ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΠΑΡΑΛΛΗΛΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ CLUSTERS ΜΕ ΤΟ ΜΟΝΤΕΛΟ MAPREDUCE ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ ΤΟ APACHE HADOOP

Του φοιτητή

Καρανίκου Νικόλαου

Αρ. Μητρώου: 05/2899

Επιβλέπων καθηγητής

Κ. Διαμαντάρας

Θεσσαλονίκη 2013

ΠΡΟΛΟΓΟΣ

Η επιστήμη των υπολογιστών μετά από μια πλήρη ενασχόληση με τον σειριακό προγραμματισμό, άρχισε τα τελευταία χρόνια να στρέφεται ολοένα και περισσότερο στον παράλληλο προγραμματισμό.

Οι σημερινοί υπολογιστές είναι περίπου 100 φορές πιο γρήγοροι από αυτούς που κατασκευάζονταν πριν από 10 χρόνια, παρ ’όλα αυτά όμως, υπάρχουν πολλοί μηχανικοί και προγράμματα σήμερα που χρειάζονται ακόμη μεγαλύτερες ταχύτητες. Επομένως η επιστήμη των υπολογιστών στην αναζήτηση καινοτόμων τεχνολογιών για την επίλυση σημαντικών προβλημάτων με μεγαλύτερες απαιτήσεις σε υπολογιστική ισχύ στρέφεται ολοένα και περισσότερο στον παράλληλο προγραμματισμό. Στην επιστήμη των υπολογιστών παράλληλος προγραμματισμός ονομάζεται η ανάπτυξη εφαρμογών οι οποίες εκμεταλλεύονται την ύπαρξη πολλαπλών επεξεργαστικών μονάδων σε έναν πολυεπεξεργαστή ή μία συστάδα υπολογιστών για να επιτύχουν αύξηση των υπολογιστικών επιδόσεων και μείωση του απαιτούμενου χρόνου εκτέλεσης μίας εφαρμογής που ουσιαστικά είναι και το ζητούμενο.

Βεβαίως σημαντικό παράγοντα στην ανάπτυξη του παράλληλου προγραμματισμού αποτέλεσε η εξέλιξη των παράλληλων, κατανεμημένων συστημάτων, που είναι απαραίτητα για την ορθή και όσο το δυνατόν γρηγορότερη εκτέλεση του προγράμματος. Ένα παράλληλο σύστημα είναι ένα ειδικά σχεδιασμένο σύστημα που περιέχει πολλούς επεξεργαστές ή πολλούς υπολογιστές που είναι στενά συνδεδεμένοι μεταξύ τους μέσω δικτύου, οι οποίοι συνεργάζονται για την λύση ενός προβλήματος.

Έχουν αναπτυχθεί πολλά τέτοια παράλληλα συστήματα με διάφορες αρχιτεκτονικές. Εμείς θα εξετάσουμε την εγκατάσταση και υλοποίηση προγραμμάτων σε συστάδες υπολογιστών (Cluster) με τη χρήση του Apache Hadoop, μια υλοποίηση του μοντέλου MapReduce η οποία είναι γραμμένη σε Java και είναι ανοιχτού κώδικα.

ΠΕΡΙΛΗΨΗ

Κατά την ανάπτυξη και ολοκλήρωση της πτυχιακής μου εργασίας ερευνήθηκαν διάφορα θέματα που σχετίζονται με τον παράλληλο προγραμματισμό και πιο συγκεκριμένα με το Apache Hadoop και το MapReduce.

Στα πλαίσια αυτής της εργασίας μελετήθηκαν, το μοντέλο MapReduce θεωρητικά, οι βασικές συναρτήσεις του που πρέπει να υλοποιηθούν από τον προγραμματιστή και ο τρόπος διαχωρισμού των δεδομένων και εξαγωγής των αποτελεσμάτων. Ειδικότερα μελετήθηκε το Hadoop, μία συγκεκριμένη, ανοιχτού κώδικα υλοποίηση του μοντέλου MapReduce η οποία είναι γραμμένη σε Java.

Η εργασία αυτή επίσης παρέχει έναν πλήρη και καθόλα σαφή οδηγό για το πώς γίνεται η εγκατάσταση του Hadoop σε έναν υπολογιστή : (*pseudo- distributed,single- node cluster*), αλλά και τις απαραίτητες ρυθμίσεις που πρέπει να γίνουν σε αυτό ώστε να έχουμε ένα δίκτυο υπολογιστών που να λειτουργούν παράλληλα σαν μία συστάδα : (*fully- distributed, multi-node cluster*).

Επίσης δίνονται οδηγίες για την παραμετροποίηση του Eclipse ώστε να μπορούμε να γράψουμε κώδικα σε Java, σύμφωνα με τους κανόνες του MapReduce, έχοντας παράλληλα και τη βοήθεια του compiler.

Τέλος έγινε υλοποίηση ενός παράλληλου προγράμματος σε Java, για λόγους επίδειξης (demo), το οποίο επεξεργάζεται μεγάλο όγκο δεδομένων προερχόμενων από τον server του Α.Τ.Ε.Ι.Θ. και παράγει τα επιθυμητά αποτελέσματα.

ABSTRACT

During the development and completion of my Final Thesis, several issues related to parallel programming have been investigated and more specifically the Apache Hadoop framework and the MapReduce system.

Within this paper the MapReduce programming model is theoretically studied and in particular the basic functions that the programmer has to implement and how the model divides the data across its nodes and delivers the results. Hadoop, which is a specific open source java code implementation of MapReduce is studied and explained as well.

This paper also describes a full and clear tutorial on how to set-up a : *pseudo- distributed, single- node* Hadoop cluster and the necessary configuration that has to be done so that we have a fully functional cluster : *fully- distributed, multi-node* Hadoop cluster Furthermore instructions are given on how to set up Eclipse so we can write java code according to the MapReduce rules and also have the compiler's assistance.

Finally a demo Java code is written which deals with big Data-sets from the A.T.E.I.Θ server and produces the desired results.

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα πτυχιακή εργασία δεν θα μπορούσε να ολοκληρωθεί και να πάρει την τελική της μορφή χωρίς τη βοήθεια και συμπαράσταση όλων των ανθρώπων του προσωπικού και ακαδημαϊκού περιβάλλοντος στο οποίο ζω.

Καταρχήν, θα ήθελα να ευχαριστήσω τον επιβλέποντα της παρούσας εργασίας κ. Διαμαντάρα Κωνσταντίνο για την ευκαιρία που μου έδωσε να συνεργαστώ μαζί του. Τον ευχαριστώ για την καθοδήγηση, τη συμπαράσταση και την εμπιστοσύνη που μου έδειξε καθόλη τη διάρκεια της προσπάθειάς μου αυτής. Πάνω από όλα όμως θα ήθελα να τον ευχαριστήσω γιατί η αξιοπρέπεια και η ηθική του αποτελούν για εμένα το πιο σημαντικό παράδειγμα.

Ευχαριστώ θερμά τον κ. Φιλέλη, του Κέντρου Διαχείρισης Δικτύου του Α.Τ.Ε.Ι.Θ για τη πολύτιμη βοήθεια του στην παροχή ενός αρχείου μεγάλου όγκου δεδομένων.

Τέλος το πιο σημαντικό ευχαριστώ στους γονείς μου Θεόδωρο και Βασιλική, καθώς και στις αδερφές μου Φυλλονίκη και Μαρία, για τη στήριξη που μου έδιναν καθ’ όλη τη διάρκεια αυτής της χρονιάς και για όλα αυτά που έχουν προσφέρει στη ζωή μου. Αποτελούν για μένα την αφετηρία και τη βάση όσων κάνω.

ΠΕΡΙΕΧΟΜΕΝΑ

Ευρετήριο σχημάτων

Ευρετήριο Εικόνων

ΚΕΦΑΛΑΙΟ 1
1. Θεωρητικά για το MapReduce.

1.1 Το Προγραμματιστικό μοντέλο MapReduce

Το MapReduce είναι ένα μοντέλο υπολογισμού για επεξεργασία και παραγωγή τεραστίων συνόλων δεδομένων που προτάθηκε για πρώτη φορά από το Google το 2004 στο άρθρο των JeffreyDean και SanjayGhemawat[4]. Αρχικός σκοπός του μοντέλου MapReduce ήταν η επεξεργασία τεραστίων συνόλων δεδομένων πάνω σε ένα σύνολο από δικτυωμένους οικιακούς υπολογιστές (clusterofcommoditymachines). Για την επίλυση κάποιου προβλήματος με το MapReduce, ο προγραμματιστής πρέπει να υλοποιήσει τουλάχιστο την συνάρτηση map, η οποία επεξεργάζεται ένα ζεύγος κλειδιού/τιμής για να παράγει ένα ενδιάμεσο ζεύγος κλειδιού/τιμής, και την συνάρτηση reduce, η οποία ενώνει όλες τις ενδιάμεσες τιμές που σχετίζονται με το ίδιο ενδιάμεσο κλειδί.

Το προγραμματιστικό μοντέλο, που χρησιμοποιείται για την υλοποίηση του MapReduce, είναι αρκετά απλό και γενικό. Δηλαδή για οποιοδήποτε πρόβλημα θα λυθεί με το μοντέλο η ιδέα της υλοποίησης και ο αλγόριθμος που ακολουθείται είναι κοινός. Η ιδέα της λειτουργίας του MapReduce είναι να παίρνει σαν είσοδο ένα σύνολο από ζευγάρια κλειδιών εισόδου και στην έξοδο να παράγει ένα σύνολο από ζευγάρια κλειδιών εξόδου. Ο χρήστης του MapReduce εκφράζει την πράξη αυτή σαν δύο συναρτήσεις/μεθόδους, την map και την reduce.
Ο προγραμματιστής το μόνο που έχει να κάνει είναι να υλοποιήσει τις δύο αυτές συναρτήσεις.

Η **map** συνάρτηση παίρνει σαν είσοδο ένα ζεύγος κλειδιού και, αφού εφαρμόσει την συνάρτηση του χρήστη πάνω τους, παράγει ένα ενδιάμεσο ζεύγος κλειδιού, το οποίο στην συνέχεια θα δώσει στην συνάρτηση reduce, η οποία με τη σειρά της ενώνει όλες αυτές τις ενδιάμεσες τιμές που σχετίζονται με το ίδιο ενδιάμεσο κλειδί.

Η συνάρτηση **reduce** προσπαθεί να συγχωνέψει όλες αυτές τις τιμές εφαρμόζοντας τον κώδικα του χρήστη, για να μπορέσει πιθανώς να δημιουργήσει ένα μικρότερο σύνολο τιμών. Συνήθως δημιουργείται μια τιμή εξόδου για κάθε εκτέλεση της reduce.

Αυτό που έχει να κερδίσει ένας προγραμματιστής μέσα από την χρήση του μοντέλου αυτού είναι η απλότητα του κώδικα και του αλγορίθμου. Ο ίδιος δε χρειάζεται να ασχοληθεί καθόλου με τον παραλληλισμό της εκτέλεσης του αλγόριθμου αυτού. Ότι αφορά την παράλληλη εκτέλεση, τη συνοχή των δεδομένων και αποτελεσμάτων, αλλά και την επικοινωνία επαφίενται στο runtime system. Συνεπώς, ο κώδικας γίνεται εύκολα φορητός σε διαφορετικά συστήματα.

Οι κώδικες του MapReduce εύκολα μπορούν να παραλληλιστούν, αφού η map μπορεί να εκτελεστεί με τα ίδια δεδομένα σε πολλές μηχανές. Η reduce, αφού έχει πάρει τα αποτελέσματα της map, μπορεί να δημιουργηθεί σε πολλά αντίγραφα με τα ίδια δεδομένα ή ακόμα να μοιραστούν τα δεδομένα στα διάφορα αντίγραφα και να εκτελεστούν παράλληλα.

