο, τον *Secondary Namenode*, ο οποίος κρατά snapshots των φακέλων του name node και μαζί με τα αρχεία ιστορικού (logs) του name node επαναφέρει το σύστημα αρχείων μετά από αποτυχία. Οι υπόλοιποι κόμβοι ονομάζονται *datanodes* και απλά αποθηκεύουν δεδομένα.



Σχήμα 2. Αρχιτεκρονική Hadoop Cluster

Ο ΜΗΧΑΝΙΣΜΟΣ ΤΟΥ MAP REDUCE



Σχήμα 3. Μηχανισμός MapReduce

Ο μηχανισμός του Map Reduce ακολουθεί το μοντέλο αρχιτεκτονικής master/slave. Ο κεντρικός master server, *jobtracker*, αναλαμβάνει τον διαμερισμό των εργασιών στους υπόλοιπους κόμβους, slave servers – *tasktrackers*. Ο χρήστης στέλνει τις map και reduce εργασίες του στον jobtracker, ο οποίος με πολιτική FIFO (First In First Out) τις στέλνει στους tasktrackers για εκτέλεση. Οι tasktrackers απλά εκτελούν τις εργασίες που του αναθέτει ο jobtracker. Πριν από την έκδοση 0.21 εάν ένας tasktracker αποτύγχανε τότε η αντίστοιχη εργασία έπρεπε να εκτελεστεί από την αρχή.

	Master	Slave
MapReduce	jobtracker	tasktracker
HDFS	namenode	datanode

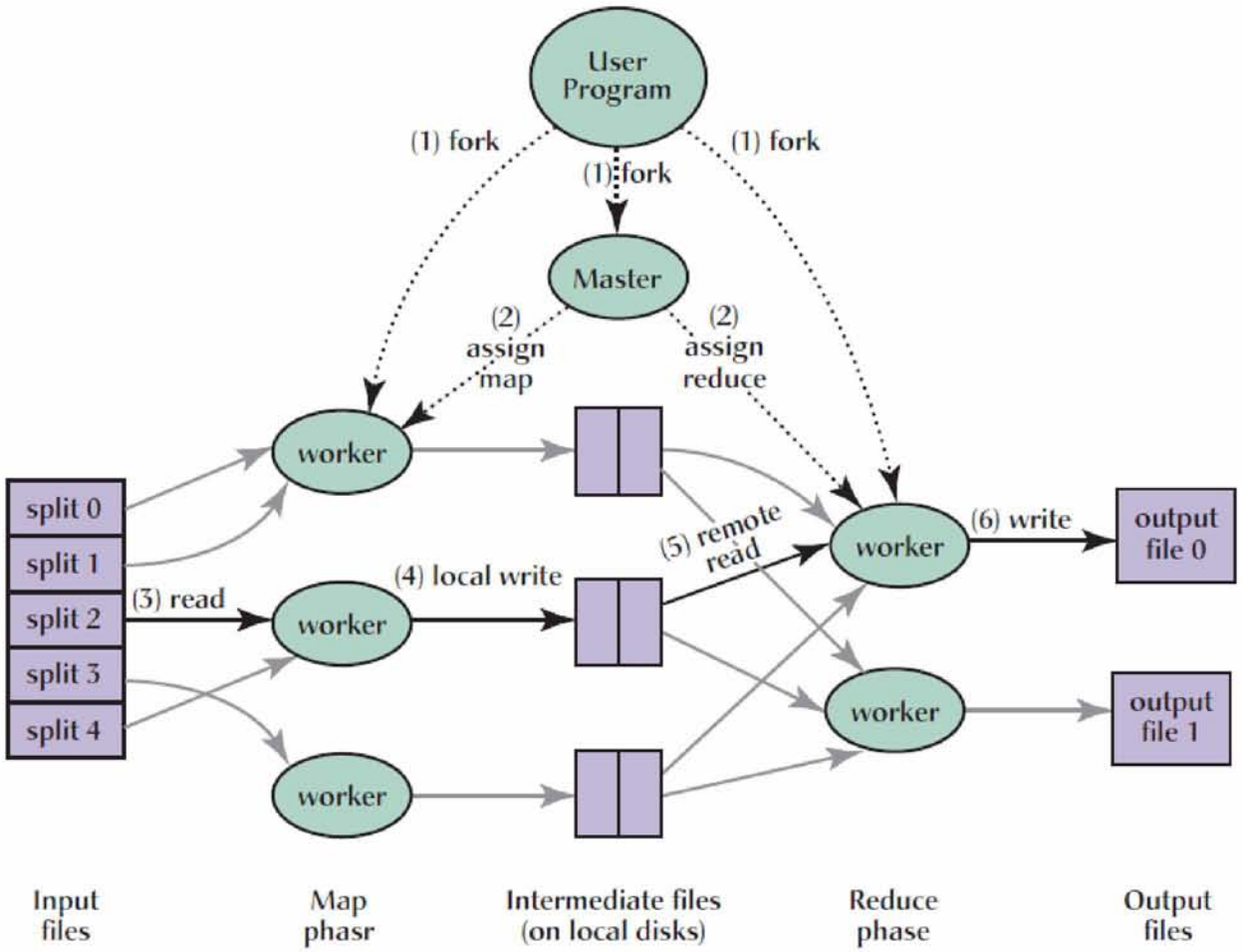
Πίνακας 4. Master/Slave - MapReduce και DFS

ΕΠΙΣΚΟΠΗΣΗ ΤΗΣ ΕΚΤΕΛΕΣΗΣ ΜΙΑΣ MAP REDUCE ΕΡΓΑΣΙΑΣ

Αρχικά η είσοδος μοιράζεται σε M κομμάτια, που μπορούν να επεξεργαστούν παράλληλα, κάθε ένα από τα οποία θα πάει σε ξεχωριστή map διεργασία. Τα ενδιάμεσα κλειδιά μοιράζονται στους R Reducers με χρήση κάποιας συνάρτησης κατακερματισμού (π.χ. hash (key) mod R). Η όλη εκτέλεση παρουσιάζεται στο πιο κάτω σχήμα και οι ενέργειες γίνονται με την εξής σειρά:

1. Η βιβλιοθήκη Map Reduce διασπά την είσοδο σε M κομμάτια, συνήθως μεγέθους 16-64Mb, και ξεκινά αρκετά αντίγραφα του προγράμματος στους κόμβους.
2. Ο master node - jobtracker αναλαμβάνει να βρει τους ανενεργούς κόμβους, tasktrackers, και να τους αναθέσει μια εργασία map ή reduce.
3. Εάν στο κόμβο, tasktracker, έχει ανατεθεί μια εργασία map τότε διαβάζει το κομμάτι εισόδου που του αντιστοιχεί και το διαμορφώνει ως κλειδί-τιμή και το δίνει ως παράμετρο στην συνάρτηση map. Τα ενδιάμεσα αποτελέσματα κρατούνται στην μνήμη.
4. Με βάση την συνάρτηση κατακερματισμού τα ενδιάμεσα αποτελέσματα γράφονται στο τοπικό δίσκο (σε R μέρη). Οι θέσεις των αποτελεσμάτων στέλνονται στον master - jobtracker ο οποίος είναι υπεύθυνος να τις διαμοιράσει στους R reducers - tasktrackers.
5. Όταν κάποιος reducer ειδοποιηθεί για αυτές τις τοποθεσίες διαβάζει απομακρυσμένα τα δεδομένα από το δίσκο του αντίστοιχου mapper. Αφού διαβάσει όλα τα δεδομένα τα ταξινομεί με βάση το κλειδί και ομαδοποιεί τιμές που αντιστοιχούν στο ίδιο κλειδί. Εάν τα δεδομένα δεν χωράνε στην μνήμη τότε χρησιμοποιείται κάποιος αλγόριθμος εξωτερικής ταξινόμησης.
6. Τα ταξινομημένα ενδιάμεσα αποτελέσματα περνιούνται στην συνάρτηση reduce για να παραχθεί η τελική έξοδος η οποία γράφεται στο τέλος ενός αρχείου για την έξοδο της συγκεκριμένης διαμέρισης.
7. Αφού τελειώσουν όλες οι εργασίες στους tasktrackers (map και reduce) ο master - jobtracker επιστρέφει και η εκτέλεση συνεχίζεται από τον κώδικα του χρήστη.

Στο κατάλογο που έχει ορισθεί για την έξοδο υπάρχουν R αρχεία (ένα για κάθε reducer). Συνήθως αυτά χρησιμοποιούνται σαν είσοδος σε κάποια άλλη Map Reduce εργασία και δεν χρειάζεται να συγχωνευτούν. Τα ονόματα των αρχείων, ο αριθμός των reducers καθώς και ο διαμερισμός της εισόδου μπορούν να οριστούν από τον χρήστη. Σε όλες τις φάσεις εκτελείτε διαχείριση σφαλμάτων και σε περίπτωση αποτυχίας το Hadoop μπορεί να επανεκκινήσει την εργασία που απέτυχε σε κάποιο άλλο κόμβο ή να ανακάμψει και να συνεχίσει την εκτέλεση (hadoop v0.21) με μικρό overhead. Το hadoop επιτυγχάνει καλό load balancing λόγω του μεγάλου αριθμού tasktrackers.



Σχήμα 4. Φάσεις εκτέλεσης μιας εργασίας MapReduce

ΧΡΗΣΕΙΣ ΤΟΥ HADOOP

Το Hadoop σήμερα είναι η πιο διαδεδομένη υλοποίηση του MapReduce και χρησιμοποιείται για διδακτικούς σκοπούς σε αρκετά πανεπιστήμια του κόσμου, αλλά και σε μεγάλους οργανισμούς ανά το παγκόσμιο για την επεξεργασία μεγάλων δεδομένων εισόδου. Κάποιοι από τους οργανισμούς, που διατηρούν clusters για εκτέλεση Hadoop εργασιών είναι: Yahoo!, Amazon, AOL, Alibaba, Cornell University Web Lab, ETH Zurich Systems Group, Facebook, Google, IBM, New York Times κ.α.

CO-CLUSTERING

Το co-clustering είναι μία τεχνική εξόρυξης γνώσης που ομαδοποιεί ταυτόχρονα τις γραμμές και τις στήλες ενός πίνακα. Συγκεκριμένα δοθέντος ενός πίνακα mxn ένας αλγόριθμος co-clustering δίνει στην έξοδο ομάδες γραμμών που έχουν παρόμοια χαρακτηριστικά σε κάποιο υποσύνολο στηλών και αντίστροφα.

Το πρόβλημα ανήκει στην κατηγορία NP-Complete για αυτό για την επίλυση του απαιτείται μεγάλη υπολογιστική ισχύς και/ή χρήση κάποιας ευρετικής συνάρτησης με μικρό σφάλμα για επίσπευση των υπολογισμών.

Το co-clustering αναφέρεται στην βιβλιογραφία και με τους όρους biclustering, biodimentional clustering, και subspace clustering. Για την επίλυση του co-clustering προτάθηκαν αρκετοί αλγόριθμοι μερικοί από τους οποίους είναι οι ακόλουθοι: Block clustering, CTWC, ITWC, δ-bicluster, δ-pCluster, δ-pattern, FLOC, OPC, Plaid Model, OPSMs, Gibbs, SAMBA, Robust Biclustering Algorithm (RoBA), Crossing Minimization, cMonkey, PRMs, DCC, LEB(Localize and Extract Biclusters), QBUIC(QUalitative BIClustering) και BCCA(Bi-Correlation Clustering Algorithm).

Οι πιο πάνω αλγόριθμου δίνουν διαφορετικά αποτελέσματα και αρκετές φορές δεν είναι ντετερμινιστικοί. Λόγω έλλειψης ενός προτύπου και λόγω της μη επιβλεπόμενης κατηγοριοποίησης δεν είμαστε σε θέση να κρίνουμε κατά πόσο το αποτέλεσμα είναι αξιόπιστο. Μια λύση στο πρόβλημα αυτό είναι να τρέξουμε αρκετούς αλγορίθμους και να επιλέξουμε το καλύτερο αποτέλεσμα.

Παρόλα αυτά το co-clustering έχει μελετηθεί και χρησιμοποιηθεί σε διάφορα συστήματα όπως εξόρυξη γνώσης από κείμενο, βιοπληροφορική, συστήματα εισηγήσεων και εξόρυξης δεδομένων κλπ.

ΟΡΓΑΝΩΣΗ

Στο επόμενο κεφάλαιο παρουσιάζονται βασικές εντολές για την χρήση του Hadoop καθώς και οδηγίες για την σύνταξη ενός προγράμματος. Στην συνέχεια στο κεφάλαιο 3 παρουσιάζεται μια λύση στο πρόβλημα του co-clustering με σειριακό αλγόριθμο και στο κεφάλαιο 4 ακολουθεί μια λύση βασισμένη στο μηχανισμό Map Reduce και Hadoop. Ακολούθως στο κεφάλαιο 5 παρατίθενται τα αποτελέσματα από πειραματικές μετρήσεις που έγιναν και τέλος στο κεφάλαιο 6 παρουσιάζονται τα συμπεράσματα και οι μελλοντικοί στόχοι της παρούσας εργασίας.

HADOOP

ΒΑΣΙΚΕΣ ΕΝΤΟΛΕΣ ΧΡΗΣΗΣ HADOOP

Για το χειρισμό του Hadoop όλες οι εντολές εκτελούνται στον κεντρικό κόμβο (master node). Σε αυτό το τμήμα παρουσιάζονται οι βασικές εντολές που πιθανόν να χρειαστούν για την λειτουργία του. Όλες οι εντολές εκτελούνται από τον κατάλογο εγκατάστασης του Hadoop.