1.2 Τρόπος λειτουργίας του μοντέλου

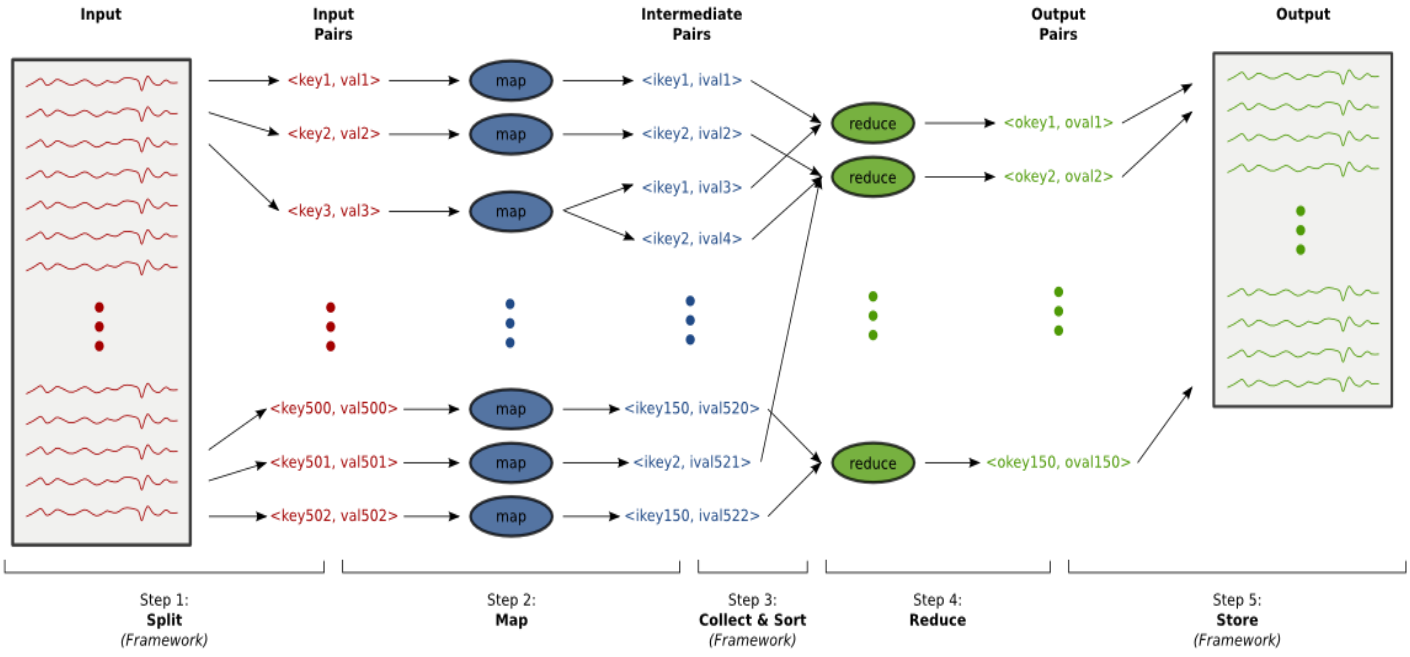
Στην πραγματικότητα το MapReduce είναι ένα μοντέλο ή μια στρατηγική για τη συγγραφή προγραμμάτων που να μπορούν εύκολα να επεξεργάζονται μεγάλο όγκο δεδομένων παράλληλα. Αυτό αυτομάτως συνεπάγεται την ύπαρξη μίας πλατφόρμας (Framework) πάνω στην οποία στηρίζεται το μοντέλο και η οποία είναι υπεύθυνη να διαχειρίζεται αυτόματα τις λεπτομέρειες του διαχωρισμού και του διαμερισμού των εργασιών και την ανοχή σφαλμάτων.

Οι βασικές εργασίες που υλοποιεί το μοντέλο όπως αναφέρονται παραπάνω είναι δύο, η φάση του mappingκαι η φάση του reducing. Και οι 2 αυτές μέθοδοι δουλεύουν πάνω σε ζευγάρια κλειδιών – τιμών, *key-value pairs.* Το τί ακριβώς τιμές θα περιέχουν αυτά τα κλειδιά εξαρτάται απόλυτα από τον προγραμματιστή και τον τύπο της μεταβλητής που θέλει να χρησιμοποιήσει. Τα ζεύγη κλειδιών – τιμών δεν χρειάζεται να είναι του ίδιου τύπου και οι δύο μέθοδοι mapκαι reduceμπορούν να εξάγουν διαφορετικού τύπου ζεύγη από αυτά που θα δεχθούν. Είναι όμως απαραίτητη προϋπόθεση η μέθοδος reduceνα δέχεται ζεύγη του ίδιου τύπου με αυτά που θα εξάγει η map.

Θεωρητικά, και όπως φαίνεται στο σχήμα 1.1 ο τρόπος με το οποίο λειτουργεί το μοντέλο περιγράφεται από τα παρακάτω πέντε βήματα :

1. Τα δεδομένα εισόδου χωρίζονται σε ζεύγη κλειδιών-τιμών και δίνονται σαν είσοδος στον mapper (*Splitting*). Υπεύθυνη για την διαδικασία αυτή είναι η πλατφόρμα.
2. Ο mapperεπεξεργάζεται κάθε ζεύγος ξεχωριστά και εξάγει ένα ενδιάμεσο ζεύγος κλειδιού-τιμής (*intermediate key-value pairs*).

3. Όλα τα ενδιάμεσα ζεύγη συγκεντρώνονται, ταξινομούνται και ομαδοποιούνται με βάση το κλειδί, μία διαδικασία για την οποία υπεύθυνη είναι η πλατφόρμα (*Shuffling*).
4. Για κάθε μοναδικό κλειδί, ο reducer λαμβάνει το κλειδί αυτό συνοδευόμενο από μία λίστα με όλες τις τιμές που σχετίζονται με αυτό. Ο reducer τώρα τις επεξεργάζεται με οποιοδήποτε τρόπο (πρόσθεση, αφαίρεση, εύρεση μεγίστου κτλ.) και εξάγει τα αποτελέσματα σε ένα ή περισσότερα ζεύγη κλειδιών-τιμών.
5. Τελικά η πλατφόρμα συλλέγει τα ζεύγη και τα αποθηκεύει στο αρχείο εξόδου.



Σχήμα 1.1 Λειτουργία του MapReduce

1.3 Πλεονεκτήματα και μειονεκτήματα

Το μοντέλο MapReduce ουσιαστικά είναι ένας απλός και αποτελεσματικός τρόπος υπολογισμού αθροισμάτων και γενικά επεξεργασίας πολύ μεγάλου όγκου δεδομένων. Τα κύρια χαρακτηριστικά του, που συμβάλουν στην αποτελεσματικότητα και την ευρεία χρήση του, είναι αυτά που το καθιστούν ως ένα από τα πλέον δημοφιλέστερα μοντέλα στον παράλληλο προγραμματισμό και είναι τα εξής :

1. **Απλό και εύκολο στη χρήση** . Το μοντέλο MapReduce δεν είναι μονάχα απλό αλλά και εύκολα κατανοητό. Με το μοντέλο αυτό ο προγραμματιστής καθορίζει τον κώδικα του έχοντας να υλοποιήσει μόνο δυο μεθόδους, τις Map και Reduce, και δεν χρειάζεται να τον απασχολεί το πως θα γίνει η διανομή και ο διαμερισμός των εργασιών ή των δεδομένων ανάμεσα στους κόμβους.
2. **Ευελιξία στο File System**. Το MapReduce δεν εξαρτάται από κάποιο άλλο μοντέλο ή σχήμα. Μπορεί να δεχθεί, να κατανοήσει, και να επεξεργαστεί δεδομένα τα οποία δεν έχουν κάποια συγκεκριμένη ή γνωστή δομή.
3. **Ανεξαρτησία**. Ένα μεγάλο πλεονέκτημα του MapReduce είναι το γεγονός ότι δουλεύει ανεξάρτητα από το σύστημα αρχείων που χρησιμοποιεί η πλατφόρμα πάνω στην οποία γίνεται η υλοποίηση του μοντέλου. Συνεπώς μπορεί να υλοποιηθεί και να λειτουργήσει σε όλα σχεδόν τα File Systems.
4. **Ανοχή στα λάθη**. Ένα ακόμα σπουδαίο πλεονέκτημα του μοντέλου είναι το γεγονός ότι είναι πολύ ανθεκτικό και διαθέτει έναν εξαιρετικό 'built-in' μηχανισμό αντιμετώπισης σφαλμάτων.
5. **Επεκτασιμότητα**. Ίσως το μεγαλύτερο από τα πλεονεκτήματα του μοντέλου είναι το ότι μπορεί εύκολα και γρήγορα να προσαρμόζεται στις αλλαγές που γίνονται όσον αφορά τους κόμβους που είναι συνδεδεμένοι στο σύστημα, είτε αυτοί μειώνονται είτε αυτοί αυξάνονται. Το γεγονός ότι είναι σχεδιασμένο να λειτουργεί ακόμα και με φθηνούς υπολογιστές του εμπορίου το κάνει ακόμη πιο ελκυστικό.
6. **Καλό Load-Balancing**. Πολύ σημαντικό στον παράλληλο προγραμματισμό είναι να υπάρχει καλό Load-Balancing. Να μοιράζονται δηλαδή οι εργασίες σε όλους τους κόμβους έτσι ώστε ανά πάσα στιγμή ο φόρτος εργασίας του καθενός να βρίσκεται στα ίδια επίπεδα με των υπολοίπων. Χρησιμοποιώντας το δυναμικά, κατά τη διάρκεια της εκτέλεσης(on-runtime), το μοντέλο πετυχαίνει αρκετά ικανοποιητικό Load-Balancing.

Παρά τα πολλά του πλεονεκτήματα το MapReduce δεν διαθέτει μερικά χαρακτηριστικά που είναι αναγκαία για την εκτέλεση ορισμένων εργασιών και στα συγκεκριμένα σημεία αυτά ελλείπει. Γεγονός είναι βέβαια ότι παρά τα μειονεκτήματα του είναι ακόμη αρκετά "νέο" σαν μοντέλο, καθώς εισήχθη από την Google μόλις το 2004, και αυτό δίνει στον κόσμο του προγραμματισμού αρκετές ελπίδες για βελτίωση. Αναλυτικά :

1. **Ακαμψία**. Το μοντέλο επεξεργασίας μοναδικής εισόδου δύο σταδίων (map, reduce), που υλοποιεί το MapReduce δεν είναι αρκετά ευέλικτο ώστε να μπορεί να προσαρμοστεί σε όλα τα επιθυμητά σενάρια. Αυτό αυτομάτως συνεπάγεται την ανικανότητα του μοντέλου να προσφέρει λύση σε όλα τα προβλήματα.
2. **Γλώσσα χαμηλού επιπέδου**. Το μοντέλο αυτό δεν υποστηρίζει γλώσσες υψηλού επιπέδου όπως για παράδειγμα η SQL στην DBMS ούτε και κάποια άλλη τεχνική βελτιστοποίησης των ερωτημάτων (Queries). Συνεπώς ο προγραμματιστής είναι αναγκασμένος να χρησιμοποιήσει περιττό κώδικα στις map και reduce μεθόδους για να φτάσει στο επιθυμητό αποτέλεσμα.
3. **Δυσκολία βελτιστοποίησης**. Το γεγονός ότι το μοντέλο μας "κρύβει" τον κώδικα που χρησιμοποιεί, για να υλοποιήσει τις map και reduce διεργασίες, προσδίδει έναν μεγάλο βαθμό δυσκολίας στην προσπάθεια του προγραμματιστή για βελτιστοποίηση του παραλληλισμού και κατά συνέπεια του προγράμματος του.
4. **Δυσκολίες στην επεκτασιμότητα**. Παρά το ότι η επεκτασιμότητα ανήκει στα πλεονεκτήματα του μοντέλου, το γεγονός ότι χρησιμοποιεί έναν μόνο κόμβο-υπολογιστή σαν master του συστήματος, θέλει αρκετή προσοχή κατά την επέκταση και μπορεί να την επηρεάσει αρνητικά.
5. **Έλεγχος του προγράμματος**. Όσον αφορά το πόσο καλά χειρίζεται την παραγωγή και τον έλεγχο του κώδικα, το MapReduce υστερεί σε αρκετά σημεία. Η παρακολούθηση του cluster , το πως αυτό χειρίζεται τον κώδικα, η συλλογή των αρχείων με τις πληροφορίες(Logs) και το debugging είναι αρκετά επίπονο έως και αδύνατο στο MapReduce.

ΕΠΙΛΟΓΟΣ

Σαν πρώτο κεφάλαιο αυτής της εργασίας έγινε μία γενική και θεωρητική αναφορά στο μοντέλο MapReduce και στην υλοποίηση του που θα χρησιμοποιηθεί, το Hadoop. Αναφέρεται πολύ γενικά η ιδέα του μοντέλου και το πως αυτό λειτουργεί και επεξεργάζεται τα δεδομένα. Εκτενής αναφορά γίνεται επίσης στα πλεονεκτήματα και μειονεκτήματα του μοντέλου.

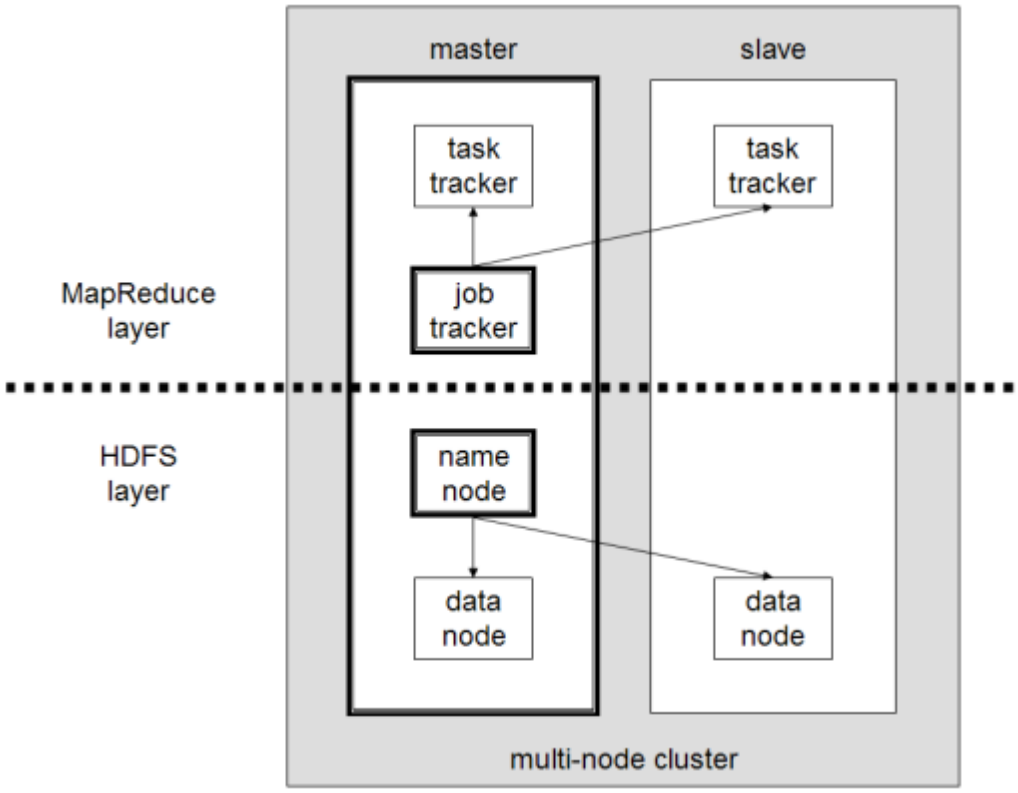
Στο επόμενο κεφάλαιο θα αναφερθούμε πολύ γενικά στην πλατφόρμα του Hadoop και θα παραθέσουμε τις πιο σημαντικές από τις εντολές χρήσης του, εντολές που πρέπει να γνωρίζει κάποιος ώστε να μπορεί να το χρησιμοποιήσει αποδοτικά.

ΚΕΦΑΛΑΙΟ 2
2. Αρχιτεκτονικές και πλατφόρμες - Hadoop.

2.1 Επισκόπηση Hadoop

Το Hadoop, είναι μια πολύ σημαντική υλοποίηση του μοντέλου MapReduce ανοιχτού κώδικα (open source) από την Yahoo! γραμμένη σε Java, η οποία έχει ως κύριο σκοπό την επίλυση προβλημάτων με μεγάλα δεδομένα εισόδου πάνω σε ένα δίκτυο από διασυνδεδεμένους οικιακούς υπολογιστές (cluster). Για την λειτουργία του Hadoop είναι απαραίτητη η αποθήκευση των δεδομένων εισόδου, αλλά και των αποτελεσμάτων, στο HDFS (Hadoop'sDistributedFileSystem), το οποίο είναι ένα κατανεμημένο σύστημα αποθήκευσης εξαιρετικά μεγάλων δεδομένων σε πολλαπλές μηχανές, που ανήκουν σε ένα δίκτυο (cluster) παρόμοιο με το GFS (GoogleFileSystem).

Η υλοποίηση αυτή ακολουθεί το μοντέλο αρχιτεκτονικής master/slave με ένα κεντρικόmaster server «jobtracker» και πολλούς slave servers «tasktrackers», ένα σε κάθε κόμβο του δικτύου μας. Ο χρήστης στέλνει τις map και reduce εργασίες του στονjobtracker, ο οποίος τις τοποθετεί σε μια σειρά από έτοιμες να εκτελεστούν εργασίες και τις εξυπηρετεί με την πολιτική FIFO (FirstInFirstOut) στέλλοντας τις στους tasktrackers για εκτέλεση. Ο κάθε tasktracker εκτελεί απλώς τις εργασίες που του αναθέτει ο jobtracker. Ακόμα υπάρχει ένας κεντρικός «namenode», ο οποίος κρατάει την θέση του κάθε δεδομένου μέσα στο HDFS και πολλοί «datanodes», πάλι ένας σε κάθε κόμβο του δικτύου μας, οι οποίοι αποθηκεύουν τα δεδομένα.



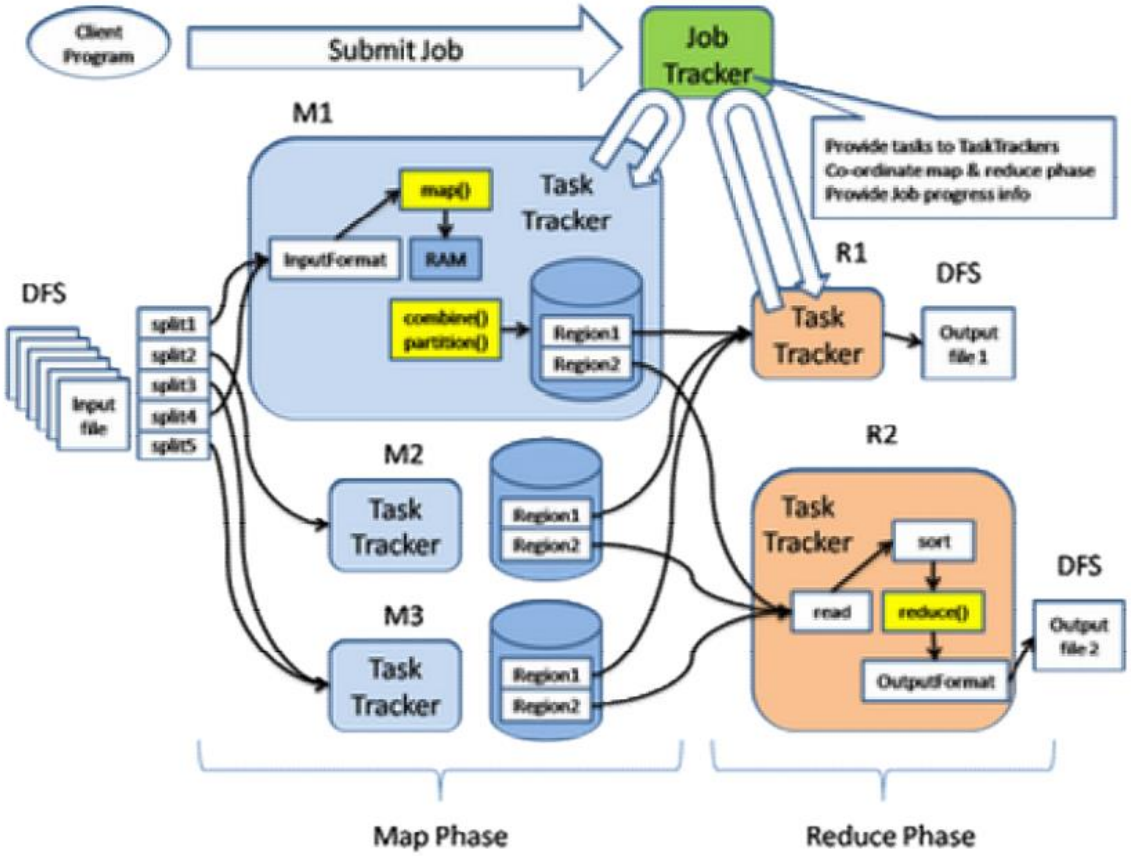
Σχήμα 2.1 Αρχιτεκτονική Hadoop cluster

Στη φάση map της εκτέλεσης το Hadoop «σπάζει» τα δεδομένα εισόδου σε παρά πολλά πιο μικρά κομμάτια, ανάλογα με το αρχικό μέγεθος τους αλλά και το μέγεθος του δικτύου μας, και αναθέτει το κάθε κομμάτι σε μια map εργασία. Στην συνέχεια διαμοιράζει τις map εργασίες μέσω του jobtracker στους υπολογιστές, που βρίσκονται μέσα στο δίκτυο, για εκτέλεση. Μετά την εκτέλεση της, η κάθε map εργασία γράφει το ενδιάμεσο αποτέλεσμα της στο HDFS στο τοπικό datanode του υπολογιστή, που έγινε η εκτέλεση.

Μετά την εκτέλεση των map εργασιών το Hadoop ταξινομεί τα ενδιάμεσα αποτελέσματα, με βάση το κλειδί τους, έτσι όλες οι τιμές, πού αναφέρονται στο ίδιο κλειδί, να εμφανίζονται μαζί διευκολύνοντας την φάση του reduce, που θα ακολουθήσει. Στην συνέχεια «σπάζει» τα ενδιάμεσα δεδομένα σε τόσα κομμάτια όσα και οι reduce εργασίες, που θα ακολουθήσουν.

Στη φάση reduce, που ακολουθεί, ο jobtracker στέλνει τις reduce εργασίες όσο πιο κοντά στα δεδομένα. Αν είναι δυνατό αναθέτει την εκτέλεση στον κόμβο, που έχει αποθηκευμένα τα ενδιάμεσα δεδομένα. Κάθε reduce εργασία εφαρμόζει την συνάρτηση reduce του χρήστη πάνω στο κομμάτι των ενδιάμεσων δεδομένων, που της έχουν ανατεθεί, και παράγει το τελικό αποτέλεσμα.

Η κάθε φάση εκτελείται με διαχείριση των σφαλμάτων. Αν ένας κόμβος αποτύχει την ώρα πού εκτελεί μια εργασία, τότε το Hadoop αναθέτει την ίδια εργασία σε ένα άλλο κόμβο του δικτύου για επανεκτέλεση.



Σχήμα 2.2 Οι φάσεις εκτέλεσης του Hadoop

Διατηρώντας μεγάλο αριθμό από map και reduce εργασίες το Hadoop επιτυγχάνει καλό load balancing και σε περιπτώσεις οπού υπάρχει σφάλμα η επανεκτέλεση της συγκεκριμένης εργασίας έχει μικρό overhead.

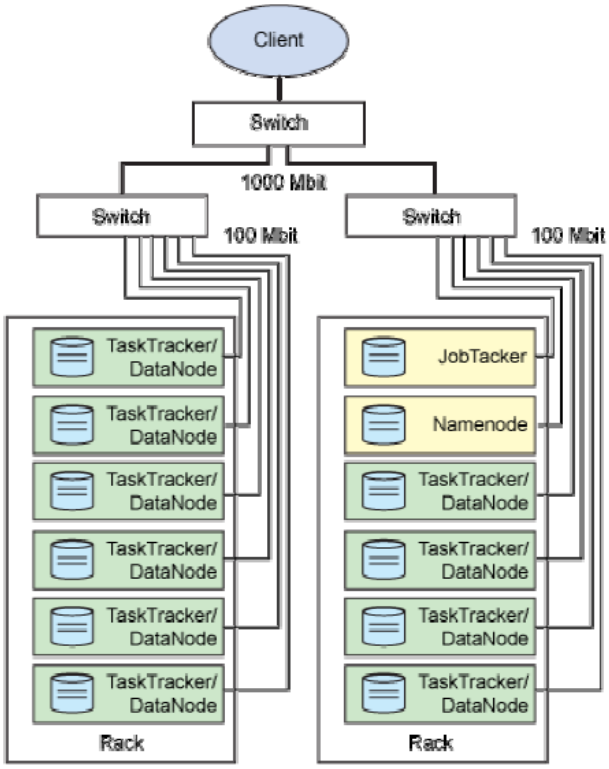
Το Hadoop σήμερα είναι η πιο διαδεδομένη υλοποίηση του MapReduce και χρησιμοποιείται για διδακτικούς σκοπούς σε αρκετά πανεπιστήμια του κόσμου, αλλά και σε μεγάλους οργανισμούς ανά το παγκόσμιο για την επεξεργασία μεγάλων δεδομένων εισόδου. Κάποιοι από τους οργανισμούς, που διατηρούν clusters για εκτέλεση Hadoop εργασιών, είναι: Yahoo!, Amazon, AOL, Alibaba, CornellUniversityWebLab, ETHZurichSystemsGroup, Facebook, Google, IBM, NewYorkTimes κ.α.

2.2 Αρχιτεκτονική Hadoop cluster

Ένα δίκτυο εκτέλεσης Hadoop μπορεί να αποτελείται από μια μέχρι και αρκετές δεκάδες χιλιάδες δικτυωμένες μηχανές. Δεν απαιτείται να εγκατασταθεί σε μηχανές συγκεκριμένης αρχιτεκτονικής, λειτουργικού συστήματος ή προδιαγραφών, αφού μπορεί να τρέξει ακόμα και σε οικιακούς υπολογιστές φτάνει να μπορούν να υποστηρίξουν την πλατφόρμα Java.

Μια τυπική μεγάλης κλίμακας εγκατάσταση του Hadoop αποτελείται από τουλάχιστο τις εξής μηχανές κόμβους (nodes): Μια μηχανή που τρέχει τον namenode, μια μηχανή που τρέχει τον jobtracker και αρκετές μηχανές που τρέχουν ένα datanode και ένα tasktracker μαζί (βλ. σχ. 1.2).