Εντολή	Αποτέλεσμα και σχόλια
bin/hadoop namenode -format	Γίνεται format το σύστημα αρχείων. Πρέπει να εκτελεστεί αμέσως μετά την εγκατάσταση για να αρχικοποιηθεί το namenode. Δεν πρέπει να εκτελεστεί ξανά, γιατί θα χαθούν όλα τα δεδομένα από το file system.
bin/start-all.sh	Εκκίνηση του Hadoop στο cluster. Πρέπει να εκτελεστεί ούτως ώστε να αρχίσουν να τρέχουν όλοι οι συστατικοί κόμβοι του Hadoop (namenode, datanode, jobtracker και tasktracker).
bin/hadoop dfs –copyFromLocal [source dir] [destination dir]	Αντιγραφή δεδομένων από το τοπικό σύστημα αρχείων στο HDFS. Αντιγράφει το [source dir] από το τοπικό σύστημα αρχείων στο σύστημα αρχείων του Hadoop (HDFS) στον κατάλογο [destination dir].
bin/hadoop dfs –copyToLocal [source dir] [destination dir]	Αντιγραφή δεδομένων από το HDFS στο τοπικό σύστημα αρχείων. Αντιγράφει το [source dir] από το HDFS στο τοπικό σύστημα αρχείων στον κατάλογο [destination dir].
bin/hadoop dfs -ls	Προβολή περιεχομένων HDFS. Μπορεί να χρησιμοποιηθεί ακριβώς όπως η εντολή ls των Unix με επιπλέον παράμετρο το όνομα ενός φακέλου του HDFS.
bin/hadoop jar [hadoop program] [input] [output]	Εκτέλεση προγράμματος Hadoop. Η εντολή αυτή τρέχει το [hadoop program] πάνω στο Hadoop και χρησιμοποιεί σαν είσοδο τον κατάλογο [input] και έξοδο τον κατάλογο [output]. Και οι δύο κατάλογοι βρίσκονται στο HDFS.
bin/hadoop dfs –cat [path to file]	Προβολή αρχείων. Η εντολή αυτή μπορεί να χρησιμοποιηθεί για να έχει κανείς πρόσβαση στο περιεχόμενο ενός αρχείου που βρίσκεται μέσα στο HDFS. Έχει ίδια λειτουργία με την εντολή cat των Unix.
bin/hadoop dfs –rmr [directory]	Διαγραφής ενός κατάλογου από το HDFS. Η συγκεκριμένη εντολή διαγράφει τον κατάλογο [directory] από το HDFS είτε είναι άδειος είτε όχι.

Πίνακας 5. Βασικές εντολές χρήσης Hadoop

Πλήρης οδηγός εντολών του Hadoop βρίσκεται στο Παράρτημα Β.

ΠΩΣ ΝΑ ΓΡΑΨΕΤΕ ΕΝΑ ΠΡΟΓΡΑΜΜΑ HADOOP

Αν και το Hadoop είναι γραμμένο σε Java, τα προγράμματα τα οποία τρέχουμε σε αυτό δεν πρέπει κατά ανάγκη να είναι και αυτά γραμμένα σε Java. Μια εφαρμογή Hadoop μπορεί να είναι γραμμένη εκτός από Java σε γλώσσες όπως Python ή C++ (από την έκδοση 0.14.1 και μετά). Στο παράδειγμα που θα περιγράψω πιο κάτω θα εξηγήσω πώς να γράψουμε το δικό μας πρόγραμμα Word Count, σε γλώσσα Python, το οποίο κάνει χρήση του HadoopStreaming API, για να μας επιτρέψει να περνούμε δεδομένα μεταξύ του mapper και του reducer κάνοντας χρήση του STDIN (standard input) και STDOUT (standard output).

Πρώτα πρέπει να υλοποιήσουμε την μέθοδο map, η οποία θα μετρά τις εμφανίσεις της κάθε λέξης στο σύνολο εισόδου. Η map, η οποία θα φυλαχθεί σε ξεχωριστό αρχείο (π.χ map.py), θα γράφει τα αποτελέσματα της στο STDOUT.

map.py

```
#!/usr/bin/env python
import sys
# input comes from STDIN (standard input)
for line in sys.stdin:
        # remove leading and trailing whitespace
        line = line.strip()
        # split the line into words
        words = line.split()
        # increase counters
        for word in words:
                # write the results to STDOUT (standard output);
                # what we output here will be the input for the
                # Reduce step, i.e. the input for reducer.py
                # tab-delimited; the trivial word count is 1
                print '%s\t%s' % (word, 1)
```

Πίνακας 6. Κώδικας της συνάρτησης map για το παράδειγμα WordCount σε Python

Στην συνέχεια θα υλοποιήσουμε τη μέθοδο reduce, η οποία θα αθροίζει την κάθε εμφάνιση της κάθε λέξης και θα παρουσιάζει τα τελικά αποτελέσματα. Η reduce, η οποία θα φυλαχθεί σε ξεχωριστό αρχείο (π.χ reduce.py), θα παίρνει την είσοδο της από το STDIN.

reduce.py

```
#!/usr/bin/env python
from operator import itemgetter
import sys
# maps words to their counts
word2count = {}
# input comes from STDIN
for line in sys.stdin:
        # remove leading and trailing whitespace
        line = line.strip()
        # parse the input we got from mapper.py
        word, count = line.split('\t', 1)
        # convert count (currently a string) to int
        try:
                count = int(count)
                word2count[word] = word2count.get(word, 0) + count
        except ValueError:
                # count was not a number, so silently
                # ignore/discard this line
                pass
        # sort the words lexigraphically;
        # this step is NOT required, we just do it so that our
        # final output will look more like the official Hadoop
        # word count examples
        sorted_word2count = sorted(word2count.items(), key=itemgetter(0))
        # write the results to STDOUT (standard output)
        for word, count in sorted_word2count:
                print '%s\t%s'% (word, count)
```

Πίνακας 7. Κώδικας της συνάρτησης reduce για το παράδειγμα WordCount σε Python

Αφού αποθηκεύσουμε τα αρχεία map.py και reduce.py στον κατάλογο που έχουμε εγκαταστήσει το Hadoop, μπορούμε να δοκιμάσουμε το πρόγραμμα μας με την εντολή

bin/hadoop jar contrib/streaming/hadoop-0.19.1-streaming.jar –mapper mapper.py -reducer reducer.py -input [input directory] -output [output directory]

Πλήρης οδηγός για το Hadoop Streaming API βρίσκεται στο Παράρτημα Γ.

ΠΩΣ ΝΑ ΤΡΕΞΕΤΕ ΕΝΑ ΠΡΟΓΡΑΜΜΑ HADOOP

Για να τρέξετε ένα πρόγραμμα Hadoop πρέπει πρώτα να εγκαταστήσετε μια έκδοση του Hadoop σε ένα φάκελο της επιλογής σας. Επειδή όλα τα δεδομένα εισόδου πρέπει να είναι στο HDFS πρέπει να τα αντιγράψετε από το τοπικό σύστημα αρχείων του λειτουργικού σας.

Πρώτα πρέπει να ξεκινήσουμε το Hadoop τρέχοντας, από τον φάκελο που το εγκαταστήσαμε, την εντολή:

bin/start-all.sh

Αυτή η εντολή θα ξεκινήσει τα Namenode, Datanode, Jobtracker και Tasktracker.

Με την εντολή:

bin/hadoop dfs -copyFromLocal [source dir] [destination dir]

αντιγράφετε τα δεδομένα εισόδου στο HDFS.

Για να εκτελέσετε το πρόγραμμα τρέξετε την εντολή:

bin/hadoop jar [program] [input] [output]

Επιπλέον παράμετροι για την συγκεκριμένη εκτέλεση μπορούν να τοποθετηθούν πριν από το όνομα του προγράμματος (π.χ. bin/hadoop jar -Dmapred.task.timeout=0 [program] [input] [output])

Για να μεταφέρετε τα αποτελέσματα από το HDFS στο τοπικό σύστημα αρχείων του λειτουργικού σας εκτελείτε την εντολή:

bin/hadoop dfs –copyToLocal [source dir] [destination dir]

Τα αποτελέσματα μπορούν να παρουσιαστούν χωρίς να αντιγραφούν στο τοπικό σύστημα αρχείων με την εντολή:

bin/hadoop dfs –cat [dir]/*

CO-CLUSTERING ΕΓΓΡΑΦΩΝ

CO-CLUSTERING ΕΓΓΡΑΦΩΝ

Για το co-clustering εγγράφων η συλλογή αναπαριστάται στην μορφή πίνακα D του οποίου οι γραμμές ορίζουν τις λέξεις και οι στήλες τα έγγραφα. Το στοιχείο D_{ij} δηλώνει πόσες φορές η λέξη i εμφανίζεται στο έγγραφο j. Στην συνέχεια εφαρμόζεται κάποιος αλγόριθμος για να βρεθούν οι ομάδες εγγράφων που περιέχουν τις ίδιες ομάδες λέξεων.

ΣΕΙΡΙΑΚΟΣ ΑΛΓΟΡΙΘΜΟΣ CO-CLUSTERING ΕΓΓΡΑΦΩΝ

ΑΡΧΙΚΟΠΟΙΗΣΗ – ΥΠΟΛΟΓΙΣΜΟΣ ΠΙΝΑΚΑ ΓΕΙΤΝΙΑΣΗΣ

Στο στάδιο της αρχικοποίησης τα έγραφα αναπαρίστανται με δύο πίνακες. Το πίνακα D και τον ανάστροφο του. Ο ψευδοκώδικας για τον υπολογισμό του πίνακα D είναι:

```
For each word w in the collection of documents
        For each document d in the collection
                Count occurrence of w in d
        Print "w [occurrences]"
```

ΑΛΓΟΡΙΘΜΟΣ

Ο ψευδοκώδικας για τον σειριακό αλγόριθμο co-clustering εγγράφων είναι ο εξής:

1. Given a partition, calculate the "prototype" of each row cluster.
2. Assign each row x to its nearest cluster.
3. Update the probabilities based on the new row clusters and then compute new column cluster "prototype".
5. Assign each column y to its nearest cluster.
6. Update the probabilities based on the new column clusters and then compute new row cluster "prototype".
8. If converge, stop. Otherwise go to Step 2.

CO-CLUSTERING ΕΓΓΡΑΦΩΝ ΜΕ MAP REDUCE ΚΑΙ HADOOP

ΑΛΓΟΡΙΘΜΟΣ CO-CLUSTERING ΕΓΓΡΑΦΩΝ ΜΕ MAP REDUCE

ΑΡΧΙΚΟΠΙΗΣΗ – ΥΠΟΛΟΓΙΣΜΟΣ ΠΙΝΑΚΑ ΓΕΙΤΝΙΑΣΗΣ

Στο στάδιο της αρχικοποίησης υπολογίζουμε τον πίνακα D και τον ανάστροφο του. Ο πίνακας D περιέχει μία γραμμή για κάθε λέξη (w) που εμφανίζεται στη συλλογή εγγράφων και μία στήλη για κάθε έγγραφο (d) της συλλογής. Το στοιχείο στην θέση $D_{i,j}$ αντιστοιχεί στο πόσες φορές εμφανίζεται η λέξη i στο έγγραφο j.

Η αρχικοποίηση, δηλαδή ο υπολογισμός του πίνακα D καθώς και του αντίστροφου του, μπορεί να γίνει με χρήση του μηχανισμού Map Reduce. Παρακάτω παρατίθεται ο ψευδοκώδικας για τις συναρτήσεις map και reduce για τον υπολογισμό του πίνακα D που παίρνει σαν είσοδο την συλλογή εγγράφων που θέλουμε.

Map Input: Any document	Reduce Output: For each input output a vector of occurrences in documents
For each word w in input EmitIntermediate(w, 1-filename)	for each input key Sum values per filename output word_id – [occurrences]

Πίνακας 8. Ψευδοκώδικας για υπολογισμού του πίνακα γειτνίασης με Map Reduce

Η συνάρτηση map προσπελάζει όλα τα αρχεία στην συλλογή και για κάθε λέξη δίνει έξοδο το ζεύγος <λέξη, 1-όνομα αρχείου>. Η τιμή 1 δίνεται στην έξοδο ούτως ώστε να είναι δυνατή η χρήση μια κλάσης τύπου combiner για μείωση του όγκου δεδομένων που θα σταλούν πάνω από το δίκτυο. Κάθε reducer παίρνει όλα τα ζεύγη <λέξη, εμφανίσεις – αρχείο> για μια λέξη και έχοντας μια αντιστοίχιση για τα αρχεία σε αριθμούς δίνει στην έξοδο το ζεύγος < αριθμός που αντιστοιχεί στην λέξη(κλειδί) - ένα αραιό διάνυσμα με τις εμφανίσεις της λέξης στα έγγραφα>. Η χρήση αραιών διανυσμάτων μειώνει σημαντικά τον όγκο δεδομένων στην έξοδο καθώς μια λέξη μπορεί να εμφανίζεται μόνο σε πολύ λίγα αρχεία. Τα έγγραφα καθώς και οι λέξεις αντιστοιχίζονται σε ακέραιους αριθμούς. Έτσι δεν χρειάζεται να μεταφέρουμε ολόκληρη την λέξη ή ολόκληρο το όνομα ενός εγγράφου αλλά έναν ακέραιο αριθμό.

Για τον υπολογισμό του αντίστροφου πίνακα D^T μπορούμε και πάλι να χρησιμοποιήσουμε τον μηχανισμό Map Reduce. Μπορούμε να αντιστρέψουμε τον πιο πάνω πίνακα χωρίς να είναι απαραίτητη η γνώση οποιασδήποτε πληροφορίας για την συλλογή εγγράφων που θέλουμε να ομαδοποιήσουμε. Ο πιο κάτω ψευδοκώδικας παίρνει σαν είσοδο μόνο τον πίνακα D που θα αντιστρέψουμε.