Μπορούμε να επιλέξουμε να εγκαταστήσουμε τα datanodes και τα tasktrackers σε διαφορετικές μηχανές αλλά πρέπει να γίνει με τέτοιο τρόπο, ώστε να μην χάσουμε την βελτιστοποίηση rack awareness, που μας παρέχει τοHadoop, όπου ο jobtracker ξέρει ποιος κόμβος κατέχει τα δεδομένα προς επεξεργασία και ποιοι άλλοι κόμβοι είναι κοντά του και έτσι στέλλει την map ή reduce εργασία σε αυτούς. Αυτή η βελτιστοποίηση βασίζεται στην αρχή του ότι είναι πιο συμφέρον από άποψη χρόνου εκτέλεσης η μετακίνηση της επεξεργασίας παρά η μετακίνηση των δεδομένων.



Σχήμα 2.3 Τυπική Αρχιτεκτονική Hadoop

Namenode: αποτελεί τον master κόμβο του HDFS και είναι υπεύθυνος για την διαχείριση του namespace του file system, δηλαδή αναλαμβάνει να κατανέμει τα blocks των δεδομένων, που αποθηκεύονται στο HDFS, σε όλους τους κόμβους του δικτύου και ρυθμίζει την πρόσβαση των client κόμβων στα αρχεία αυτά εκτελώντας εντολές ανοίγματος και κλεισίματος αρχείων.

Τέλος, ο namenode είναι υπεύθυνος για την αντιγραφή των δεδομένων και τη διατήρηση αντιγράφων τους σε διάφορους κόμβους του δικτύου για σκοπούς απόδοσης, αλλά και για προστασία από την απώλεια δεδομένων λόγω αστοχίας του υλικού.

Datanodes: αποτελούν τους client κόμβους του HDFS και είναι υπεύθυνοι για την αποθήκευση των block των δεδομένων πάνω στους σκληρούς δίσκους ή σε αλλά αποθηκευτικά μέσα πιθανόν να διαθέτουν οι μηχανές στις οποίες τρέχουν. Οι datanodes εξυπηρετούν αιτήσεις διαβάσματος και γραψίματος από τους jobtracker κόμβους του δικτύου και μετά από εντολή του namenode δημιουργούν ή διαγράφουν blocks δεδομένων.

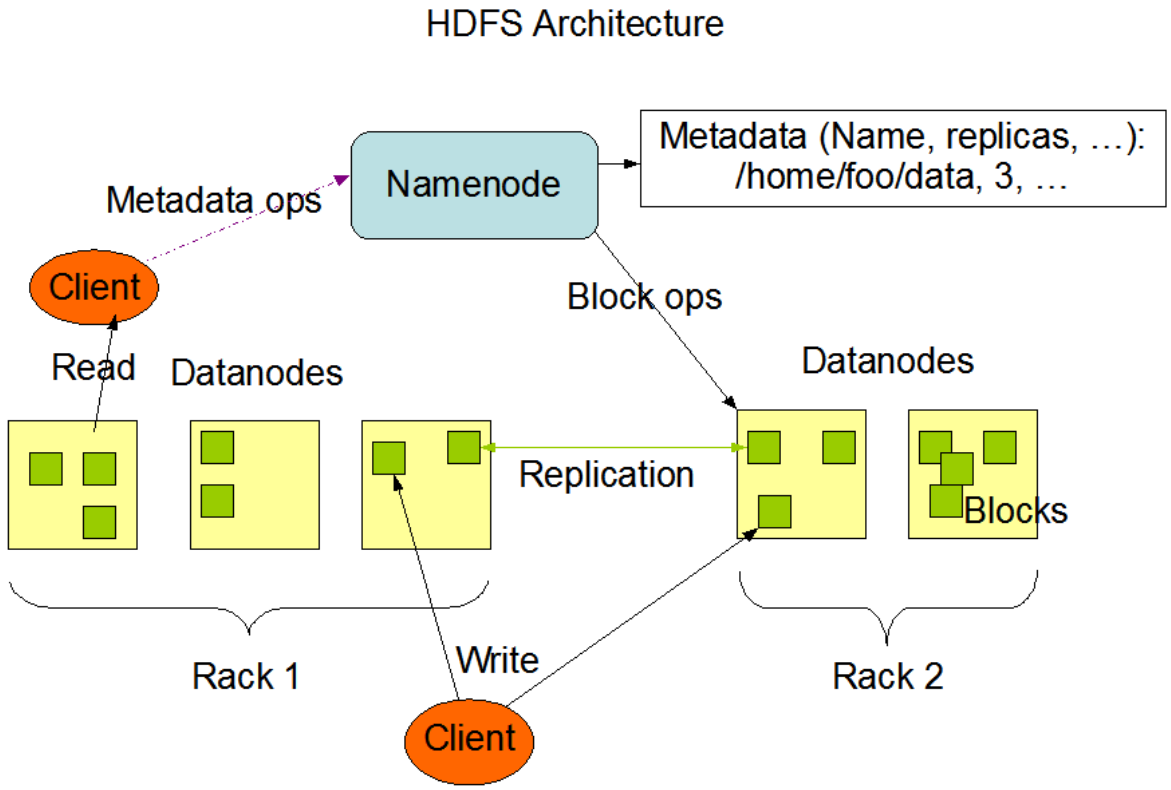
Jobtracker: αποτελεί τον scheduler, που ρυθμίζει την παράλληλη εκτέλεση των εργασιών map και reduce πάνω στο δίκτυο. Ο jobtracker διαθέτει μια στοίβα (stack) στην οποία τοποθετούνται, σε κατάσταση αναμονής, από το Hadoop όλες οι εργασίες που είναι έτοιμες για εκτέλεση. Εφαρμόζοντας την τεχνική FIFO ο jobtracker αναθέτει μια εργασία από την στοίβα του σε όποιο tasktracker είναι διαθέσιμος (idle) για εκτέλεση εφαρμόζοντας ,οπού είναι δυνατό, και την βελτιστοποίηση rack awareness. Για να αρχίσει να μοιράζει τις reduce εργασίες στους κόμβους προς εκτέλεση θα πρέπει πρώτα να τελειώσουν την εκτέλεση τους όλες οι map εργασίες.

Tasktrackers: αποτελούν ουσιαστικά το κομμάτι το οποίο εκτελεί τους υπολογισμούς και διεκπεραιώνει τις map και reduce εργασίες. Αρχίζουν να εκτελούν οποία εργασία τους στείλει ο jobtracker διαβάζοντας τα δεδομένα από το datanode και μετά την επεξεργασία γράφουν τα αποτελέσματα πίσω σε αυτόν. Σε κάθε μηχανή του δικτύου hadoop συνήθως τρέχει ένα μόνο tasktracker και επεξεργάζεται κυρίως τα δεδομένα, που υπάρχουν στο datanode, το οποίο τρέχει στην ίδια με αυτό μηχανή.

2.3 Αρχιτεκτονική HDFS

Το HDFS ακολουθεί το μοντέλο αρχιτεκτονικής client/server και αποτελείται από ένα και μοναδικό namenode, τον master δηλαδή του δικτύου. Ο namenode διαχειρίζεται το namespace του file system και ρυθμίζει την πρόσβαση στα αρχεία από τους clients κόμβους του δικτύου. Εκτός από τον κόμβο namenode υπάρχουν και αρκετοί, συνήθως όσοι και οι κόμβοι του δικτύου Hadoop, datanodes, οι οποίοι χειρίζονται τους σκληρούς δίσκους της μηχανής στην οποία τρέχουν. Αυτή η διαρρύθμιση δίνει στο χρήστη την εντύπωση ότι χρησιμοποίει ένα κοινό και ενιαίο namespace για όλους τους κόμβους του δικτύου πάνω στο οποίο τα δεδομένα μπορούν να αποθηκευτούν / ανακτηθούν από οποιοδήποτε κόμβο του δικτύου.

Εσωτερικά, μέσα στο HDFS, ένα αρχείο σπάζει σε ένα ή περισσότερα κομμάτια (blocks) και αυτά με την σειρά τους αποθηκεύονται σε ένα ή περισσότερα datanodes του δικτύου μας. Ο namenode αναλαμβάνει να καθορίζει την κατανομή (mapping) των blocks μέσα στα datanodes και εκτελεί τις λειτουργίες του namespace, όπως το άνοιγμα / κλείσιμο και η μετονομασία ή μεταφορά αρχείων. Οι datanodes αναλαμβάνουν να εξυπηρετούν τις αιτήσεις διαβάσματος και εγγραφής από τους tasktrackers των κόμβων του Hadoop και μετά από απαίτηση του namenode εκτελούν εντολές δημιουργίας, καταστροφής και αντιγραφής αρχείων.



Σχήμα 2.4 Αρχιτεκτονική HDFS

ΕΠΙΛΟΓΟΣ

Στο κεφάλαιο που προηγήθηκε αναφερθήκαμε στο θεωρητικό κομμάτι του Hadoop και εξηγηθήκαν αναλυτικά οι βασικές διεργασίες που χρησιμοποιεί για την υλοποίηση του μοντέλου MapReduce. Επίσης επεξηγήθηκε και το σύστημα αρχείων που χρησιμοποιεί η πλατφόρμα. Όσον αφορά το Hadoop δείξαμε ουσιαστικά πως υλοποιεί το μοντέλο MapReduce.

Έχοντας καλύψει το θεωρητικό κομμάτι της εργασίας στο επόμενο κεφάλαιο παρατίθεται ένας πλήρης και αναλυτικός οδηγός για την εγκατάσταση του Hadoop cluster. Μία διαδικασία που ακολουθήθηκε κατά την εγκατάσταση της πλατφόρμας δοκιμαστικά ώστε να είναι δυνατή η συγγραφή και η εκτέλεση παράλληλου κώδικα καθώς και η εξαγωγή αποτελεσμάτων.

ΚΕΦΑΛΑΙΟ 3
3. Βασικές εντολές Hadoop

3.1 Εντολές χρήσης Hadoop

Το Hadoop τυγχάνει χειρισμού από εντολές που εισάγονται από τον χρήστη στην γραμμή εντολών του κεντρικού master κόμβου. Σε αυτό το τμήμα παραθέτω τις εντολές που πιθανόν να χρειαστούν για τη λειτουργία του. Όλες οι εντολές που παρατίθενται εκτελούνται από τον κατάλογο εγκατάστασης του Hadoop.

Εντολή για να γίνει format το file system :

hduser@ubuntu:~$ bin/hadoop namenode -format

Η συγκεκριμένη εντολή 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.

Εντολή για να ξεκινήσει το cluster του Hadoop :

hduser@ubuntu:~$ bin/start-all.sh

Η συγκεκριμένη εντολή πρέπει να εκτελεστεί ούτως ώστε να αρχίσουν να τρέχουν όλοι οι συστατικοί κόμβοι του Hadoop (namenode, datanode, jobtracker και tasktracker).

Εντολή αντιγραφής δεδομένων από το τοπικό file system στο HDFS :

hduser@ubuntu:~$ bin/hadoop dfs -copyFromLocal [source dir] [destination dir]

Η εντολή αυτή αντιγράφει το [sourcedir] από το τοπικό filesystem της μηχανής μας στο filesystem του hadoop (HDFS) στον κατάλογο [destination dir].

Εντολή αντιγραφής δεδομένων από το HDFS στο τοπικό filesystem :

hduser@ubuntu:~$ bin/hadoop dfs -copyToLocal [source dir] [destination dir]

Η εντολή αυτή αντιγράφει το [sourcedir] από το HDFS στο τοπικό filesystem της μηχανής μας στον κατάλογο [destination dir].

Εντολή προβολής περιεχομένων HDFS :

hduser@ubuntu:~$ bin/hadoopdfs -ls

Η εντολή αυτή μπορεί να χρησιμοποιηθεί για να έχει κανείς πρόσβαση στο περιεχόμενο του HDFS ή αν χρησιμοποιηθεί σε συνδυασμό με το όνομα ενός φακέλου του file system, μπορεί να προβάλει το περιεχόμενο του φακέλου αυτού ακριβώς όπως η εντολή ls των Linux.

Εντολή εκτέλεσης προγράμματος Hadoop :

hduser@ubuntu:~$ bin/hadoopjar [hadoopprogram] [input] [output]

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.

Εντολή προβολής αρχείων :

hduser@ubuntu:~$ bin/hadoop dfs -cat [directory]/[file]

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 Linux.

Εντολή διαγραφής ένας κατάλογος από το HDFS :

hduser@ubuntu:~$ bin/hadoop dfs –rmr [directory]

Η συγκεκριμένη εντολή διαγράφει τον κατάλογο [directory] από το HDFS είτε είναι άδειος είτε όχι.

Εντολή για να σταματήσει το cluster του Hadoop :

hduser@ubuntu:~$ bin/stop-all.sh

Η συγκεκριμένη εντολή πρέπει να εκτελεστεί ώστε να τερματιστεί η λειτουργία των συστατικών κόμβων του Hadoop (namenode, datanode, jobtracker και tasktracker).

Πλήρης οδηγός εντολών παραθέτεται σαν παράρτημα στο Παράρτημα Α.