Map Input: D	Reduce Output: D^T
for each record r in input for each element e in r EmitIntermediate(Index Of e, r Index-value)	for each input key output e – [values ordered by r]

Πίνακας 9. Ψευδοκώδικας υπολογισμού του αντίστροφου πίνακα γειτνίασης με Map Reduce

Κάθε map παίρνει σαν είσοδο ένα κομμάτι του πίνακα D. Για κάθε γραμμή στο πίνακα η συνάρτηση map δίνει στην έξοδο το ζεύγος < στήλη στοιχείου , γραμμή στοιχείου – τιμή στοιχείου>. Στη συνέχεια ο κάθε reducer θα πάρει όλα τα στοιχεία που ανήκουν στην ίδια στήλη. Με βάση την γραμμή που εμφανίζεται το στοιχείο στο αρχικό πίνακα τα ταξινομεί και δίνει έξοδο το ζεύγος <στήλη – αραιό διάνυσμα με τιμές>. Με χρήση κατάλληλης δομής δεν είναι απαραίτητη η ταξινόμηση με βάση τον αριθμό γραμμής π.χ. ένας πίνακας που στις θέσεις r έχει την τιμή που πήρε σαν είσοδο και 0 αλλού. Κάθε reducer παράγει το αραιό διάνυσμα για κάθε γραμμή του αρχικού πίνακα και τέλος όλα μαζί δίνουν τον αντίστροφο του αρχικού πίνακα.

ΑΛΓΟΡΙΘΜΟΣ

Κατά το πρώτο στάδιο εκτέλεσης πρέπει να υπολογιστούν κάποια αρχικά κέντρα στα οποία θα ταξινομηθούν τα δεδομένα. Ο αριθμός των αρχικών κέντρων επιλέγεται από τον χρήστη αλλά ο τελικός αριθμός ομάδων μπορεί να διαφέρει από αυτόν. Για το υπολογισμό των αρχικών κέντρων χρησιμοποιείται ο πιο κάτω κώδικας.

Map Input: records (r) (sparse vectors) of matrix D	Reduce Output: Initial Cluster Centers
Init: Number of initial cluster centers (k) Cluster Centers - C = {}	Init: Number of initial cluster centers (k)

If (C.size !=k) {
For any center c in C
If (r.cos_similarity(c) > 0.5) {
Emit(c id,r)
Return
}
C += {(new id,r)}
Emit(r id,r)
}
else {
For any center c in C
If (r.cos_similarity(c) > 0.5) {
Emit(c id,r)
Return;
}
Emit(k,r)
}
	Create a new ClusterDescriptor cd from values
	If (key == k) {
	Split cd using first element as center
	For any new ClusterDescriptor ncd {
	Output(key+i,ncd.centroid)
	Transpose ncd
	Output2(key+I,ncd.centroid)
	}
	}
	else {
	Output(key,cd.centroid)
	Transpose cd
	Ouptut2(key,cd.centroid)
	}

Πίνακας 10. Ψευδοκώδικας αρχικοποίησης CoClustering με Map Reduce

Το σύνολο C, που περιέχει τα μέχρι στιγμής κέντρα των νέων clusters, είναι κοινό για όλους τους mappers. Κάθε mapper προσπελάζει ένα μέρος από τον πίνακα D και τοποθετεί κάθε νέα εγγραφή (αραιό διάνυσμα) του πίνακα στο cluster που βρίσκει ότι έχει ομοιότητα (cosine similarity) μεγαλύτερη από ένα κατώφλι (0.5). Εάν το σύνολο C δεν είναι μεγέθους k τότε η εγγραφή προστίθεται στο σύνολο πριν την έξοδο. Εάν το σύνολο C είναι μεγέθους k και μια εγγραφή δεν έχει ομοιότητα μεγαλύτερη από το κατώφλι (στο κώδικα 0.5) τότε το κλειδί είναι ο αριθμός k. Τα ενδιάμεσα αποτελέσματα ταξινομούνται με βάση το κλειδί (id cluster) και καλείτε η συνάρτηση reduce.

Όλοι οι reducers γνωρίζουν από πριν τον αριθμό k που έχει καθορίσει ο χρήστης. Έτσι ο reducer που παίρνει τα ενδιάμεσα αποτελέσματα με κλειδί = k γνωρίζει ότι αυτά δεν μπορούν να βρίσκονται στο ίδιο cluster γιατί είναι αυτά που δεν είχαν ομοιότητα μεγαλύτερη από το κατώφλι με κανένα υπάρχον cluster. Έτσι τα τοποθετεί σε ένα νέο cluster το οποίο στην συνέχεια μοιράζει θεωρώντας ως κέντρο το πρώτο στοιχείο. Για κάθε νέο cluster που υπολογίζει βρίσκει το κέντρο και το δίνει στην έξοδο. Τα υπόλοιπα κλειδιά έχουν στοιχεία με ομοιότητα μεγαλύτερη από το κατώφλι σε σχέση με το πρώτο στοιχείο. Έτσι από αυτά υπολογίζεται το πραγματικό κέντρο του cluster το οποίο δίδεται και στην έξοδο. Επιπλέον για κάθε κέντρο που δίδεται στην έξοδο υπολογίζεται και το κέντρο για το αντίστοιχο cluster των στηλών.

Ο πιο πάνω κώδικας παρουσιάζει ένα απλό και εύκολο τρόπο για τον υπολογισμό των αρχικών κέντρων. Εάν ο χρήστης επιθυμεί μπορεί να ορίσει αυτός τα αρχικά κέντρα (και να αποφευχθεί ο υπολογισμός αυτός) ή να χρησιμοποιήσει κάποια άλλη μέθοδο (π.χ. canopy) για τον υπολογισμό τους.

Αφού υπολογιστούν τα αρχικά κέντρα των clusters ο αλγόριθμος εκτελείται επαναληπτικά μέχρι να συγκλίνει. Για την σύγκλιση μπορούμε να ορίσουμε σαν κριτήριο όταν κανένα cluster δεν μεταβάλλεται ή όταν τα νέα clusters που υπολογίζονται απέχουν μικρότερη απόσταση από ένα κατώφλι. Σε κάθε επανάληψη ο ψευδοκώδικας είναι ο εξής:

Map Input: records (r) (sparse vectors) of matrix D or D^T	Reduce Output: New Cluster Centers
Init: Centroids from previous iteration C	Init: Number of cluster centers from previous iteration (k)

For any center c in C
If (r.similarity(c) > 0.5) {
Emit(c id,r)
Return;
}
Emit(k,r)
	Create a new ClusterDescriptor cd from values
	If (key == k) {
	Split cd using first element as center
	For any new ClusterDescriptor ncd {
	Output(key+i,ncd.centroid)
	Transpose ncd
	Output2(key+i,ncd.centroid)
	}
	}
	else {
	Output(key,cd.centroid)
	Transpose cd
	Ouptut2(key,cd.centroid)
	}

Πίνακας 11. Ψευδοκώδικας CoClustering με Map Reduce

Στο κώδικα αυτό σε κάθε επανάληψη χρειάζεται οι mappers να προσπελάσουν όλο τον πίνακα D για υπολογισμό των clusters για τις γραμμές του πίνακα ή όλο τον πίνακα D^T για υπολογισμό των clusters των στηλών του πίνακα.

Ο κώδικας της συνάρτησης reduce είναι ο ίδιος με το στάδιο της αρχικοποίησης αλλά ο κώδικας της συνάρτησης map δεν υπολογίζει ξανά τα αρχικά κέντρα αλλά παίρνει σαν είσοδο τα κέντρα από την έξοδο της προηγούμενης επανάληψης.

ΣΧΟΛΙΑ ΓΙΑ ΤΟΝ ΚΩΔΙΚΑ

Σημειώνεται ότι ο τελικός αριθμός των clusters δεν είναι ίδιος με τον αρχικό αριθμό k που καθορίζεται από τον χρήστη και μπορεί να είναι μεγαλύτερος ή μικρότερος. Ο χρήστης μπορεί να επιλέξει το κατώφλι για την απόσταση στοιχείων από το κέντρο του cluster, τον αριθμό των αρχικών κέντρων καθώς και το κριτήριο τερματισμού του αλγορίθμου.

ΑΠΟΤΕΛΕΣΜΑΤΑ ΠΕΙΡΑΜΑΤΩΝ

ΠΛΑΤΦΟΡΜΕΣ ΕΚΤΕΛΕΣΗΣ

Για την εκτέλεση των πειραμάτων χρησιμοποιήθηκε ένας υπολογιστής που αποτελείται από 4 4-πύρηνους επεξεργαστές Quad-Core AMD Opteron(tm) Processor 8350 και 16Gb RAM. Η χωρητικότητα του συστήματος HDFS είναι 200GB. Το λειτουργικό σύστημα είναι SUSE Linux Enterprise Server 10 (x86_64) με έκδοση πυρήνα 2.6.16.21-0.8-smp. Το Hadoop είχε ρυθμιστεί σε pseudo distributed mode και έκδοση 0.18.3. Ο υπολογιστής διαμοιράζεται και με άλλους χρήστες.

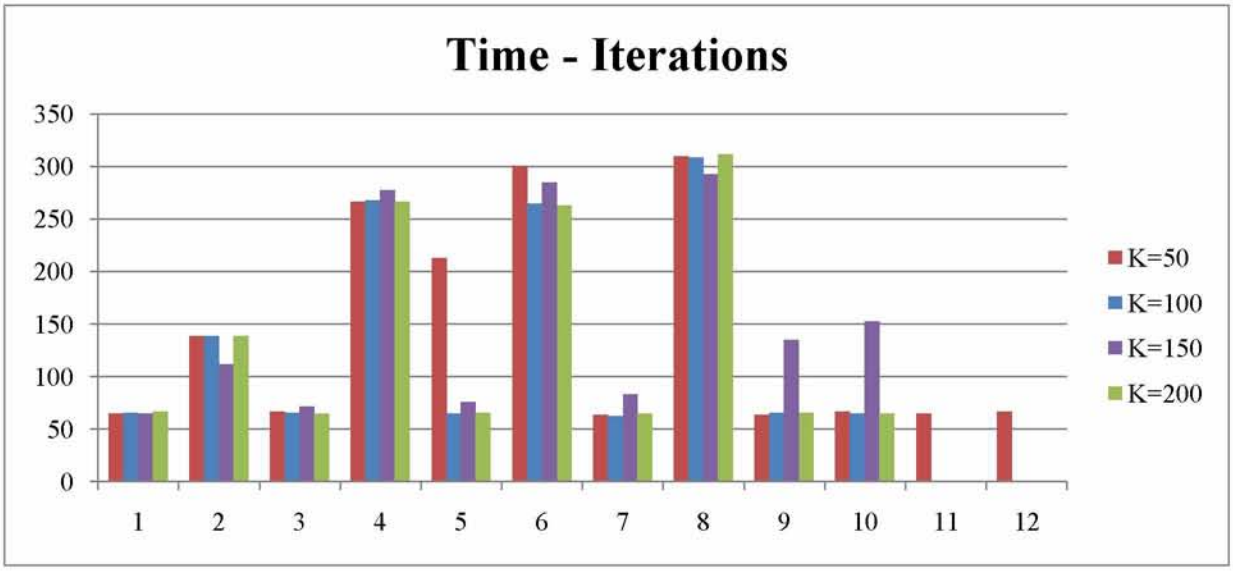
Για την εκτέλεση των πειραμάτων, μετά από πρόχειρες μετρήσεις, ο αριθμός των map tasks τέθηκε ίσος με 16 και reduce tasks τέθηκε ίσος με 12.

ΑΠΟΤΕΛΕΣΜΑΤΑ ΠΕΙΡΑΜΑΤΩΝ

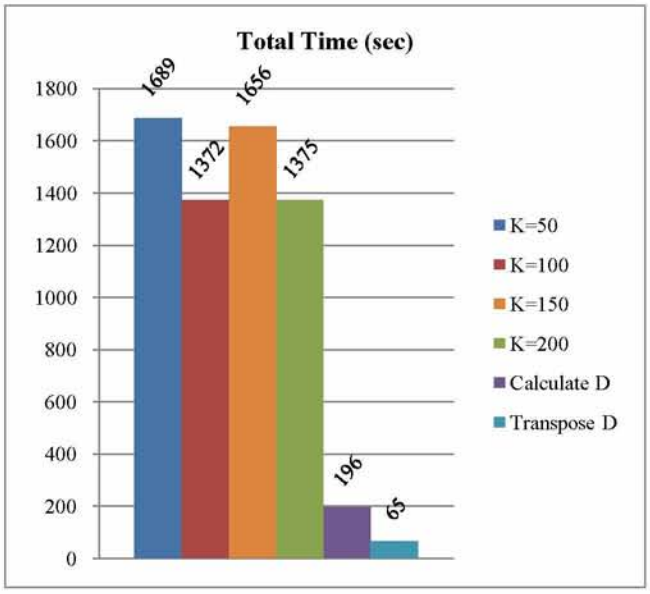
Για τα πειράματα χρησιμοποιήθηκαν συλλογές αρχείων κειμένου (txt) από την ιστοσελίδα www.textfiles.com. Στο πιο κάτω πίνακα φαίνονται τα μεγέθη των συλλογών καθώς και οι διαστάσεις του πίνακα γειτνίασης. Σημειώνεται ότι ο αριθμός των στηλών είναι ο ίδιος με τον αριθμό των εγγράφων στην κάθε συλλογή.