3.2 Πώς να τρέξετε ένα πρόγραμμα Hadoop

Για να τρέξετε ένα πρόγραμμα Hadoop πρέπει πρώτα να εγκαταστήσετε μια έκδοση του Hadoop σε ένα φάκελο της επιλογής σας. Αναλυτικότερες οδηγίες παρατίθενται παρακάτω στο κεφάλαιο 4. Επειδή όλα τα δεδομένα εισόδου πρέπει να είναι στο HDFS πρέπει να τα αντιγράψετε από το τοπικό file system του λειτουργικού σας. Πρώτα πρέπει να ξεκινήσουμε το Hadoop τρέχοντας, από τον φάκελο που το εγκαταστήσαμε, την εντολή:

hduser@ubuntu:~$ bin/start-all.sh

Αυτή η εντολή θα ξεκινήσει τα Namenode, Datanode, Jobtracker και Tasktracker πάνω στην μηχανή. Με την εντολή :

hduser@ubuntu:~$ bin/hadoop dfs -copyFromLocal [source dir] [destination dir]

Αντιγράφετε τα δεδομένα εισόδου στο HDFS. Για να εκτελέσετε το πρόγραμμα τρέξετε την εντολή:

hduser@ubuntu:~$ bin/hadoop jar [program] [input] [output]

Για να μεταφέρετε τα αποτελέσματα από το HDFS στο τοπικό file system του λειτουργικού σας για να μπορείτε να τα διαβάσετε εκτελείτε την εντολή:

hduser@ubuntu:~$ bin/hadoop dfs –copyToLocal [source dir] [destination dir]

ΕΠΙΛΟΓΟΣ

Στο κεφάλαιο που προηγήθηκε αναφερθήκαμε πολύ γενικά στην πλατφόρμα του Hadoop και παραθέσαμε τις σπουδαιότερες και πιο σημαντικές εντολές που πρέπει να γνωρίζει κάποιος ώστε να μπορεί να το χρησιμοποιήσει αποδοτικά.

Στο επόμενο κεφάλαιο θα γίνει μία αναφορά στο θεωρητικό κομμάτι της πλατφόρμας και θα επεξηγηθούν αναλυτικά οι σημαντικότερες από τις διεργασίες που αυτή χρησιμοποιεί, καθώς και το σύστημα αρχείων πάνω στο οποίο βασίζεται.

ΚΕΦΑΛΑΙΟ 4
4. Οδηγός εγκατάστασης του Hadoop (Single-Node).

4.1 Εισαγωγή

Στόχος αυτού του tutorial είναι η δημιουργία ενός multi-node Hadoop cluster. Παρακάτω δίνονται τα απαραίτητα βήματα, με όσο το δυνατόν περισσότερες λεπτομέρειες, ώστε να γίνει μια επιτυχημένη εγκατάσταση και λειτουργία ενός multi-node Hadoop cluster. Η εγκατάσταση θα γίνει σε Ubuntu-Linux 10.04 χρησιμοποιώντας το HadoopDistributedFileSystem (HDFS).

4.2 Εγκατάσταση λογισμικού

Οι GNU\Linux πλατφόρμες υποστηρίζονται από το Hadoop τόσο σαν πλατφόρμες ανάπτυξης όσο και παραγωγής. Το Hadoop έχει επίσης δοκιμαστεί επιτυχημένα σε GNU\Linux clusters με 2000 κόμβους. Η Win32 υποστηρίζεται μόνο σαν πλατφόρμα ανάπτυξης και δεν έχει δοκιμαστεί άρα δεν υποστηρίζεται σαν πλατφόρμα αναπαραγωγής.

Στους υπολογιστές του συγκεκριμένου cluster επιλέχθηκε η εγκατάσταση του Λειτουργικού Συστήματος Ubuntu Linux 10.04 LTS (ανήκει στην κατηγορία GNU\Linux πλατφορμών) για λόγους εξοικονόμησης πόρων του συστήματος (σε σχέση με νεότερες πιο ‘βαριές’ εκδόσεις) και ασυμβατότητας υλικού των Η\Υ που χρησιμοποιήθηκαν.

Σαν λογισμικό το Ubuntu είναι freeware. Από την ιστοσελίδα της Ubuntu[9] και με τη χρήση του UNetBootIn (ένα εργαλείο για τη δημιουργία bootableLiveUSBdrives[10]) περάστηκε σε USB flash drive και έγινε η εγκατάστασή του στους υπολογιστές.

4.3 Εγκατάσταση της Sun JAVA 6 JDK

Απαραίτητη είναι η ύπαρξη εγκατεστημένου Java JDK προκειμένου να λειτουργήσει το Hadoop. Σύμφωνα με το Hadoop προτιμώμενη έκδοση είναι η Java™ 1.6.x από την SUN.

Αυτή εγκαταστάθηκε με τον παρακάτω τρόπο :

Προσθήκη του FerramoscaRoberto repository στα δικά μου apt repositories

$ sudoadd-apt-repositoryppa:ferramroberto/java

Update της λίστας με τις πηγές

$ sudoapt-getupdate

Εγκατάσταση της Sun Java 6 JDK

$ sudo apt-get install sun-java6-jdk

Επιλογή της Sun Java ως προεπιλογή στον υπολογιστή

$ sudo update-java-alternatives -s java-6-sun

Η σωστή εγκατάσταση της Java ελέγχεται με την εντολή: java–version και το αποτέλεσμα πρέπει να είναι το παρακάτω.

user@ubuntu:~$ java –version

Java version "1.6.0_20"
Java(TM) SE Runtime Environment (build 1.6.0_20-b02)
Java HotSpot(TM) Client VM (build 16.3-b01, mixed mode, sharing)

4.4 Νέος Dedicated Hadoop User

Εφόσον προτείνεται χρησιμοποιήθηκε ένας νέος χρήστης για την λειτουργία του Hadoop, ώστε να ξεχωρίζει από άλλους χρήστες που χρησιμοποιούν διαφορετικά λογισμικά στον ίδιο υπολογιστή. Τα οφέλη αφορούν κυρίως σε θέματα ασφαλείας και δικαιωμάτων.
Οι εντολές είναι οι εξής :

```
$ sudo addgroup Hadoop
$ sudo adduser --ingroup hadoop hduser
```

Έτσι δημιουργείται ο χρήστης hduser που ανήκει στο group Hadoop

4.5 Εγκατάσταση του SSH(Secure Shell)

Όπως είναι γνωστό το SSH χρησιμοποιεί κρυπτογραφημένα κλειδιά για να αναγνωρίσει απομακρυσμένους υπολογιστές και να τους επιτρέψει την αλληλεπίδραση μεταξύ αυτών και του τρέχοντος υπολογιστή. Το Hadoop απαιτεί SSH πρόσβαση για να μπορεί να διαχειριστεί τους κόμβους του.

```
$sudo apt-get install ssh
```

4.6 Απενεργοποίηση της IPv6

Στα Ubuntu μηχανήματα, όταν σε διάφορες ρυθμίσεις του Hadoop χρησιμοποιούμε μία διεύθυνση IP της μορφής 0.0.0.0, τότε το Hadoop αυτόματα τις ερμηνεύει σαν IPv6 διευθύνσεις. Ένας τρόπος επίλυσης του προβλήματος αυτού θα ήταν να ρυθμίσουμε το Hadoop ώστε να αγνοεί τις IPv6 διευθύνσεις. Εφόσον το δίκτυο μας χρησιμοποιεί μόνο IPv4 αρκεί απλώς να απενεργοποιήσουμε την IPv6.

Για να το πετύχουμε αυτό πρέπει να προσθέσουμε τις παρακάτω γραμμές στο τέλος του αρχείου /etc/sysctl.conf

```
#disableipv6
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

4.7 Ρυθμίσεις SSH (Secure Shell)

Σαν πρώτο βήμα θα δώσουμε πρόσβαση στο localhost του hduser , εφόσον θέλουμε να τρέξουμε το Hadoop και τοπικά (στον master) και απομακρυσμένα (στους slaves).

Αρχικά πρέπει να δημιουργήσουμε ένα κλειδί για τον hduser χρήστη.

```
hduser@ubuntu:~$ ssh-keygen -t rsa -P ""
```

Το οποίο έχει ως αποτέλεσμα την δημιουργία ενός ζευγαριού κλειδιών χωρίς την ύπαρξη κωδικού όπως φαίνεται παρακάτω :

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
Created directory '/home/hduser/.ssh'.
Your identification has been saved in /home/hduser/.ssh/id_rsa.
Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.
The key fingerprint is:
9b:82:ea:58:b4:e0:35:d7:ff:19:66:a6:ef:ae:0e:d2hduser@ubuntu
.........
```

Στη συνέχεια πρέπει να επιτρέψουμε την πρόσβαση στον υπολογιστή, μέσω SSH, με το καινούργιο κλειδί που μόλις δημιουργήσαμε.

```
hduser@ubuntu:~$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

Τέλος θα δοκιμάσουμε την σύνδεση μέσω localhost. Επίσης θα προσθέσουμε το αποτύπωμα του κλειδιού του τοπικού μηχανήματος στο αρχείο του hduser που περιέχει τα γνωστά hosts.

```
hduser@ubuntu:~$ sshlocalhost
The authenticity of host 'localhost (::1)' can't be established.
RSA key fingerprint is d7:87:25:47:ae:02:00:eb:1d:75:4f:bb:44:f9:36:26.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (RSA) to the list of known hosts.
Linux ubuntu 2.6.32-22-generic #33-Ubuntu SMP Wed Apr 28 13:27:30 UTC 2010 i686
GNU/LinuxUbuntu 10.04 LTS
```

4.8 Εγκατάσταση του λογισμικού Hadoop 1.0.4

Πρώτο βήμα στην εγκατάσταση του Hadoop είναι να κατεβάσουμε μια οποιαδήποτε έκδοσή του από την ιστοσελίδα της Apache Hadoop[2] .
Στη συνέχεια θα αποσυμπιέσουμε τα περιεχόμενα του πακέτου και θα αλλάξουμε τον ιδιοκτήτη του καταλόγου ώστε να ανήκει στον hduser.

Αυτό επιτυγχάνεται με τις παρακάτω εντολές :

```
$ cd /usr/local
$ sudotarxzfhadoop-1.0.3.tar.gz
$ sudomvhadoop-1.0.3 hadoop
$ sudochown -Rhduser:hadoophadoop
```

4.9 Επεξεργασία αρχείων - Ρυθμίσεις

Αρχικά θα προσθέσουμε κάποιες εντολές και μεταβλητές στο αρχείο $HOME/.bashrc του χρήστη hduser. Για να το πετύχουμε αυτό απλά προσθέτουμε τις παρακάτω γραμμές στο τέλος του αρχείου.

```
# Set Hadoop-related environment variables
export HADOOP_HOME=/usr/local/hadoop

# Set JAVA_HOME (we will configure JAVA_HOME directly for Hadoop later on)
export JAVA_HOME=/usr/lib/jvm/java-6-sun

# Some convenient aliases and functions for running Hadoop-related commands
unalias fs &> /dev/null
alias fs="hadoop fs"
unalias hls &> /dev/null
alias hls="fs -ls"

# Requires installed 'lzop' command.
#
lzohead () {
    hadoop fs -cat $1 | lzop -dc | head -1000 | less
}

# Add Hadoop bin/ directory to PATH
export PATH=$PATH:$HADOOP_HOME/bin
```

Η μόνη μεταβλητή περιβάλλοντος που πρέπει να ρυθμίσουμε για το Hadoop είναι η JAVA_HOME. Για να το πετύχουμε αυτό πρέπει να ανοίξουμε με έναν editor της επιλογής μας το αρχείο *conf/hadoop-env.sh* το οποίο βρίσκεται στον κατάλογο που εγκαταστήσαμε το hadoop και να θέσουμε τη μεταβλητή JAVA_HOME στον κατάλογο που την εγκαταστήσαμε. Πρακτικά αλλάζουμε αυτό :

```
# The java implementation to use.  Required.
# export JAVA_HOME=/usr/lib/j2sdk1.5-sun
```

Σε αυτό :

```
# The java implementation to use.  Required.
export JAVA_HOME=/usr/lib/jvm/java-6-sun
```

Έπειτα πρέπει να ρυθμίσουμε τους καταλόγους που το hadoop θα αποθηκεύει τα δεδομένα, τις θύρες δικτύου στις οποίες θα ακούει κ.α. Αρχικά ρυθμίζουμε τον κατάλογο που το hadoop αποθηκεύει τα προσωρινά δεδομένα, και για το τοπικό σύστημα αρχείων αλλά και για το HDFS. Πρέπει εμείς να δημιουργήσουμε τον κατάλογο αυτό και να του αποδώσουμε την επιθυμητή ιδιοκτησία και άδειες.