	Μέγεθος (Megabytes)	Γραμμές	Στήλες	Μη μηδενικά στοιχεία
Συλλογή 1	20.2	153078	46	262243
Συλλογή 2	43.8	152213	43	468476
Συλλογή 3	74.4	152213	68	839463
Συλλογή 4	156.7	152213	156	1678926

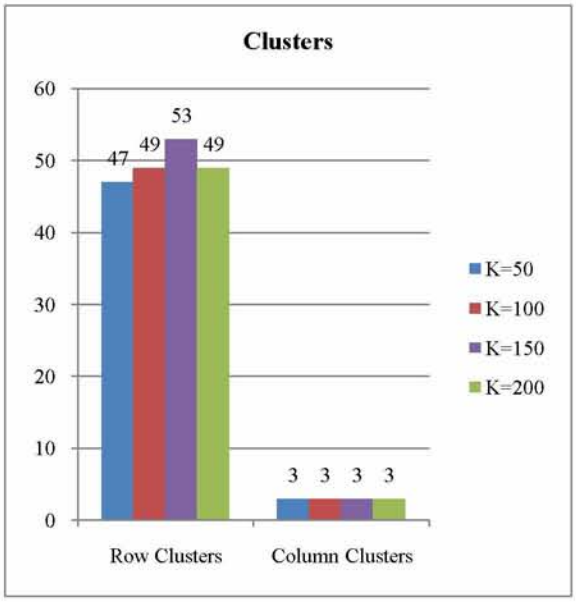
ΑΠΟΤΕΛΕΣΜΑΤΑ ΑΠΟ ΣΥΛΛΟΓΗ 1



Διάγραμμα 1. Χρόνος εκτέλεσης ανά επανάληψη για K=50,100,150,200

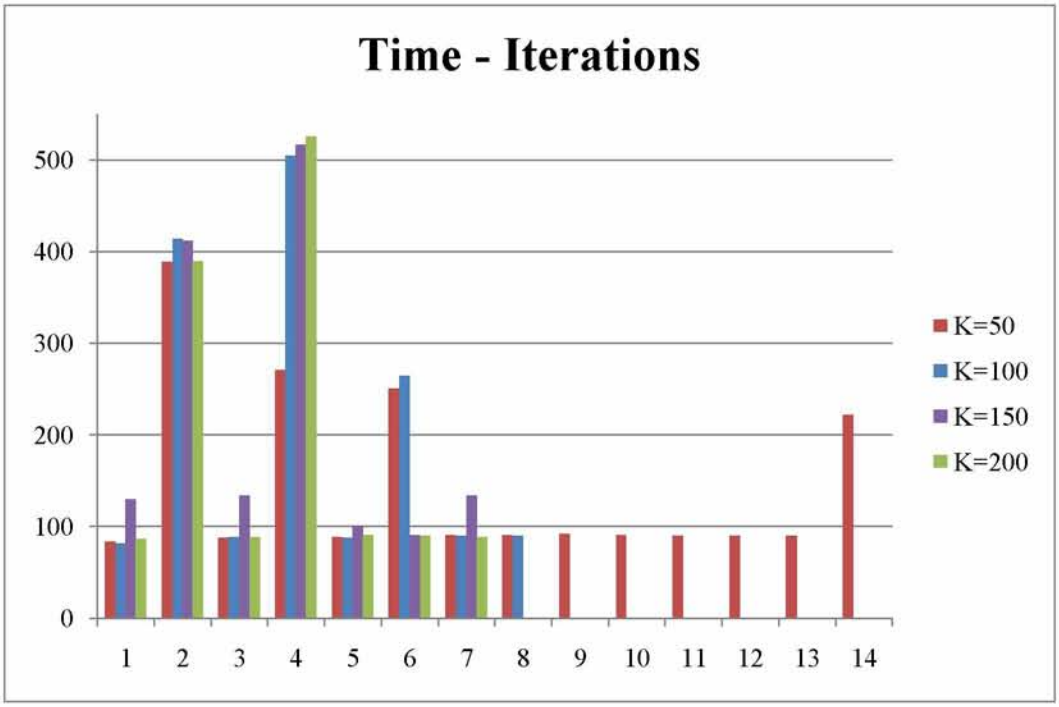


Διάγραμμα 2. Συνολικός Χρόνος Εκτέλεσης

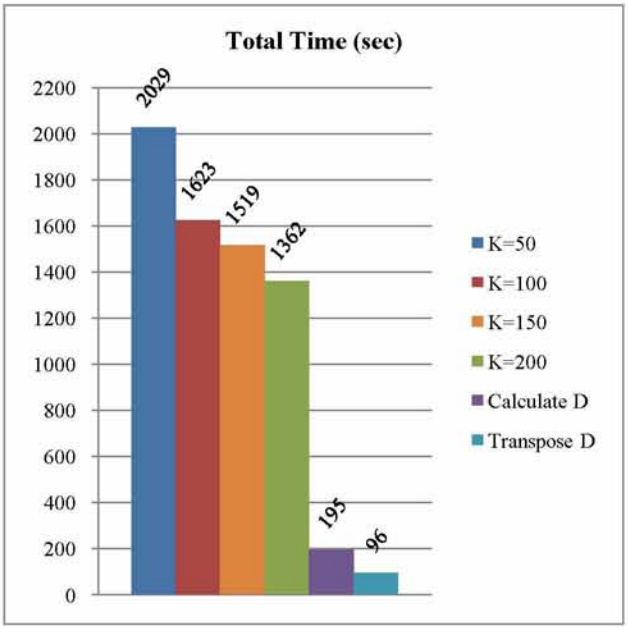


Διάγραμμα 3. Αριθμός Cluster

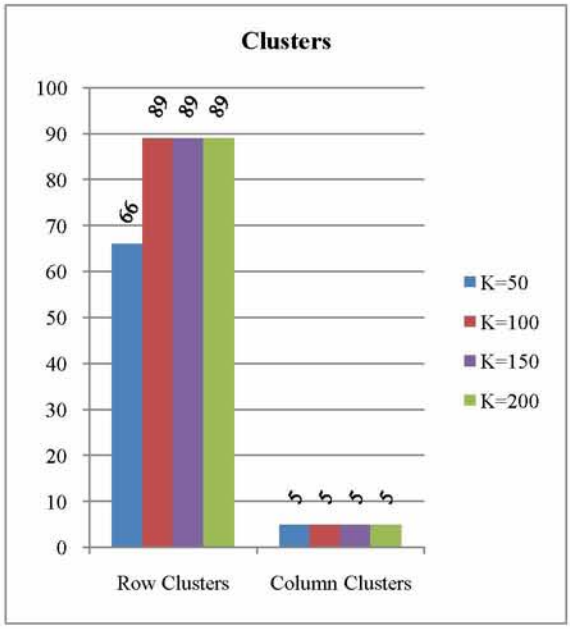
ΑΠΟΤΕΛΕΣΜΑΤΑ ΑΠΟ ΣΥΛΛΟΓΗ 2



Διάγραμμα 4. Χρόνος εκτέλεσης ανά επανάληψη για K=50,100,150,200

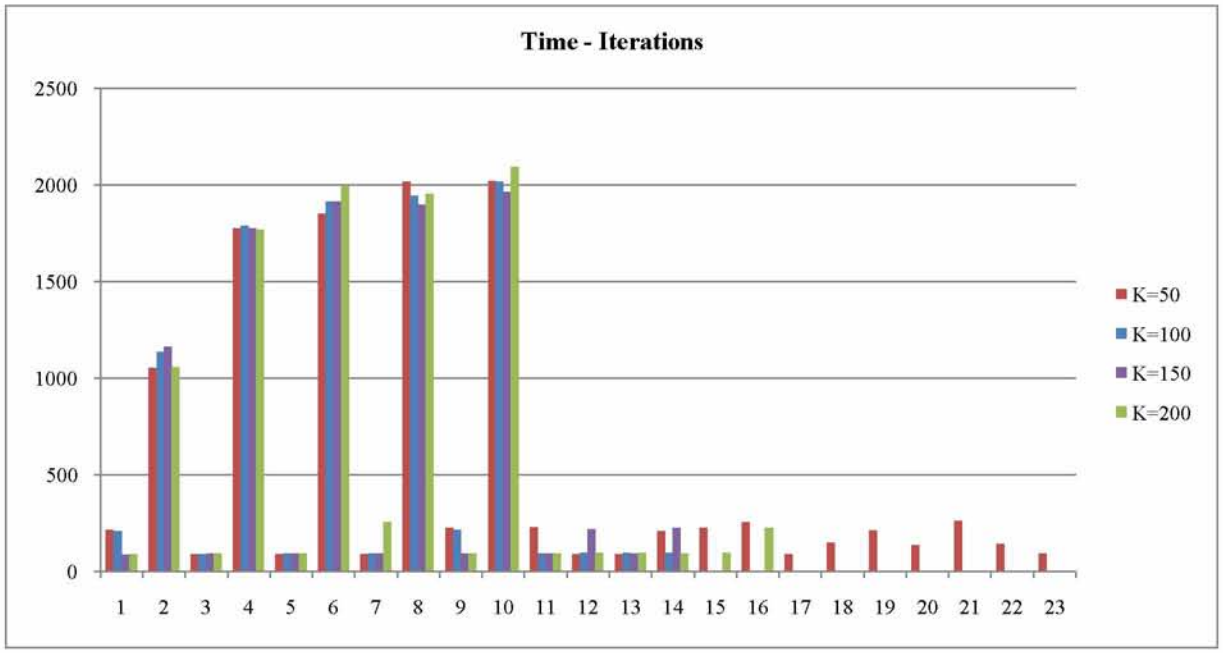


Διάγραμμα 5. Συνολικός Χρόνος Εκτέλεσης

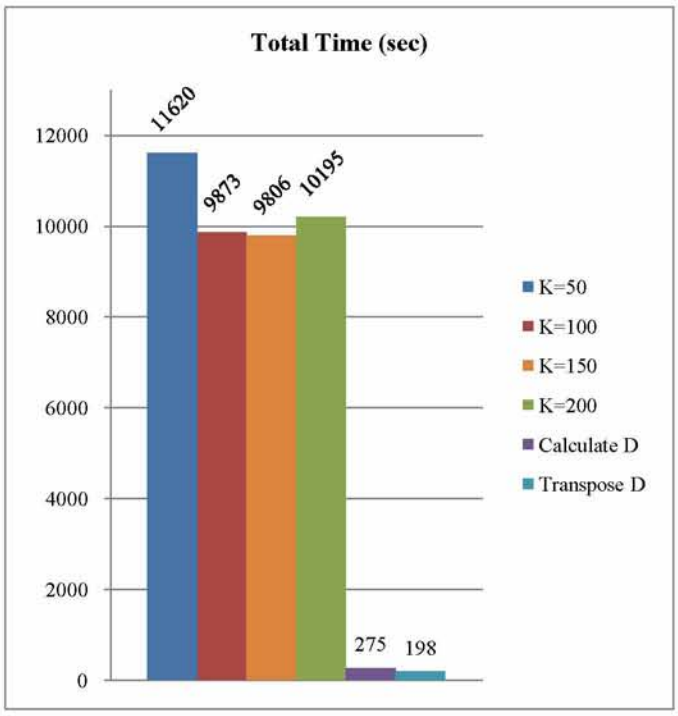


Διάγραμμα 6. Αριθμός Cluster

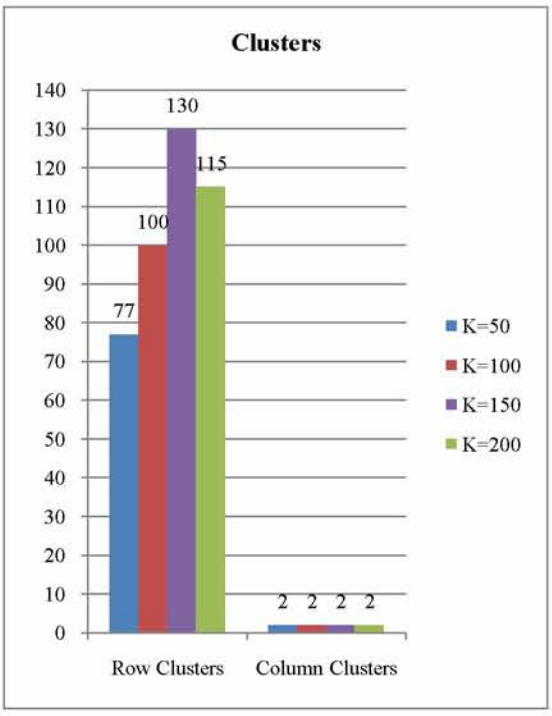
ΑΠΟΤΕΛΕΣΜΑΤΑ ΑΠΟ ΣΥΛΛΟΓΗ 3



Διάγραμμα 7. Χρόνος εκτέλεσης ανά επανάληψη για K=50,100,150,200

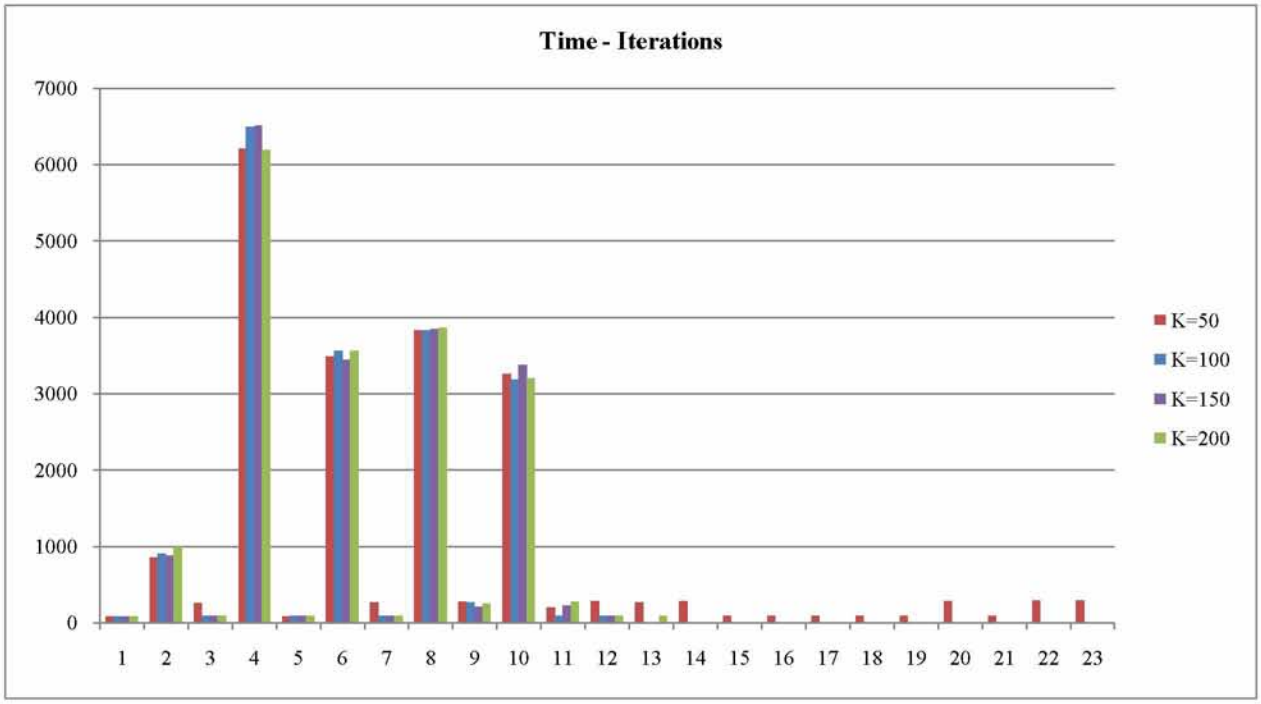


Διάγραμμα 8. Συνολικός Χρόνος Εκτέλεσης

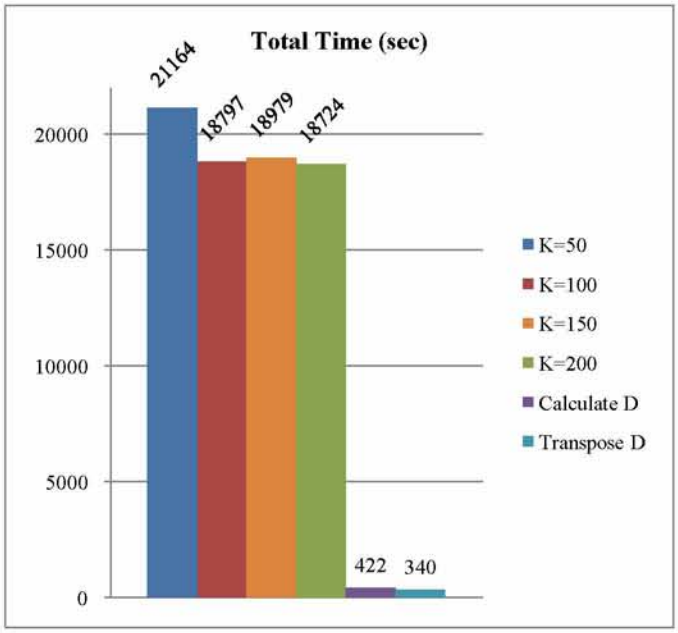


Διάγραμμα 9. Αριθμός Cluster

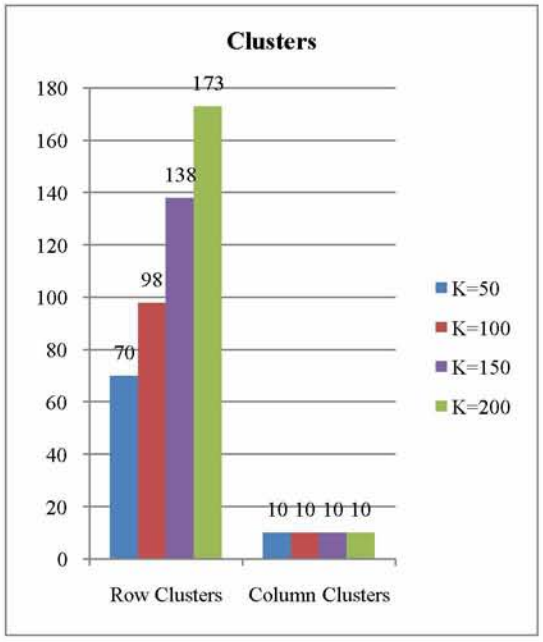
ΑΠΟΤΕΛΕΣΜΑΤΑ ΑΠΟ ΣΥΛΛΟΓΗ 4



Διάγραμμα 10. Χρόνος εκτέλεσης ανά επανάληψη για K=50,100,150,200

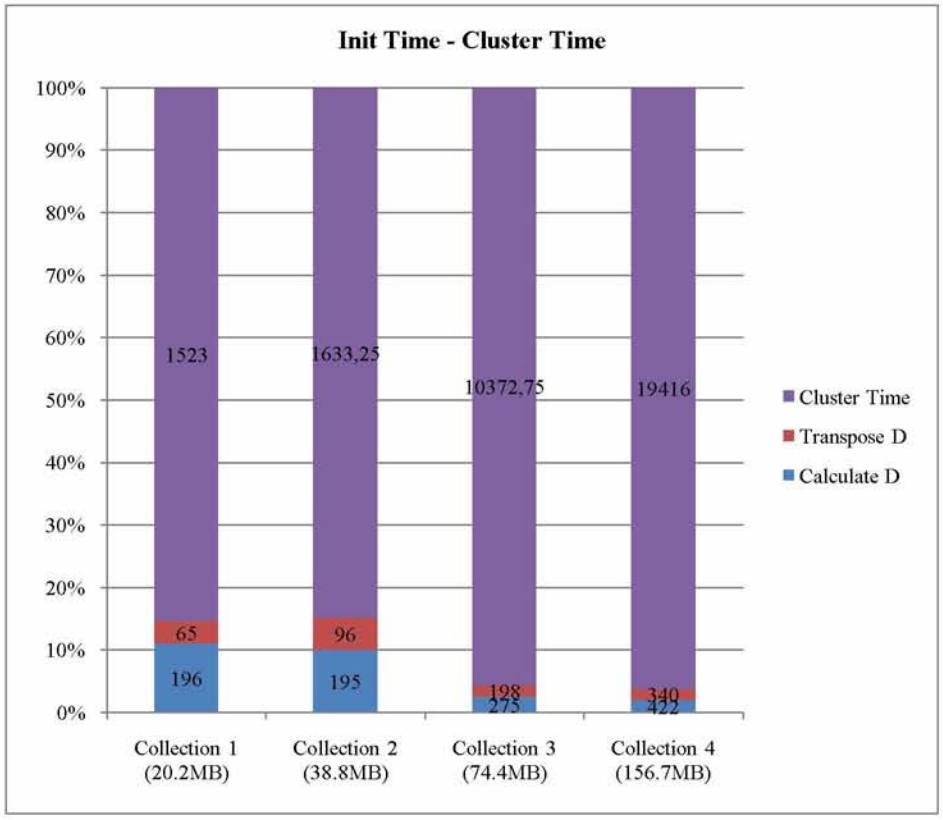


Διάγραμμα 11. Συνολικός Χρόνος Εκτέλεσης



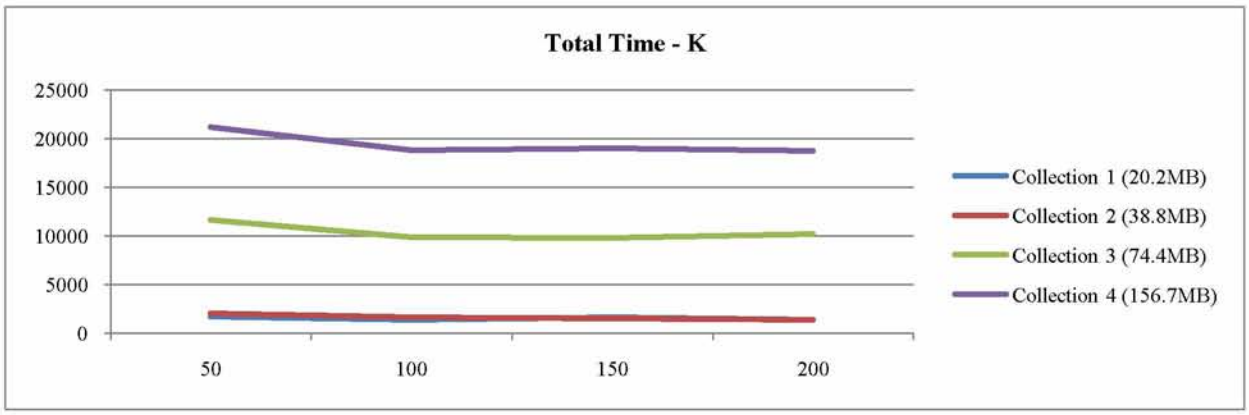
Διάγραμμα 12. Αριθμός Cluster

ΣΥΓΚΡΙΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ



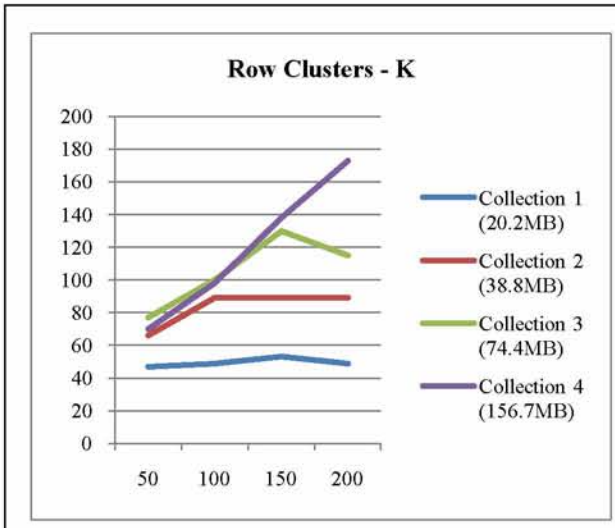
Διάγραμμα 13. Χρόνος αρχικοποίησης ως προς τον χρόνο ομαδοποίησης

Από το διάγραμμα φαίνεται ότι ο χρόνος για τον υπολογισμό του πίνακα D και του αντίστροφου του είναι πολύ μικρότερος από τον χρόνο που χρειαζόμαστε για να ομαδοποιήσουμε την συλλογή. Στις δυο πρώτες συλλογές το στάδιο της αρχικοποίησης χρειάστηκε περίπου το 15% του συνολικού χρόνου ενώ στις δύο τελευταίες, που ήταν αρκετά μεγαλύτερες, χρειάστηκε μόλις το 5% του συνολικού χρόνου.

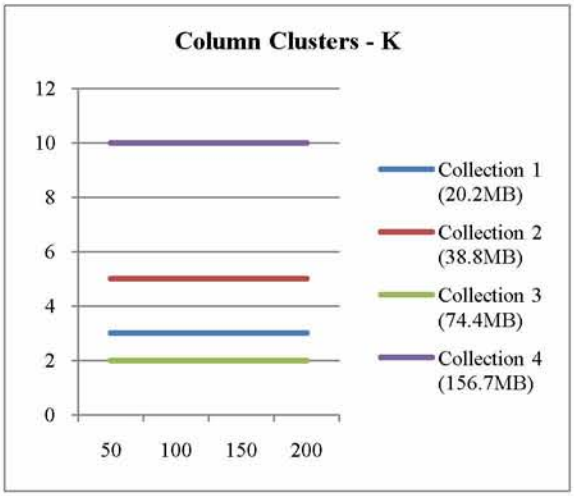


Διάγραμμα 14. Συνολικός χρόνος ως προς K

Από το πιο πάνω διάγραμμα παρατηρούμε ότι η επιλογή του αρχικού αριθμού cluster (k) μπορεί να επηρεάσει αρκετά τον χρόνο εκτέλεσης, ειδικά εάν η συλλογή μας είναι σχετικά μεγάλη.

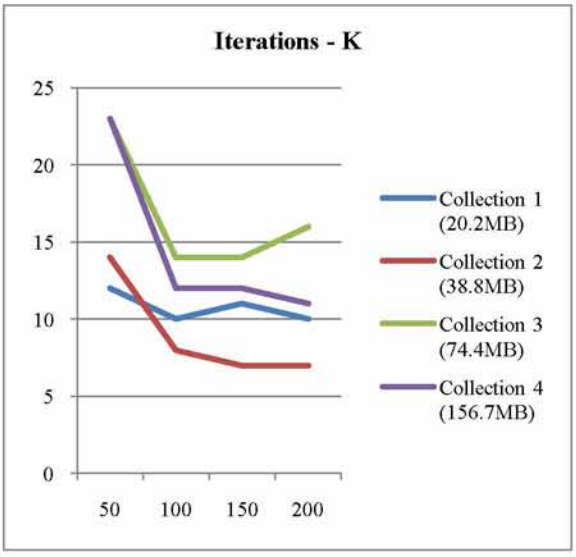


Διάγραμμα 15. Αριθμός cluster γραμμών ως προς K

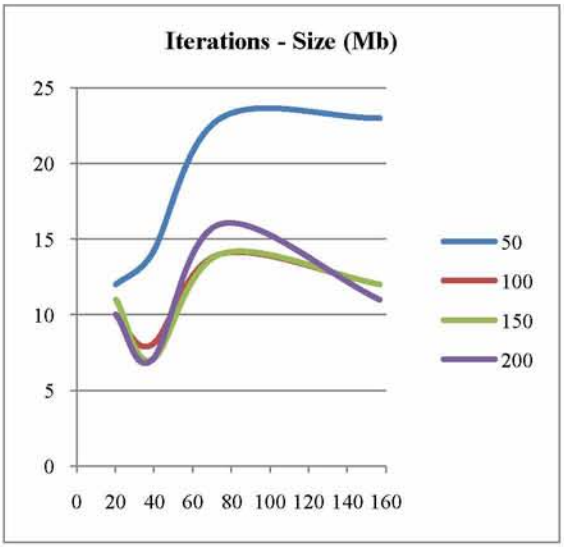


Διάγραμμα 16. Αριθμός cluster στηλών ως προς K

Στο διάγραμμα αυτό βλέπουμε ότι ο αριθμός των cluster σε κάποιες περιπτώσεις αλλάζει ανάλογα με την επιλογή του αρχικού αριθμού K. Για τις γραμμές ο τελικός αριθμός είναι κοντά στον αριθμό K αλλά για τις στήλες υπάρχει μεγάλη διαφορά. Αυτό οφείλεται στο γεγονός ότι σε όλες τις συλλογές που χρησιμοποιήθηκαν ο αριθμός των γραμμών ήταν τάξης μεγαλύτερος από τον αριθμό των στηλών και σε αρκετές περιπτώσεις ο αρχικός αριθμός K ήταν μεγαλύτερος από τον συνολικό αριθμό στηλών στο πίνακα D.