```
$ sudomkdir -p /app/hadoop/tmp
$ sudochownhduser:hadoop /app/hadoop/tmp
# ...και αν θέλουμε μεγαλύτερη ασφάλεια, chmodfrom 755 to 750...
$ sudochmod 750 /app/hadoop/tmp
```

Στη συνέχεια προσθέτουμε κάποια δεδομένα στα παρακάτω αναφερόμενα XML αρχεία σε συγκεκριμένο σημείο (στην περιοχή <configuration>). Τα αρχεία αυτά εντοπίζονται στον κατάλογο που εγκαταστήσαμε το hadoop.

Στο αρχείο *conf/core-site.xml* :

```
<property>
<name>hadoop.tmp.dir</name>
<value>/app/hadoop/tmp</value>
<description>A base for other temporary directories.</description>
</property>
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:54310</value>
<description>The name of the default file system.
</description>
</property>
```

Στο αρχείο *conf/mapred-site.xml* :

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

Στο αρχείο *conf/hdfs-site.xml* :

```
<property>
<name>dfs.replication</name>
<value>1</value>
<description>Default block replication.
  The actual number of replications can be specified when the file is created.
  The default is used if replication is not specified in create time.
</description>
</property>
```

ΕΠΙΛΟΓΟΣ

Στο σημείο αυτό έχουμε ουσιαστικά ολοκληρώσει την εγκατάσταση του hadoop στον υπολογιστή. Ο συγκεκριμένος υπολογιστής μπορεί να χρησιμοποιηθεί σαν ένα ψευδοκατανεμημένο σύστημα το οποίο μπορεί να χρησιμοποιηθεί για εκμάθηση της συγκεκριμένης πλατφόρμας και είναι πλήρως λειτουργικός.

Στο επόμενο κεφάλαιο θα προχωρήσουμε στο πώς θα τροποποιήσουμε την εγκατάσταση μας για να προσθέσουμε τους πρώτους κόμβους οι οποίοι θα λειτουργούν σαν slaves.

ΚΕΦΑΛΑΙΟ 5
5. Οδηγός εγκατάστασης του Hadoop (Multi-Node).

5.1 Εισαγωγή

Στη συνέχεια του οδηγού θα δούμε τη διαδικασία με την οποία μπορούμε να προσθέσουμε κόμβους στο δίκτυο μας. Αυτό προϋποθέτει απαραίτητα να έχει γίνει η εγκατάσταση του hadoop, όπως περιγράφεται στο προηγούμενο κεφάλαιο, σε κάθε υπολογιστή που θέλουμε να προσθέσουμε. Στην συγκεκριμένη εργασία και λόγω του μικρού αριθμού των κόμβων ο master υπολογιστής εκτελεί και χρέη slave. Για την ύπαρξη ενός υπολογιστή «καθαρά» σαν master θα πρέπει να γίνουν κάποιες επιπλέον ρυθμίσεις.

5.2 Ρύθμιση διευθύνσεων IP

Αρχική και απαραίτητη προϋπόθεση οι υπολογιστές μας να συνδέονται μέσω δικτύου. Σε περίπτωση όπως και της δικής μου εργασίας ο απλούστερος τρόπος είναι και αυτός που ακολουθήθηκε. Σύνδεση των υπολογιστών μέσω ενός router και απόδοση στατικής τοπικής IP διεύθυνσης ώστε να μην αλλάζει κάθε φορά που επανεκκινεί ο router.

Η μόνη ρύθμιση που χρειάζεται να γίνει σε επίπεδο λογισμικού σε αυτή τη φάση είναι να προσθέσουμε τις IP διευθύνσεις του master αλλά και των slaves σε κάθε ένα υπολογιστή στο αρχείο */etc/hosts*.

```
192.168.0.70    master
192.168.0.72    slave1
192.168.0.73    slave2
192.168.0.75    slave3
```

5.3 Επιπλέον παραμετροποίηση του SSH

Ο hduser στον υπολογιστή master θα πρέπει να είναι εφικτό να συνδεθεί με τον δικό του λογαριασμό στον master (εφόσον είπαμε ότι ο master θα λειτουργεί και σαν slave) αλλά και με τον αντίστοιχο λογαριασμό hduser στους slaves .

Αυτό θα πρέπει να γίνεται μέσω SSH και χωρίς κωδικό για να μη χρειάζεται να τον πληκτρολογούμε κάθε φορά που γίνεται μία σύνδεση από τον master σε κάποιον slave υπολογιστή και αντίστροφα. Το μόνο που έχουμε να κάνουμε είναι να προσθέσουμε το δημόσιο SSH κλειδί του hduser@master στα εξουσιοδοτημένα κλειδιά του hduser@slave σε όλους τους slaves. Αυτό επιτυγχάνεται με την παρακάτω εντολή :

```
hduser@master:~$ ssh-copy-id -i $HOME/.ssh/id_rsa.pubhduser@slave
```

Η εντολή αυτή αντιγράφει αυτόματα το δημόσιο κλειδί, δημιουργεί τον απαραίτητο κατάλογο και τις κατάλληλες άδειες όπου είναι απαραίτητο.

Το επόμενο βήμα είναι απαραίτητο για να εγγραφεί το κλειδί του slave στα εξουσιοδοτημένα κλειδιά του master. Απλά συνδεόμαστε από τον master στον slave :

```
hduser@master:~$ ssh slave
The authenticity of host 'slave (192.168.0.2)' can't be established.
RSA key fingerprint is 74:d7:61:86:db:86:8f:31:90:9c:68:b0:13:88:52:72.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'slave' (RSA) to the list of known hosts.
Ubuntu 10.04
...
hduser@slave:~$
```

Και από τον master στον master για επιβεβαίωση σωστής επικοινωνίας :

```
hduser@master:~$ ssh master
The authenticity of host 'master (192.168.0.1)' can't be established.
RSA key fingerprint is 3b:21:b3:c0:21:5c:7c:54:2f:1e:2d:96:79:eb:7f:95.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'master' (RSA) to the list of known hosts.
Linuxmaster 2.6.20-16-386 #2 ThuJun 7 20:16:13 UTC 2007 i686
...
hduser@master:~$
```

5.4 Επιπλέον Ρυθμίσεις αρχείων Hadoop

Σε πρώτη φάση πρέπει να γίνουν κάποιες ρυθμίσεις σε δύο αρχεία στον master. Στο αρχείο *conf/masters*θα ρυθμίσουμε σε ποια μηχανήματα το hadoop θα ξεκινήσει όταν του ζητηθεί τους secondaryNameNodes . Σε αυτή την υλοποίηση ο υπολογιστής αυτός είναι ο master. Ο primaryNameNode και ο JobTracker πάντα θα ξεκινούν στον υπολογιστή από τον οποίο θα ενεργοποιούμε το hadoop, δηλαδή τον master. Άρα επεξεργαζόμαστε το αρχείο *conf/masters*στον master ώστε να έχει μία μόνο εγγραφή :

```
master
```

Στο αρχείο *conf/slaves* ρυθμίζουμε ποιοι υπολογιστές θέλουμε να λειτουργούν σαν slaves. Σε ποιους υπολογιστές δηλαδή θα ξεκινάει το hadoop και θα τρέχει τους DataNodes και τους TaskTrackers. Επομένως ρυθμίζουμε το αρχείο στον master ώστε να περιέχει μόνο τις εγγραφές που εμείς θέλουμε :

```
Master
slave1
slave2
slave3
```

Δεν περιγράφεται και δεν χρησιμοποιήθηκε κατά τη διάρκεια της εργασίας αλλά μπορούμε να προσθέσουμε κόμβους σαν slaves κατά τη διάρκεια εκτέλεσης του hadoop cluster. Οι ρυθμίσεις που έγιναν παραπάνω ουσιαστικά ρυθμίζουν ποιοι υπολογιστές θα κάνουν ‘τι’ κατά την εκκίνηση του hadoop και δεν αποκλείουν άλλους από το να προστεθούν στη συνέχεια.

Στη συνέχεια θα δούμε ρυθμίσεις που πρέπει να γίνουν σε όλους τους υπολογιστές του cluster σε συγκεκριμένα XML αρχεία.

Αρχικά θα αλλάξουμε την παράμετρο του fs.default.name στο αρχείο *conf/core-site.xml* το οποίο ορίζει τον NameNode, δηλαδή τον master του HDFS και την θύρα στην οποία θα ακούει :

```
<property>
<name>fs.default.name</name>
<value>hdfs://master:54310</value>
<description>The name of the default file system.
</description>
</property>
```

Δεύτερον θα αλλάξουμε την παράμετρο του mapred.job.tracker στο αρχείο *conf/mapred-site.xml* το οποίο ορίζει τον JobTracker, δηλαδή τον master του MapReduce και την θύρα στην οποία θα ακούει :

```
<property>
<name>mapred.job.tracker</name>
<value>master:54311</value>
<description>The host and port that the MapReduce job tracker runs
  at.  If "local", then jobs are run in-process as a single map
  and reduce task.
</description>
</property>
```

Τρίτον και τελευταίο θα αλλάξουμε την παράμετρο dfs.replication στο αρχείο *conf/hdfs-site.xml*το οποίο ουσιαστικά ορίζει σε πόσα μηχανήματα θα αναπαράγεται ένα μοναδικό αρχείο πριν καταστεί διαθέσιμο. Πρακτικά το θέτουμε αυτό ίσο με όσα slave μηχανήματα έχουμε ή πιο σωστά με όσους υπολογιστές έχουμε, στους οποίους τρέχουν οι DataNodes.

```
<property>
<name>dfs.replication</name>
<value>2</value>
<description>Default block replication.
  The actual number of replications can be specified when the file is created.
  The default is used if replication is not specified in create time.
</description>
</property>
```

ΕΠΙΛΟΓΟΣ

Στο σημείο αυτό έχουμε ολοκληρώσει με επιτυχία την εγκατάσταση του cluster και μπορούμε να προχωρήσουμε στο επόμενο κεφάλαιο που θα δούμε πως ακριβώς ενεργοποιούμε και απενεργοποιούμε το Hadoop και με ποιον τρόπο ελέγχουμε την σωστή λειτουργία αυτού.

ΚΕΦΑΛΑΙΟ 6
6. Ενεργοποίηση και απενεργοποίηση του cluster.

6.1 Εισαγωγή

Πριν ξεκινήσουμε για πρώτη φορά το cluster θα πρέπει να αρχικοποιήσουμε το σύστημα αρχείων του hadoop το HDFS. Αυτό θα γίνει μέσω του NameNode. Ποτέ δεν πρέπει να αρχικοποιούμε το HDFS κατά τη διάρκεια εκτέλεσης του hadoop γιατί θα χαθούν όλα τα δεδομένα. Για να πετύχουμε την αρχικοποίηση απλά τρέχουμε την παρακάτω εντολή :

```
hduser@master:/usr/local/hadoop$ bin/hadoop namenode –format
```

6.2 Εκκίνηση του cluster

Η εκκίνηση του cluster πραγματοποιείται σε 2 βήματα :

1. Πρώτα ξεκινάμε τους HDFS daemons δηλαδή τους NameNode στον master και τους DataNode σε όλους τους slaves.
2. Στη συνέχεια ξεκινάμε τους MapReduce daemons δηλαδή τους JobTracker στον master και τους TaskTracker σε όλους τους slaves.

Στον master τρέχουμε την εντολή *bin/start-dfs.sh* η οποία έχει ως αποτέλεσμα να ξεκινήσουν τα αναφερθέντα στο #1.