Διάγραμμα 17. Αριθμός επαναλήψεων ως προς K

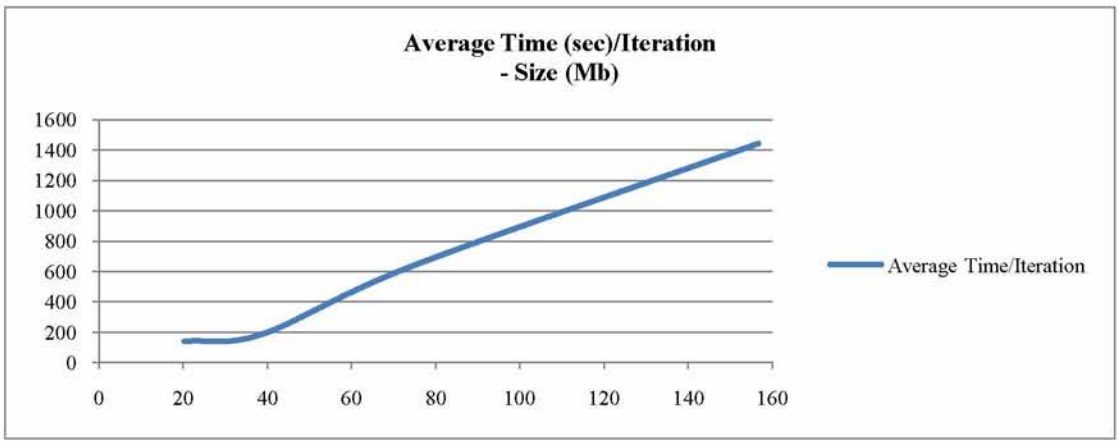


Διάγραμμα 18. Αριθμός επαναλήψεων ως προς το μέγεθος

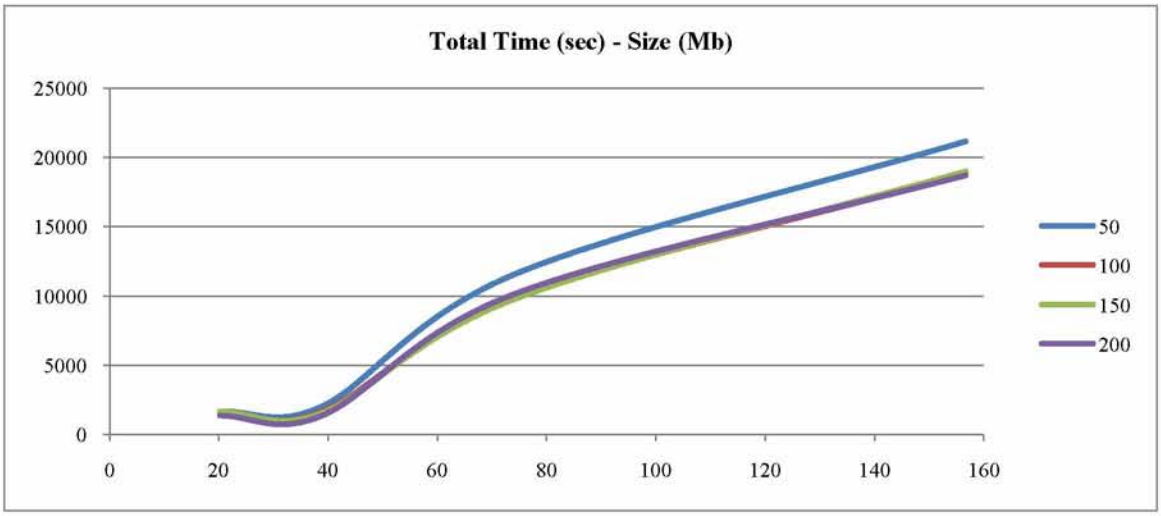
Από το αριστερά διάγραμμα παρατηρούμε ότι ο αρχικός αριθμός K επηρεάζει και τον αριθμό επαναλήψεων που θα χρειαστούν για να συγκλίνει ο αλγόριθμος. Σε αντίθεση με το K,

από το δεξιά διάγραμμα, το μέγεθος της συλλογής δεν επηρεάζει τον αριθμό των επαναλήψεων που θα χρειαστούν. Φαίνεται ότι καθώς το μέγεθος της συλλογής αυξάνει υπάρχει μια μικρή μείωση στον αριθμό των επαναλήψεων που χρειάζονται.

Σε καμία περίπτωση δεν χρειάστηκαν περισσότερες από 23 επαναλήψεις ανεξάρτητα από το μέγεθος της συλλογής.



Διάγραμμα 19. Μέσος χρόνος ανά επανάληψη ως προς το μέγεθος της συλλογής



Διάγραμμα 20. Συνολικός χρόνος εκτέλεσης ως προς το μέγεθος της συλλογής ανά K

Από τα δυο τελευταία διαγράμματα επιβεβαιώνεται ότι η επιλογή του αρχικού αριθμού K μπορεί να επηρεάσει ελάχιστα τον χρόνο εκτέλεσης. Επιπλέον βλέπουμε ότι στην δεύτερη συλλογή, παρόλο που έχει διπλάσιο μέγεθος από την πρώτη, ο χρόνος που χρειάστηκε είναι μικρότερος. Αυτό οφείλεται στο γεγονός ότι η δεύτερη συλλογή περιείχε λιγότερα αρχεία μεγαλύτερου μεγέθους και περίπου 1000 λέξεις λιγότερες (τα μη μηδενικά στοιχεία ήταν περίπου διπλάσια στην δεύτερη συλλογή). Το πιο σημαντικό που βλέπουμε σε αυτά τα διαγράμματα είναι η γραμμική συμπεριφορά του προγράμματος.

ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

ΣΥΜΠΕΡΑΣΜΑΤΑ

Τα πιο πάνω πειράματα κρίνονται ικανοποιητικά για να καταλήξουμε σε κάποια συμπεράσματα όσο αφορά τον χρόνο εκτέλεσης, το μέγεθος της συλλογής, τον αριθμό των cluster και τον αρχικό αριθμό κέντρων (K) που επιλέγεται από τον χρήστη.

Όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο, από τα αποτελέσματα των πειραμάτων μπορούμε να συμπεράνουμε τα εξής:

- Ο αριθμός των επαναλήψεων δεν είναι γνωστός εκ των προτέρων και εξαρτάται από την επιλογή, καθώς και τον αριθμό, των αρχικών κέντρων.
- Ο αριθμός των επαναλήψεων δεν είναι μεγάλος (περίπου 15) και δεν επηρεάζεται από το μέγεθος των δεδομένων.
- Ο τελικός αριθμός των clusters εξαρτάται από την επιλογή, καθώς και τον αριθμό, των αρχικών κέντρων.
- Ο χρόνος αρχικοποίησης (υπολογισμού του πίνακα D και του αντίστροφου του) είναι πολύ μικρότερος από τον χρόνο που χρειάζεται για την ομαδοποίηση καταλαμβάνοντας, κατά μέσο όρο, το 10% του συνολικού χρόνου.
- Ο χρόνος εκτέλεσης επηρεάζεται πολύ από τις διαστάσεις του πίνακα D. Όσο πιο τετραγωνικός είναι τόσο λιγότερος χρόνος χρειάζεται ανεξάρτητα από τον αριθμό των μη μηδενικών στοιχείων.
- Ο αλγόριθμος κλιμακώνει γραμμικά ως προς το μέγεθος των δεδομένων (σημειώνεται ότι το Hadoop ήταν σε pseudo-distributed mode).

Επιπλέον από τις εκτελέσεις παρατηρήθηκε ότι ο χρόνος που χρειάζονται οι reducers είναι αρκετά μεγαλύτερος από τον χρόνο που χρειάζονται οι mappers. Εντούτοις περισσότεροι reducers δεν έδιδαν μικρότερο χρόνο εκτέλεσης καθώς ο συνολικός χρόνος ήταν πάντα ίσος με το χρόνο που χρειαζόταν ο πιο αργός από αυτούς.

ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

Είναι βέβαιο ότι η βελτιστοποίηση και ανάπτυξη των πιο πάνω δεν απορρίπτεται. Παρόλο που ο πιο πάνω κώδικας είναι πλήρης, και δίδει σωστά αποτελέσματα, υπάρχουν δυνατότητες του Hadoop που δεν έχουν χρησιμοποιηθεί ούτως ώστε να ελέχθη εάν με αυτές τις δυνατότητες μπορούμε να επιτύχουμε ακόμα καλύτερους χρόνους εκτέλεσης. Για παράδειγμα μπορούμε να κάνουμε χρήση κάποιου κώδικα συμπίεσης των δεδομένων ούτως ώστε να έχουμε χαμηλότερη χρήση πόρων του δικτύου ή να χρησιμοποιήσουμε κάποιες έτοιμες βιβλιοθήκες

(π.χ.GNU Trove), που θα έχουν πιο αποδοτικό κώδικα, για την διαχείριση των αραιών διανυσμάτων.

Μια επιπλέον επέκταση θα ήταν να μπορούσαμε να ανανεώσουμε τα αποτελέσματα χωρίς να χρειάζεται υπολογισμός από την αρχή όταν προστεθεί ή αφαιρεθεί κάποιο έγγραφο στην συλλογή.

Μπορούμε επίσης να χρησιμοποιήσουμε τα πιο πάνω ούτως ώστε να βρούμε ίδια έγραφα σε συλλογές συγκρίνοντας μεταξύ τους τα έγγραφα που είναι στα ίδια clusters (καθώς δύο ίδια αρχεία θα βρίσκονται στην ίδια ομάδα).

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] "Welcome to Apache Hadoop!", http://hadoop.apache.org

[2] "Wikipedia, the free encyclopedia", http://en.wikipedia.org

[3] Dean, Jeffrey, Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", Google Inc, 2004

[4] Spiros Papadimitriou, Jimeng Sun, "DisCo: Distributed Co-clustering with Map-Reduce", ICDM 2008

[5] Inderjit S. Dhillon, Subramanyam Mallela Dharmendra S. Modha, "Information-Theoretic Co-clustering", KDD, 2003

[6] Dilpesh Shrestha, "Text mining with Lucene and Hadoop: Document clustering with feature extraction", Wakhok University, 2009

[7] Kenneth Heafield, "Hadoop Design and k-Means Clustering", Google Inc, January 2008

[8] Weizhong Zhao, Huifang Ma, Qing He, "Parallel K-Means Clustering Based on MapReduce", 2009

[9] Michael Steinbach, George Karypis, Vipin Kumar, "A Comparison of Document Clustering Techniques", University of Minnesota, 1999

[10] Jian Wan, Wenming Yu, Xianghua Xu, "Design and Implement of Distributed Document Clustering Based on MapReduce", Grid and Services Computing Lab School of Computer Science and Technology, Hangzhou Dianzi University, December 2009

[11] Yifeng Liu, Lucio Gutierrez, Amirhossein Firouzmanesh, Xiaoyu Shi, Xiaodi Ke, Md Solimul Chowdhury, "Parallel Document Clustering with Hadoop on Amazon Elastic Computing Cloud", CMPUT 681 Project, December 2009

[12] Jimmy Lin, Chris Dyer, "Data-Intensive Text Processing with MapReduce", University of Maryland, 2010

[13] Jason Venner, "Pro Hadoop", Apress June 2009

[14] Tom White, "Hadoop: The definitive guide – MapReduce for the cloud", O'Reilly Media, May 2009

ΠΑΡΑΡΤΗΜΑ Α – ΚΩΔΙΚΑΣ ΠΑΡΑΔΕΙΓΜΑΤΩΝ

ΚΩΔΙΚΑΣ ΠΑΡΑΔΕΙΓΜΑΤΟΣ 1 – WORD COUNT

```
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class WordCount {

 public static class Map extends MapReduceBase implements
Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            output.collect(word, one);
        }
    }
 }

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

    public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output, Reporter reporter) throws
IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
 }

 public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    conf.setMapperClass(Map.class);
```

```
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);

    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);

    FileInputFormat.setInputPaths(conf, new Path("input"));
    FileOutputFormat.setOutputPath(conf, new Path("output"));

    try {
      JobClient.runJob(conf);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

ΚΩΔΙΚΑΣ ΠΑΡΑΔΕΙΓΜΑΤΟΣ 2 – INVERTED INDEX

```
import java.io.IOException;
import java.util.Iterator;
import java.util.StringTokenizer;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.FileSplit;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class InvertedIndex {

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

    private final static Text word = new Text();
    private final static Text location = new Text();

    public void map(LongWritable key, Text val,
        OutputCollector<Text, Text> output, Reporter reporter)
        throws IOException {

      FileSplit fileSplit = (FileSplit)reporter.getInputSplit();
      String fileName = fileSplit.getPath().getName();
      location.set(fileName);
```

```
      String line = val.toString();
      StringTokenizer itr = new StringTokenizer(line.toLowerCase());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        output.collect(word, location);
      }
    }
  }


  public static class InvertedIndexReducer extends MapReduceBase
      implements Reducer<Text, Text, Text, Text> {

    public void reduce(Text key, Iterator<Text> values,
        OutputCollector<Text, Text> output, Reporter reporter)
        throws IOException {

      boolean first = true;
      StringBuilder toReturn = new StringBuilder();
      while (values.hasNext()){
        if (!first)
          toReturn.append(", ");
        first=false;
        toReturn.append(values.next().toString());
      }

      output.collect(key, new Text(toReturn.toString()));
    }
  }

  public static void main(String[] args) {
    JobConf conf = new JobConf(InvertedIndex.class);

    conf.setJobName("InvertedIndex");

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(Text.class);

    FileInputFormat.addInputPath(conf, new Path("input"));
    FileOutputFormat.setOutputPath(conf, new Path("output"));

    conf.setMapperClass(InvertedIndexMapper.class);
    conf.setReducerClass(InvertedIndexReducer.class);

    try {
      JobClient.runJob(conf);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

}
```

ΠΑΡΑΡΤΗΜΑ Β – ΕΝΤΟΛΕΣ ΧΡΗΣΗΣ HADOOP

ΕΠΙΣΚΟΠΙΣΗ

Όλες οι εντολές εκτελούνται από το bin/hadoop script. Εάν εκτελέσουμε το hadoop script χωρίς παραμέτρους βλέπουμε την περιγραφή όλων των εντολών.

Χρήση: hadoop [--config confdir] [COMMAND] [GENERIC_OPTIONS] [COMMAND_OPTIONS]
Hadoop has an option parsing framework that employs parsing generic options as well as running classes.

COMMAND_OPTION	Περιγραφή
--config confdir	Αλλάζει το φάκελο με τις ρυθμίσεις από ${HADOOP_HOME}/conf σε confdir.
GENERIC_OPTIONS	Συνηθισμένες εντολές που υποστηρίζονται από πολλές εντολές.
COMMAND COMMAND_OPTIONS	Διάφορες εντολές που περιγράφονται πιο κάτω. Οι εντολές χωρίζονται σε «Εντολές Χρήστη» και «Εντολές Διαχειριστή».

ΓΕΝΙΚΕΣ ΕΠΙΛΟΓΕΣ - GENERIC OPTIONS

Οι πιο κάτω επιλογές υποστηρίζονται από dfsadmin, fs, fsck και job. Οι εφαρμογές πρέπει να υλοποιούν την διεπαφή Tool για να υποστηρίξουν γενικές επιλογές (GenericOptions).

GENERIC_OPTION	Περιγραφή
-conf <configuration file>	Καθορίζει το αρχείο με τις ρυθμίσεις μιας εφαρμογής.
-D <property=value>	Αλλάζει την τιμή μιας ιδιότητας (property).
-fs <local|namenode:port>	Καθορίζει ένα namenode.
-jt <local|jobtracker:port>	Καθορίζει ένα job tracker. Εφαρμόζεται μόνο σε job.
-files <comma separated list of files>	Καθορίζει κάποια αρχεία, χωρισμένα με κόμμα, για να αντιγραφούν στο cluster. Εφαρμόζεται μόνο σε job.
-libjars <comma separated list of jars>	Καθορίζει κάποια αρχεία jar, χωρισμένα με κόμμα, για να συμπεριληφθούν στο classpath. Εφαρμόζεται μόνο σε job.
-archives <comma separated list of archives>	Καθορίζει κάποια συμπιεσμένα αρχεία, χωρισμένα με κόμμα, για να αποσυμπιεστούν. Εφαρμόζεται μόνο σε job.

ΕΝΤΟΛΕΣ ΧΡΗΣΤΩΝ - USER COMMANDS

Εντολές που χρησιμοποιούνται από όλους τους χρήστες ενός hadoop cluster.

ARCHIVE

Δημιουργία ενός συμπιεσμένου αρχείου hadoop.
Χρήση: hadoop archive -archiveName NAME <src>* <dest>

-archiveName NAME	Όνομα του συμπιεσμένου αρχείου που θα δημιουργηθεί.
src	Διαδρομή σε κάποια αρχεία. Δουλεύει και με κανονικές εκφράσεις.
dest	Φάκελος που θα αποθηκευτεί το νέο αρχείο.

DISTCP

Αντιγράφει αναδρομικά αρχεία ή φακέλους.
Χρήση: hadoop distcp <srcurl> <desturl>

srcurl	Source Url
desturl	Destination Url

FS

Πρόσβαση στο σύστημα αρχείων. Όλες οι εντολές παίρνουν παραμέτρους διαδρομές της *scheme://autority/path*. Για το HDFS το scheme είναι *hdfs*, και για το τοπικό σύστημα αρχείων είναι *file*. Το scheme και το authority δεν είναι υποχρεωτικά. Οι πιο πολλές εντολές συμπεριφέρονται όπως και οι αντίστοιχες εντολές σε Unix. Πληροφορίες για σφάλματα γράφονται στο *stderr* και η έξοδος στο *stdout*.

Χρήση: hadoop fs [GENERIC_OPTIONS] [COMMAND_OPTIONS]

Τα COMMAND_OPTIONS είναι:

CAT

Χρήση: `hadoop fs -cat URI [URI …]`
Αντιγράφει στο *stdout* τα αρχεία που καθορίζονται από τα URIs.

CHGRP

Χρήση: `hadoop fs -chgrp [-R] GROUP URI [URI …]`
Αλλαγή της ομάδας που έχουν πρόσβαση στα αρχεία. Με την παράμετρο `-R`, οι αλλαγές εφαρμόζονται αναδρομικά. Ο χρήστης πρέπει να είναι ο ιδιοκτήτης των αρχείων ή super-user.

CHMOD

Χρήση: `hadoop fs -chmod [-R] <MODE[,MODE]... | OCTALMODE> URI [URI …]`

Αλλαγή των δικαιωμάτων πρόσβασης στα αρχεία. Με την παράμετρο -R, οι αλλαγές εφαρμόζονται αναδρομικά. Ο χρήστης πρέπει να είναι ο ιδιοκτήτης των αρχείων ή super-user.

CHOWN

Χρήση: `hadoop fs -chown [-R] [OWNER][:[GROUP]] URI [URI ]`

Αλλαγή του ιδιοκτήτη των αρχείων. Με την παράμετρο -R, οι αλλαγές εφαρμόζονται αναδρομικά. Ο χρήστης πρέπει να είναι super-user.

COPYFROMLOCAL

Χρήση: `hadoop fs -copyFromLocal <localsrc> URI`

Παρόμοια με την put, με την διαφορά ότι το src πρέπει να είναι στο τοπικό σύστημα αρχείων.

COPYTOLOCAL

Χρήση: `hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst>`

Παρόμοια με την get, με την διαφορά ότι το dst πρέπει να είναι στο τοπικό σύστημα αρχείων.

COUNT

Χρήση: `hadoop fs -count [-q] <paths>`

Μετρά τον αριθμό των φακέλων, αρχείων και τα bytes στις διαδρομές που καθορίζονται στο *paths*. Οι στήλες εξόδου είναι:

`DIR_COUNT, FILE_COUNT, CONTENT_SIZE FILE_NAME.`

Με την παράμετρο `-q` οι στήλες εξόδου είναι:

`QUOTA, REMAINING_QUATA, SPACE_QUOTA, REMAINING_SPACE_QUOTA, DIR_COUNT, FILE_COUNT, CONTENT_SIZE, FILE_NAME.`

CP

Χρήση: `hadoop fs -cp URI [URI …] <dest>`

Αντιγραφή αρχείων από τα URIs στο dest. Για πολλά URI το dest πρέπει να είναι φάκελος.

DU

Χρήση: `hadoop fs -du URI [URI …]`

Παρουσιάζει το συνολικό μέγεθος των αρχείων.

DUS

Χρήση: `hadoop fs -dus <args>`

Παρουσιάζει συνοπτικά το μέγεθος των αρχείων.

EXPUNGE

Χρήση: `hadoop fs -expunge`

Άδειασμα του καλάθου αχρήστων.

GET

Χρήση: `hadoop fs -get [-ignorecrc] [-crc] <src> <localdst>`

Αντιγραφή αρχείων στο τοπικό σύστημα αρχείων. Εάν τα αρχείαν αποτύχουν τον έλεγχο CRC μπορούν να αντιγραφούν με την χρήση της παραμέτρου `-ignorecrc`. Με την παράμετρο `-crc` αντιγράφονται τα αρχεία και τα CRCs.

GETMERGE

`Χρήση: hadoop fs -getmerge <src> <localdst> [addnl]`

Όλα τα αρχεία στο src ενοποιούνται και αντιγράφονται στο αρχείο localdst. Με την επιλογή `addnl` στο τέλος κάθε αρχείου προστίθεται ο χαρακτήρας newline.

LS

`Χρήση: hadoop fs -ls <args>`

Εάν το args καθορίζει κάποιο αρχείο η έξοδος είναι στατιστικά για αυτό το αρχείο της εξής μορφής:

`permissions number_of_replicas userid groupid filesize modification_date modification_time filename`

Εάν το args καθορίζει κάποιο φάκελο τότε επιστρέφεται η λίστα με τα αρχεία σε αυτό το φάκελο. Η έξοδος είναι της μορφής:

`permissions userid groupid modification_date modification_time dirname`

LSR

`Χρήση: hadoop fs -lsr <args>`

Αναδρομική ls. Ίδια με την `ls -R` των Unix.

MKDIR

`Χρήση: hadoop fs -mkdir <paths>`

Δημιουργία φακέλων. Ίδια με την *mkdir –p* των Unix.

MOVEFROMLOCAL

`Χρήση: dfs -moveFromLocal <localsrc> <dst>`

Παρόμοια με την put, με την διαφορά ότι το `localsrc` διαγράφεται μετά την αντιγραφή.

MOVETOLOCAL

`Χρήση: hadoop fs -moveToLocal [-crc] <src> <dst>`

Παρουσιάζει το μήνυμα "Not implemented yet".

MV

`Χρήση: hadoop fs -mv URI [URI …] <dest>`

Μεταφορά αρχείων από τα URIs στο *dest*. Δεν υποστηρίζει μεταφορά μεταξύ διαφορετικών συστημάτων αρχείων.

PUT

`Χρήση: hadoop fs -put <localsrc> ... <dst>`

Αντιγράφει τα αρχεία από το τοπικό σύστημα αρχείων στο dst. Μπορεί να διαβάσει είσοδο από το *stdin* π.χ. `hadoop fs -put - hdfs://nn.example.com/hadoop/hadoopfile`

RM
Χρήση: `hadoop fs -rm URI [URI …]`
Διαγραφή αρχείων ή κενών φακέλων.

RMR
Χρήση: `hadoop fs -rmr URI [URI …]`
Αναδρομική rm για διαγραφή όχι κενών φακέλων.

SETREP
Χρήση: `hadoop fs -setrep [-R] <path>`
Αλλάζει το replication factor ενός αρχείου. Με την παράμετρο -R εφαρμόζεται αναδρομικά.

STAT
Χρήση: `hadoop fs -stat URI [URI …]`
Επιστρέφει πληροφορίες για τα URIs.

TAIL
Χρήση: `hadoop fs -tail [-f] URI`
Επιστρέφει το τελευταίο kilobyte του αρχείου στο stdout. Η παράμετρος -f είναι ίδια με την εντολή στα Unix.

TEST
Χρήση: `hadoop fs -test -[ezd] URI`
Επιλογές:

-e	Επιστρέφει 0 εάν το αρχείο υπάρχει.
-z	Επιστρέφει 0 εάν το αρχείο είναι μηδενικού μεγέθους.
-d	Επιστρέφει 1 εάν το αρχείο είναι φάκελος αλλιώς 0.

TEXT
Χρήση: `hadoop fs -text <src>`
Παίρνει σαν είσοδο αρχεία της μορφής zip ή TextRecordInputStream και δίνει την έξοδο σε κείμενο txt.

TOUCHZ
Χρήση: `hadoop fs -touchz URI [URI …]`
Δημιουργία μηδενικού μεγέθους αρχεία.

FSCK
Έλεγχος του συστήματος αρχείων HDFS.
Χρήση: hadoop fsck [GENERIC_OPTIONS] <path> [-move | -delete | -openforwrite] [-files [-blocks [-locations | -racks]]]

<path>	Εκκίνηση του έλεγχου από αυτό το path.
-move	Μεταφορά των corrupted αρχείων στο /lost+found
-delete	Διαγραφή corrupted αρχείων.
-openforwrite	Τυπώνει ποια αρχεία είναι ανοικτά για εγγραφή.
-files	Τυπώνει ποια αρχεία ελέγχονται.
-blocks	Τυπώνει αναφορά για τα blocks.
-locations	Τυπώνει την τοποθεσία κάθε block.
-racks	Τυπώνει την τοπολογία του δικτύου για τους data-nodes.

JAR

Εκτέλεση ενός αρχείου jar.
Χρήση: hadoop jar <jar> [mainClass] args...

Χρησιμοποιείται για την εκτέλεση προγραμμάτων. Επιπλέον για εκτέλεση προγραμμάτων που χρησιμοποιούν το Hadoop Streaming API. Για περισσότερα σχετικά με το Hadoop Streaming API βλέπε Παράρτημα Γ.

JOB

Αλληλεπίδραση με τις εργασίες Map Reduce.
Χρήση: hadoop job [GENERIC_OPTIONS] [-submit <job-file>] | [-status <job-id>] | [-counter <job-id> <group-name> <counter-name>] | [-kill <job-id>] | [-events <job-id> <from-event-#> <#-of-events>] | [-history [all] <jobOutputDir>] | [-list [all]] | [-kill-task <task-id>] | [-fail-task <task-id>] | [-set-priority <job-id> <priority>]

-submit <job-file>	Τοποθέτηση μιας εργασίας για εκτέλεση
-status <job-id>	Τυπώνει το ποσοστό ολοκλήρωσης των map και reduce καθώς και όλους τους μετρητές.
-counter <job-id> <group-name> <counter-name>	Τυπώνει την τιμή του μετρητή.
-kill <job-id>	Σκοτώνει την εργασία.
-events <job-id> <from-event-#> <#-of-events>	Τυπώνει λεπτομέρειες για τα events που πήρε ο jobtracker στο εύρος που δίδεται.
-history <jobOutputDir> -history [all] <jobOutputDir>	Τυπώνει λεπτομέρειες για τις αποτυχημένες (failed, killed) map και reduce διεργασίες. Με την επιλογή [all] παρουσιάζονται πληροφορίες και για τις επιτυχημένες διεργασίες.
-list -list [all]	-list παρουσιάζει τις εργασίες που δεν έχουν ακόμα τελειώσει. -list all παρουσιάζει όλες τις εργασίες.
-kill-task <task-id>	Τερματίζει την συγκεκριμένη διεργασία. Δεν συμπεριλαμβάνεται στις αποτυχημένες προσπάθειες (failed attempts).
-fail-task <task-id>	Τερματίζει την συγκεκριμένη διεργασία. Συμπεριλαμβάνεται στις αποτυχημένες προσπάθειες (failed attempts).
-set-priority <job-id> <priority>	Αλλαγή της προτεραιότητας της συγκεκριμένης εργασίας. Οι επιτρεπόμενες τιμές είναι: VERY_HIGH, HIGH, NORMAL,LOW, VERY_LOW

PIPES

Εκτέλεση μιας εργασίας με σωλήνες (pipes job).
Χρήση: hadoop pipes [-conf <path>] [-jobconf <key=value>, <key=value>, ...] [-input <path>] [-output <path>] [-jar <jar file>] [-inputformat <class>] [-map <class>] [-partitioner <class>] [-reduce <class>] [-writer <class>] [-program <executable>] [-reduces <num>]

-conf <path>	Ρυθμίσεις για την εργασία
-jobconf <key=value>, <key=value>, ...	Προσθήκη/override ρυθμίσεων για την εργασία.
-input <path>	Φάκελος εισόδου.
-output <path>	Φάκελος εξόδου
-jar <jar file>	Όνομα του αρχείου Jar
-inputformat <class>	InputFormat class
-map <class>	Java Map class
-partitioner <class>	Java Partitioner
-reduce <class>	Java Reduce class
-writer <class>	Java RecordWriter
-program <executable>	Executable URI
-reduces <num>	Number of reduces

QUEUE

Αλληλεπίδραση και εμφάνιση πληροφοριών με την ουρά των εργασιών.
Usage : hadoop queue [-list] | [-info <job-queue-name> [-showJobs]]

-list	Παρουσιάζεται η λίστα με τις ουρές εργασιών πουέχουν ρυθμιστεί στο σύστημα.
-info <job-queue-name> [-showJobs]	Πληροφορίες για την ουρά εργασιών. Με την επιλογή -showJobs παρουσιάζεται μια λίστα με την εργασίες στην συγκεκριμένη ουρά.

VERSION

Εκτυπώνει την έκδοση του hadoop.
Χρήση: hadoop version

CLASSNAME

Το hadoop scriptμπορεί να χρησιμοποιηθεί για να χρησιμοποιήσει οποιαδήποτε κλάση.
Χρήση: hadoop CLASSNAME
Τρέχει την κλάση με όνομα CLASSNAME.

ΕΝΤΟΛΕΣ ΔΙΑΧΕΙΡΙΣΤΩΝ - ADMINISTRATION COMMANDS

Εντολές που χρησιμοποιούνται από τους διαχειριστές ενός Hadoop cluster.

BALANCER

Εκτέλεση της διεργασίας για εξισορρόπηση στο cluster. Ο διαχειριστής μπορεί να σταματήσει την διεργασία εξισορρόπησης με τα πλήκτρα Ctrl-C.
Χρήση: hadoop balancer [-threshold <threshold>]

-threshold <threshold>	Κατώφλι ποσοστού χρησιμοποίησης του δίσκου.

DAEMONLOG

Ανάγνωση/εγραφή του επιπέδου ιστορικού για κάθε δαίμονα.
Χρήση: hadoop daemonlog -getlevel <host:port> <name>
Χρήση: hadoop daemonlog -setlevel <host:port> <name> <level>

-getlevel <host:port> <name>	Εκτύπωση του επιπέδου ιστορικού του δαίμονα στο <host:port>. Εσωτερικά συνδέεται στο http://<host:port>/logLevel?log=<name>
-setlevel <host:port> <name> <level>	Εγγραφή του επιπέδου ιστορικού του δαίμονα στο <host:port>. Εσωτερικά συνδέεται στο http://<host:port>/logLevel?log=<name>

DATANODE

Εκτέλεση ενός HDFS datanode.
Χρήση: hadoop datanode [-rollback]

-rollback	Επαναφορά του datanode σε προηγούμενη έκδοση. Πρέπει να χρησιμοποιηθεί αφού σταματήσει ο datanode και διανεμηθεί η παλιά έκδοση του hadoop.

DFSADMIN

Εκτέλεση ενός HDFS dfsadmin client.
Χρήση: hadoop dfsadmin [GENERIC_OPTIONS] [-report] [-safemode enter | leave | get | wait] [-refreshNodes] [-finalizeUpgrade] [-upgradeProgress status | details | force] [-metasave filename] [-setQuota <quota> <dirname>...<dirname>] [-clrQuota <dirname>...<dirname>] [-help [cmd]]

-report	Αναφορά βασικών πληροφοριών και στατιστικών για το σύστημα αρχείων.
-safemode enter | leave | get | wait	Διαχείριση του Safe mode. Safe mode είναι η κατάσταση στην οποία ένας Namenode: 1. δεν δέχεται εντολές αλλαγής στο name space. 2. δεν αντιγράφει/διαγράφει blocks. Η κατάσταση Safe mode ενεργοποιείται αυτόματα όταν ένας Namenode εκκινεί, και απενεργοποιείται όταν το ρυθμισμένο ελάχιστο ποσοστό blocks ικανοποιεί τη συνθήκη του ελάχιστου αριθμού replication. Η κατάσταση Safe mode μπορεί να ενεργοποιηθεί με εντολή αλλά τότε απενεργοποιείται μόνο με εντολή.
-refreshNodes	Ξαναδιαβάζει τους hosts και ανανεώνει το σύνολο των Datanodes που μπορούν να συνδεθούν με τον Namenode.

-finalizeUpgrade	Τερματισμός αναβάθμισης του HDFS. Οι Datanodes διαγράφουν την προηγούμενη έκδοση των working directories, και στην συνέχεια το ίδιο γίνεται από τον Namenode. Αυτό ολοκληρώνει την διαδικασία αναβάθμισης.
-upgradeProgress status | details | force	Κατάσταση και πληροφορίες για την διαδικασία αναβάθμισης. Μπόρει να γίνει εξαναγκασμός της διαδικασίας να προχωρίσει.
-metasave filename	Αποθήκευση των δομών δεδομένων του κεντρικού Namenode στο <filename>. Ο κατάλογος καθορίζεται από την ιδιότητα hadoop.log.dir. Το <filename> θα περιέχει μια γραμμή για κάθε: 1. Datanodes που μιλούν με τον Namenode. 2. Blocks που περιμένουν να αντιγραφούν. 3. Blocks που αντιγράφονται. 4. Blocks που περιμένουν να διαγραφούν.
-setQuota <quota> <dirname> ...<dirname>	Ρύθμιση της αναλογίας για κάθε <dirname>. Η αναλογία είναι ένας μεγάλος ακέραιος (long integer) που βάζει ένα όριο στον αριθμό ονομάτων στο δένδρο καταλόγων.
-clrQuota <dirname>...<dirname>	Διαγραφή της αναλογίας για κάθε <dirname>.
-help [cmd]	Εκτύπωση βοήθειας για κάθε εντολή.

JOBTRACKER

Εκτέλεση ενός κόμβου σε κατάσταση jobtracker.
Χρήση: hadoop jobtracker

NAMENODE

Εκτέλεση του namenode.
Χρήση: hadoop namenode [-format] | [-upgrade] | [-rollback] | [-finalize] | [-importCheckpoint]

-format	Formats namenode. Εκκινεί τον namenode, τον κάνει format και τον κλείνει.
-upgrade	Χρησιμοποιείται μετά την διανομή μια νέας έκδοσης hadoop.
-rollback	Επαναφορά του namenode σε προηγούμενη έκδοση.
-finalize	Διαγραφή όλων τον προηγούμενων καταστάσεων του συστήματος αρχείων. Μετά την χρήση της δεν μπορεί να χρησιμοποιηθεί η –rollback.
-importCheckpoint	Ανάγνωση από το fs.checkpoint.dir ενός ckeckpoint και αποθήκευση στο τρέχων checkpoint.

SECONDARYNAMENODE

Εκτέλεση ενός δευτερεύον HDFS namenode.
Χρήση: hadoop secondarynamenode [-checkpoint [force]] | [-geteditsize]

-checkpoint [force]	Εγγραφή ενός checkpoint εάν EditLog size >= fs.checkpoint.size. Με την επιλογή -force εξαναγκάζεται η εγγραφή του checkpoint ανεξάρτητα από το μέγεθος του αρχείου αλλαγών (EditLog).
-geteditsize	Εκτύπωση του μεγέθους του αρχείου αλλαγών (EditLog).

TASKTRACKER

Εκτέλεση ενός κόμβου σε κατάσταση taskTracker.
Χρήση: hadoop tasktracker

ΠΑΡΑΡΤΗΜΑ Γ – HADOOP STREAMING

Το Hadoop Streaming είναι ένα εργαλείο που επιτρέπει την χρήση οποιουδήποτε εκτελέσιμου αρχείου σαν mapper ή/και reducer.