```
hduser@master:/usr/local/hadoop$ bin/start-dfs.sh
starting namenode, logging to /usr/local/hadoop/bin/../logs/hadoop-hduser-namenode-master.out
slave: Ubuntu 10.04
slave: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hduser-datanode-slave.out
master: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hduser-datanode-master.out
master: starting secondarynamenode, logging to usr/local/hadoop/bin/../logs/hadoop-hduser-secondarynamenode-master.out
hduser@master:/usr/local/hadoop$
```

Η επιτυχία της εντολής μπορεί να ελεγχθεί στους slaves ελέγχοντας το αρχείο αναφορών *logs/hadoop-hduser-datanode-slave.log*που βρίσκεται στον κατάλογο εγκατάστασης του hadoop.

Στον master τρέχουμε την εντολή *bin/start-mapred.sh* η οποία έχει ως αποτέλεσμα να ξεκινήσουν τα αναφερθέντα στο #2.

hduser@master:/usr/local/hadoop$ bin/start-mapred.sh
starting jobtracker, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-jobtracker-master.out
slave: Ubuntu 10.04
slave: starting tasktracker, logging to /usr/local/hadoop/bin/../logs/hadoop-hduser-tasktracker-slave.out
master: starting tasktracker, logging to /usr/local/hadoop/bin/../logs/hadoop-hduser-tasktracker-master.out
hduser@master:/usr/local/hadoop$

Η επιτυχία της εντολής μπορεί να ελεγχθεί στους slaves ελέγχοντας το αρχείο αναφορών *logs/hadoop-hduser-tasktracker-slave.log* που βρίσκεται στον κατάλογο εγκατάστασης του hadoop.

Χρησιμοποιώντας την εντολή *jps* μπορούμε να δούμε ποιες διεργασίες της java είναι ενεργές αυτή τη στιγμή στον υπολογιστή μας.

Το αποτέλεσμα της εντολής στον master πρέπει να είναι :

hduser@master:/usr/local/hadoop$ jps
Jps
NameNode
TaskTracker
DataNode
JobTracker
SecondaryNameNode
hduser@master:/usr/local/hadoop$

Το αποτέλεσμα της εντολής στον slave πρέπει να είναι :

hduser@slave:/usr/local/hadoop$ jps
DataNode
TaskTracker
Jps
hduser@slave:/usr/local/hadoop$

6.3 Απενεργοποίηση του cluster

Η απενεργοποίηση του cluster πραγματοποιείται σε 2 βήματα ακριβώς αντίστροφα από ότι στην ενεργοποίηση :

1. Πρώτα τερματίζουμε τους MapReduce daemons δηλαδή τους JobTracker στον master και τους TaskTracker σε όλους τους slaves.
2. Στη συνέχεια τερματίζουμε τους HDFS daemons δηλαδή τους NameNode στον master και τους DataNode σε όλους τους slaves.

Τρέχουμε την εντολή *bin/stop-mapred.sh* η οποία τερματίζει τα αναφερθέντα στο #1. Στο σημείο αυτό θα πρέπει να έχουν μείνει μόνον οι παρακάτω διεργασίες ενεργές :

Στον master :
hduser@master:/usr/local/hadoop$ jps
14799 NameNode
18386 Jps
14880 DataNode
14977 SecondaryNameNode
hduser@master:/usr/local/hadoop$

Στον slave :
hduser@slave:/usr/local/hadoop$ jps
15183 DataNode
18636 Jps
hduser@slave:/usr/local/hadoop$

Τρέχουμε την εντολή *bin/stop-dfs.sh* η οποία τερματίζει τα αναφερθέντα στο #2. Στο σημείο αυτό θα πρέπει να έχουν μείνει μόνον οι παρακάτω διεργασίες ενεργές :

Στον master :
hduser@master:/usr/local/hadoop$ jps
18670 Jps
hduser@master:/usr/local/hadoop$

Στον slave :
hduser@slave:/usr/local/hadoop$ jps
18894 Jps
hduser@slave:/usr/local/hadoop$

ΕΠΙΛΟΓΟΣ

Στα κεφάλαια που προηγήθηκαν έχουμε κατά σειρά καλύψει το θεωρητικό μέρος του μοντέλου που θα χρησιμοποιήσουμε, δηλαδή το MapReduce, καθώς και της υλοποίησης την οποία θα χρησιμοποιήσουμε, το Hadoop.

Έχουμε κάνει αναφορά στις βασικές λειτουργίες και εντολές του Hadoop και έχουμε ολοκληρώσει με επιτυχία την εγκατάστασή του στο δοκιμαστικό cluster που έχουμε στήσει.

Μπορούμε λοιπόν να προχωρήσουμε στο επόμενο κεφάλαιο και να γράψουμε (demo) κώδικα σε Java τον οποίο θα τρέξουμε στο cluster μας.

ΚΕΦΑΛΑΙΟ 7
7. Υλοποίηση προγράμματος σε java.

7.1 Εισαγωγή

Στα πλαίσια της εργασίας έχω αναπτύξει κώδικα γραμμένο σε Java. Το πρόγραμμα που κατασκεύασα επεξεργάζεται ένα αρχείο δεδομένων από τον server του Α.Τ.Ε.Ι.Θ. και παράγει τα επιθυμητά αποτελέσματα. Πιο συγκεκριμένα το αρχείο που χρησιμοποιήθηκε σαν είσοδος για τα δεδομένα στο πρόγραμμα είναι ένα κομμάτι από τις βαθμολογίες των φοιτητών του ΤΕΙ και είναι της παρακάτω μορφής :

[Μάθημα1];[Βαθμός]
[Μάθημα2];[Βαθμός]
.
.
.
.
[Μάθημα v]΄[Βαθμός]

Οι εγγραφές που περιέχει το αρχείο αυτό είναι περίπου 1.600.000 και είναι σε μέγεθος περίπου 780 Mbyte. Υπάρχουν πολλαπλές εγγραφές για κάθε μάθημα αλλά μόνον ένας βαθμός ανά μάθημα για κάθε φοιτητή. Οι τιμές του πεδίου βαθμός είτε είναι NULL που σημαίνει αποτυχία στο μάθημα είτε κυμαίνονται από 0.00 έως 1.00. Θεώρησα σαν βάση το 0.50 για να εξάγω σαν αποτέλεσμα τον αν ο συγκεκριμένος φοιτητής έχει περάσει το μάθημα η όχι.

Συγκεντρωτικά το πρόγραμμα επεξεργάζεται όλες τις εγγραφές και το παραγόμενο αποτέλεσμα είναι ένα αρχείο που περιέχει μία εγγραφή για κάθε μάθημα και το ποσοστό επιτυχίας και αποτυχίας των φοιτητών σε αυτό :

[Μάθημα1] Success:[….] Fail:[….]
[Μάθημα2] Success:[….] Fail:[….] .
.
.
.
[Μάθημα v] Success:[….] Fail:[….]

Στη συνέχεια γίνεται μία λεπτομερής ανάλυση του κώδικα.

7.2 Ανάλυση του κώδικα

Πρώτο και πολύ σημαντικό βήμα στη συγγραφή του προγράμματος είναι η εισαγωγή των απαραίτητων βιβλιοθηκών που χρειάζεται το Hadoop και το MapReduce για να τρέξουν.

```
import java.io.IOException;
import java.util.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

Σαν σχεδιάγραμμα θα λέγαμε του κώδικα αυτός αποτελείται από μία κύρια κλάση η οποία περιέχει άλλες δυο, την κλάση mapper που υλοποιεί τη μέθοδο map, και την κλάση reducer που υλοποιεί τη μέθοδο reduce . Φυσικά στην κύρια κλάση βρίσκεται και η main του προγράμματος μας ή driver όπως ονομάζεται στα προγράμματα MapReduce.

```
publicclass Combiner {
        publicstaticclass Map extends Mapper<Text, Text, Text, Text> {...}
        publicstaticclass Reduce extends Reducer<Text, Text, Text, Text> {...}
        publicstaticvoidmain(String[] args) throwsException {...}

}
```

Στη συνέχεια θα δούμε αναλυτικά τι ακριβώς συμβαίνει σε κάθε μία κλάση ξεχωριστά και θα εξετάσουμε τις παραμέτρους του driver . Ολόκληρος ο κώδικας παρατίθεται σαν παράρτημα στο Παράρτημα Β’ .

7.3 Η κλάση Map

Η κλάση Map υλοποιεί τη συνάρτηση **map** που παίρνει σαν είσοδο ένα ζεύγος κλειδιού και, αφού εφαρμόσει την συνάρτηση του χρήστη πάνω τους, παράγει ένα ενδιάμεσο ζεύγος κλειδιού, το οποίο στην συνέχεια θα δώσει στην συνάρτηση reduce, η οποία με τη σειρά της ενώνει όλες αυτές τις ενδιάμεσες τιμές που σχετίζονται με το ίδιο ενδιάμεσο κλειδί.

```
publicstaticclass Map extends Mapper<Text, Text, Text, Text> {

        private Text word = new Text();
        public void map(Text key, Text value, Context context) throws IOException,
        InterruptedException {

                StringTokenizer token = newStringTokenizer(value·toString());

                while (token·hasMoreTokens()){
                        word·set(token·nextToken());
        context·write(key, word);
        }

        }

}
```

Το ζεύγος κλειδιών που παίρνει σαν είσοδος η μέθοδος map είναι του τύπου Text και αυτό γιατί και οι βαθμοί διαβάζονται σαν Strings (μπορεί να έχουν την τιμή NULL).Η μέθοδος περνάει μία μία τις γραμμές του αρχείου και τις χωρίζει σε tokens. Στη main θα δούμε πως δηλώνουμε με ποιόν ακριβώς τρόπο θέλουμε να διαχωρίζει τα tokens η συνάρτηση map, και με ποιά ακριβώς εντολή θα αλλάξουμε τις ρυθμίσεις του mapper ώστε να θεωρεί σαν αλλαγή λέξης το ‘;’. Στη συνέχεια δημιουργεί ζευγάρια κλειδιών της μορφής <ΜΑΘΗΜΑ, ΒΑΘΜΟΣ>

Αυτά τα ζεύγη κλειδιών θα περαστούν αργότερα στην μέθοδο reduce η οποία θα είναι και αυτή που θα παράγει το τελικό αποτέλεσμα. Δεν υλοποιήθηκε στο συγκεκριμένο παράδειγμα λόγω του μικρού αριθμού εγγραφών (1.600.000 εγγραφές για το hadoop είναι λίγο !!), αλλά μπορεί να υλοποιηθεί μία κλάση ενδιάμεσα από την map και τη reduce, η combine, η οποία να ομαδοποιεί με βάση κάποιο κλειδί τα δεδομένα κάνοντας έτσι πιο εύκολη τη δουλειά του reducer στη συνέχεια.

7.4 Η κλάση Reduce

Η κλάση Reduce υλοποιεί τη συνάρτηση **reduce** η οποία προσπαθεί να συγχωνέψει όλες αυτές τις τιμές εφαρμόζοντας τον κώδικα του χρήστη, για να μπορέσει πιθανώς να δημιουργήσει ένα μικρότερο σύνολο τιμών. Συνήθως δημιουργείται μια τιμή εξόδου για κάθε εκτέλεση της reduce.

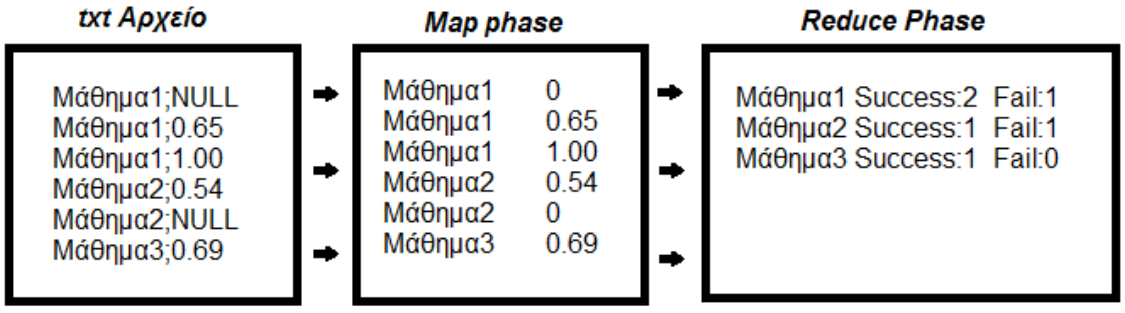
Ο κώδικας που βρίσκεται μέσα στη μέθοδο αυτή είναι και ο κώδικας που εκτελείται σε κάθε κόμβο του cluster. Κάθε ένας κόμβος δηλαδή αναλαμβάνει να τρέξει τον κώδικα για το κομμάτι του αρχείου που του έχει ανατεθεί.

```
publicstaticclass Reduce extends Reducer<Text, Text, Text, Text> {

        private Text result = new Text();
        public void reduce(Text key, Iterable<Text> values,Context context) throws
        IOException, InterruptedException {

                int success = 0;
                int fail = 0;
                for (Text val : values){
                        if (val.toString().equals("NULL")){
                        val.set("0");
                        }
                        if(Double.parseDouble(val.toString()) < 0.5){
                                fail += 1;
                        }
                        else{
                                success += 1;
                        }

                }
                result.set(" Success :" + Integer.toString(success)+" Fail :"+
                Integer.toString(fail));

                context.write(key, result);
        }

}
```

Εφόσον η μέθοδος map παρήγαγε κλειδιά του τύπου Text άρα και η reduce είναι υποχρεωμένη να διαβάσει κλειδιά του ίδιου τύπου. Για λόγους απλότητας καθώς δεν χρειάστηκε να γίνουν πράξεις ανάμεσα στους βαθμούς, και η reduce παράγει κλειδιά τύπου Text.

Αυτό που κάνει στη ουσία η μέθοδος reduce είναι να ελέγχει αρχικά αν το δεύτερο σκέλος του ζεύγους κλειδιού που έλαβε είναι ίσο με NULL και να το μετατρέπει σε '0' για να μπορεί να γίνει η σύγκριση με τη βάση στην επόμενη "if" και να αποφασιστεί αν ο φοιτητής πέτυχε ή όχι στο μάθημα. Σε περίπτωση επιτυχίας αυξάνεται ο μετρητής success αλλιώς ο μετρητής fail.

Τα αποτελέσματα των μετρητών μετατρέπονται σε strings για να μπορούν να γραφούν στην έξοδο της reduce. Οι μετρητές μηδενίζουν έξω από την "for" και άρα κάθε φορά ο reducer αναλαμβάνει ζεύγος κλειδιού με διαφορετικό πρώτο σκέλος.



Σχήμα 7.1 Τα στάδια εκτέλεσης του Hadoop

Στο παραπάνω σχήμα φαίνεται η μορφοποίηση του αρχείου κατά τη διάρκεια εκτέλεσης του προγράμματος. Στην πραγματικότητα το αρχείο δεν μεταβάλλεται καθόλου αφού τα δεδομένα εγγράφονται σε ξεχωριστό αρχείο.