```
Χρήση: $HADOOP_HOME/bin/hadoop jar build/hadoop-streaming.jar [options]

Options:
```

-input <path>	DFS input file(s) for the Map step. Globing on <path> is supported and can have multiple -input
-output <path>	DFS output directory for the Reduce step
-mapper <cmd|JavaClassName>	The streaming command to run
-combiner <JavaClassName>	Combiner has to be a Java class
-reducer <cmd|JavaClassName>	The streaming command to run
-file <file>	File/dir to be shipped in the Job jar file
-dfs <h:p>|local	Optional. Override DFS configuration
-jt <h:p>|local	Optional. Override JobTracker configuration
-additionalconfspec specfile	Optional.
-inputformat TextInputFormat(default)| SequenceFileAsTextInputFormat| JavaClassName	Optional. Default Map input format: a line is a record in UTF-8. Every line must end with an 'end of line' delimiter. The key part ends at first TAB, the rest of the line is the value.
-outputformat TextOutputFormat(default)| JavaClassName	Optional.
-partitioner JavaClassName	Optional.
-numReduceTasks <num>	Optional.
-inputreader <spec>	Optional. Custom Map input format: -inputreader package.MyRecordReader,n=v,n=v comma-separated name-values can be specified to configure the InputFormat Example: -inputreader 'StreamXmlRecordReader,begin=<doc>,end=</doc>'
-jobconf <n>=<v>	Optional. Add or override a JobConf property. To set the number of reduce tasks (num. of output files): -jobconf mapred.reduce.tasks=10 To change the local temp directory: -jobconf dfs.data.dir=/tmp Additional local temp directories with -cluster local: -jobconf mapred.local.dir=/tmp/local -jobconf mapred.system.dir=/tmp/system -jobconf mapred.temp.dir=/tmp/temp
-cmdenv <n>=<v>	Optional. Pass env.var to streaming commands To set an environment variable in a streaming command: -cmdenv EXAMPLE_DIR=/home/example_dir/
-cacheFile fileNameURI	
-cacheArchive fileNameURI	
-verbose	

PRACTICAL HELP

Using the streaming system you can develop working hadoop jobs with EXTREMELY limited knowledge of Java. At it's simplest your development task is to write two shell scripts that work well together, let's call them **shellMapper.sh** and **shellReducer.sh**. On a machine that doesn't even have hadoop installed you can get first drafts of these working by writing them to work in this way:

```
cat someInputFile | shellMapper.sh | shellReducer.sh > someOutputFile
```

With streaming, Hadoop basically becomes a system for making pipes from shell-scripting work (with some fudging) on a cluster. There's a strong logical correspondence between the unix shell scripting environment and hadoop streaming jobs. The above example with Hadoop has somewhat less elegant syntax, but this is what it looks like:

```
stream -input /dfsInputDir/someInputData -file shellMapper.sh -mapper
"shellMapper.sh" -file shellReducer.sh -reducer "shellReducer.sh" -output
/dfsOutputDir/myResults
```

The real place the logical correspondence breaks down is that in a one machine scripting environment shellMapper.sh and shellReducer.sh will each run as a single process and data will flow directly from one process to the other. With Hadoop the shellMapper.sh file will be sent to every machine on the cluster that has data chunks and each such machine will run it's own chunk through the shellMapper.sh process on each machine. The output from those scripts DOESN'T run a reduce on each of those machines. Instead the output is sorted so that different lines from various mapping jobs are streamed across the network to different machines (Hadoop defaults to four machines) where the reduce(s) can be performed.

Here are practical tips for getting things working well:

* **Use shell scripts rather than commands** - The "-file shellMapper.sh" part isn't entirely necessary. You can simply use a clause like "-mapper 'sed | grep | awk'" or some such but complicated quoting is can introduce bugs. Wrapping the job in a shell script eliminates some of these issues.

* **Don't expect shebangs to work** - If you're going to run other scripts from inside your shell script, don't expect a line like #!/bin/python to work. To be certain that things will work, run the script directly like "grep somethingInteresting | ***perl*** PERLSCRIPT | sort | uniq -c"