7.3 H Main (driver)

```
publicstaticvoid main(String[] args) throws Exception {
        Configuration conf = newConfiguration();
        //chabge the dilimiter from deafult to ";"·

conf·set("mapreduce·input·keyvaluelinerecordreader·key·value·separator", ";");

         Job job = newJob(conf, "Combiner");
job·setJarByClass(Combiner·class);
job·setOutputKeyClass(Text·class);
job·setOutputValueClass(Text·class);
job·setMapperClass(Map·class);
job·setReducerClass(Reduce·class);
job·setInputFormatClass(KeyValueTextInputFormat·class);
job·setOutputFormatClass(TextOutputFormat·class);

FileInputFormat·addInputPath(job, new Path("/user/hduser/studcour"));
FileOutputFormat·setOutputPath(job, new Path("/user/hduser/CombinerOut"));

job·waitForCompletion(true);
        }

}
```

Αυτό που αξίζει να προσέξουμε στην main είναι το γεγονός ότι εδώ δηλώνουμε το όνομα της εργασίας που θα εκτελέσει το hadoop καθώς και τις κλάσεις που θα χρησιμοποιήσει για το mapping, το reducing, και το τι τύπου θα είναι τα κλειδιά που θα πάρει σαν είσοδο και αυτά που θα δημιουργήσει.

Εδώ δηλώνουμε επίσης τους καταλόγους input και output. Από ποίον κατάλογο θα λάβει δηλαδή τα δεδομένα εισόδου και σε ποιόν κατάλογο επιθυμούμε να μας εγγράψει τα αποτελέσματα.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Το Hadoop αλλά και γενικότερα το προγραμματιστικό μοντέλο MapReduce έχει αποδειχτεί να είναι αρκετά αποτελεσματικό και αποδοτικό στην παράλληλη εκτέλεση αρκετών και διάφορων εργασιών. Αποδίδω αυτή την επιτυχία σε διάφορους λόγους:

• Καταρχάς το μοντέλο είναι πολύ εύκολο στη χρήση του ακόμα και για προγραμματιστές με λίγη ή και καθόλου εμπειρία στον παράλληλο προγραμματισμό, αφού κρύβει τις λεπτομέρειες του παραλληλισμού, της ανάκαμψης από σφάλματα, των locality optimizations και του load balancing. Ουσιαστικά ο προγραμματιστής αυτό που έχει να κάνει, για να τρέξει το πρόγραμμα του με το Hadoop, είναι να υλοποιήσει τις δυο συναρτήσεις map και reduce με τέτοιο τρόπο ώστε να λύνουν το πρόβλημα του και να αφήσει τα υπόλοιπα στο Hadoop.

• Ένα μεγάλο εύρος προβλημάτων μπορούν σχετικά εύκολα να εκφραστούν σαν MapReduce υπολογισμοί. Για παράδειγμα το μοντέλο αυτό μπορεί να εφαρμοστεί σε μηχανές αναζήτησης, για ταξινόμηση, για data mining, για machine learning και αρκετές άλλες εφαρμογές.

• Το Hadoop μπορεί εύκολα να γίνει scale για χρήση σε εκατοντάδες ή και χιλιάδες επεξεργαστές με τρόπο που να εκμεταλλεύεται αποδοτικά τους πόρους των μηχανών πάνω στις οποίες τρέχει, διατηρώντας για τον προγραμματιστή την ίδια ευκολία εκτέλεσης με τη σειριακή εκτέλεση σε ένα και μοναδικό υπολογιστή.

• Το Hadoop διαθέτει μία απίστευτα ισχυρή δυνατότητα Self-Healing. Μπορεί οποιαδήποτε στιγμή αρκετοί κόμβοι ταυτόχρονα να δυσλειτουργούν ή να τεθούν

εκτός λειτουργίας για οποιοδήποτε λόγω όμως αυτό δεν θα το σταματήσει από το να φέρει εις πέρας την εργασία που του έχει ανατεθεί.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Free eBooks by Project Gutenberg, http://www.gutenberg.org/, Μάιος 2013

[2] Hadoop MapReduce, http://hadoop.apache.org/, Μάιος 2013

[3] Hadoop MapReduce, Hadoop Wiki, http://wiki.apache.org/hadoop/HadoopMapReduce, Μάιος 2013

[4] Jeffrey Dean and Sanjay Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, Google, Inc., 2004

[5] Java Hadoop Basics, http://java.dzone.com/articles/hadoop-basics-creating, Μάιος 2013

[6] MapReduce, Wikipedia, http://en.wikipedia.org/wiki/MapReduce, Μάιος 2013

[7] Michael G. Noll, Applied Research. Big Data.Distributed Systems. Open Source. http://www.michael-noll.com/, Μάιος 2013

[8] Tom White, “Hadoop: The Definitive Guide”, O’Reilly, 2009

[9] Ubuntu 10.04.4 LTS http://releases.ubuntu.com/lucid/, Μάιος 2013

[10] UnetBootIn http://unetbootin.sourceforge.net/, Μάιος 2013

ΠΑΡΑΡΤΗΜΑΤΑ

ΠΑΡΑΤΗΜΑ Α’

1. Hadoop Overview

All the hadoop commands are invoked by the bin/hadoop script. Running hadoop script without any arguments prints the description for all commands.

```
Usage: 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	Description
`--config confdir`	Overwrites the default Configuration directory. Default is ${HADOOP_HOME}/conf.
`GENERIC_OPTIONS`	The common set of options supported by multiple commands.
`COMMAND` `COMMAND OPTIONS`	Various commands with their options are described in the following sections. The commands have been grouped into User Commandsand Administration Commands.

1.1. Generic Options

Following are supported by dfsadmin, fs, fsckand job. Applications should implement Tool to support GenericOptions.

GENERIC_OPTION	Description

`-conf <configuration file>`	Specify an application configuration file.
`-D <property=value>`	Use value for given property.
`-fs <local|namenode:port>`	Specify a namenode.
`-jt <local|jobtracker:port>`	Specify a job tracker. Applies only to job.
`-files <comma separated list of files>`	Specify comma separated files to be copied to the map reduce cluster. Applies only to job.
`-libjars <comma seperated list of jars>`	Specify comma separated jar files to include in the classpath. Applies only to job.
`-archives <comma separated list of archives>`	Specify comma separated archives to be unarchived on the compute machines. Applies only to job.

2. User Commands

Commands useful for users of a hadoop cluster.

2.1. archive

Creates a hadoop archive. More information can be found at Hadoop Archives.

`Usage: hadoop archive -archiveName NAME <src>* <dest>`

COMMAND_OPTION	Description
`-archiveName NAME`	Name of the archive to be created.
`src`	Filesystem pathnames which work as usual with regular expressions.
`dest`	Destination directory which would contain the archive.

2.2. distcp

Copy file or directories recursively. More information can be found at Hadoop DistCp Guide.

`Usage: hadoop distcp <srcurl><desturl>`

COMMAND_OPTION	Description
`srcurl`	Source Url
`desturl`	Destination Url

2.3. fs

```
Usage: hadoop fs [GENERIC_OPTIONS] [COMMAND_OPTIONS]
```

Runs a generic filesystem user client.

The various COMMAND_OPTIONS can be found at Hadoop FS Shell Guide.

2.4. fsck

Runs a HDFS filesystem checking utility. See Fsckfor more info.

```
Usage: hadoop fsck [GENERIC_OPTIONS] <path> [-move | -delete
| -openforwrite] [-files [-blocks [-locations | -racks]]]
```

COMMAND_OPTION	Description
<path>	Start checking from this path.
-move	Move corrupted files to /lost+found
-delete	Delete corrupted files.
-openforwrite	Print out files opened for write.
-files	Print out files being checked.
-blocks	Print out block report.
-locations	Print out locations for every block.
-racks	Print out network topology for data-node locations.

2.5. jar

Runs a jar file. Users can bundle their Map Reduce code in a jar file and execute it using this command.

```
Usage: hadoop jar <jar> [mainClass] args...
```

The streaming jobs are run via this command. Examples can be referred from Streaming examples

Word count example is also run using jar command. It can be referred from Wordcount example

2.6. job

Command to interact with Map Reduce Jobs.

```
Usage: 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>]
```

COMMAND_OPTION	Description
`-submit <job-file>`	Submits the job.
`-status <job-id>`	Prints the map and reduce completion percentage and all job counters.
`-counter <job-id><group-name> <counter-name>`	Prints the counter value.
`-kill <job-id>`	Kills the job.
`-events <job-id><from-event-#> <#-of-events>`	Prints the events' details received by jobtracker for the given range.
`-history [all] <jobOutputDir>`	-history <jobOutputDir> prints job details, failed and killed tip details. More details about the job such as successful tasks and task attempts made for each task can be viewed by specifying the [all] option.
`-list [all]`	-list all displays all jobs. -list displays only jobs which are yet to complete.
`-kill-task <task-id>`	Kills the task. Killed tasks are NOT counted against failed attempts.
`-fail-task <task-id>`	Fails the task. Failed tasks are counted against failed attempts.
`-set-priority <job-id><priority>`	Changes the priority of the job. Allowed priority values are VERY_HIGH, HIGH, NORMAL, LOW, VERY_LOW

2.7. pipes

Runs a pipes job.

```
Usage: 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>]
```

COMMAND_OPTION	Description
`-conf <path>`	Configuration for job
`-jobconf <key=value>, <key=value>, ...`	Add/override configuration for job
`-input <path>`	Input directory
`-output <path>`	Output directory

`-jar <jar file>`	Jar filename
`-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

2.8. queue

command to interact and view Job Queue information

```
Usage : hadoop queue [-list] | [-info <job-queue-
name> [-showJobs]]
```

COMMAND_OPTION	Description
`-list`	Gets list of Job Queues configured in the system. Along with scheduling information associated with the job queues.
`-info <job-queue-name> [-showJobs]`	Displays the job queue information and associated scheduling information of particular job queue. If -showJobs options is present a list of jobs submitted to the particular job queue is displayed.

2.9. version

Prints the version.

```
Usage: hadoop version
```

2.10. CLASSNAME

hadoop script can be used to invoke any class.

```
Usage: hadoop CLASSNAME
```

Runs the class named CLASSNAME.

3. Administration Commands

Commands useful for administrators of a hadoop cluster .

3.1. balancer

Runs a cluster balancing utility. An administrator can simply press Ctrl-C to stop the rebalancing process. See Rebalancerfor more details.

```
Usage: hadoop balancer [-threshold <threshold>]
```

COMMAND_OPTION	Description
`-threshold <threshold>`	Percentage of disk capacity. This overwrites the default threshold.

3.2. daemonlog

Get/Set the log level for each daemon.

```
Usage: hadoop daemonlog -getlevel <host:port><name>
Usage: hadoop daemonlog -setlevel <host:port><name><level>
```

COMMAND_OPTION	Description
`-getlevel <host:port><name>`	Prints the log level of the daemon running at <host:port>. This command internally connects to http://<host:port>/logLevel?log=<name>
`-setlevel <host:port><name><level>`	Sets the log level of the daemon running at <host:port>. This command internally connects to http://<host:port>/logLevel?log=<name>

3.3. datanode

Runs a HDFS datanode.

`Usage: hadoop datanode [-rollback]`

COMMAND_OPTION	Description
`-rollback`	Rollsback the datanode to the previous version. This should be used after stopping the datanode and distributing the old hadoop version.

3.4. dfsadmin

Runs a HDFS dfsadmin client.

`Usage: 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]]`

COMMAND_OPTION	Description
`-report`	Reports basic filesystem information and statistics.
`-safemode enter | leave | get | wait`	Safe mode maintenance command. Safe mode is a Namenode state in which it 1. does not accept changes to the name space (read-only) 2. does not replicate or delete blocks. Safe mode is entered automatically at Namenode startup, and leaves safe mode automatically when the configured minimum percentage of blocks satisfies the minimum replication condition. Safe mode can also be entered manually, but then it can only be turned off manually as well.
`-refreshNodes`	Re-read the hosts and exclude files to update the set of Datanodes that are allowed to connect to the Namenode and those that should be decommissioned or recommissioned.
`-finalizeUpgrade`	Finalize upgrade of HDFS. Datanodes delete their previous version working directories, followed by Namenode doing the same. This completes the upgrade process.

`-upgradeProgress status | details | force`	Request current distributed upgrade status, a detailed status or force the upgrade to proceed.
`-metasave filename`	Save Namenode's primary data structures to <filename> in the directory specified by hadoop.log.dir property. <filename> will contain one line for each of the following 1. Datanodes heart beating with Namenode 2. Blocks waiting to be replicated

	3. Blocks currrently being replicated 4. Blocks waiting to be deleted
`-setQuota <quota> <dirname>...<dirname>`	Set the quota <quota> for each directory <dirname>. The directory quota is a long integer that puts a hard limit on the number of names in the directory tree. Best effort for the directory, with faults reported if 1. N is not a positive integer, or 2. user is not an administrator, or 3. the directory does not exist or is a file, or 4. the directory would immediately exceed the new quota.
`-clrQuota <dirname>...<dirname>`	Clear the quota for each directory <dirname>. Best effort for the directory. with fault reported if 1. the directory does not exist or is a file, or 2. user is not an administrator. It does not fault if the directory has no quota.
`-help [cmd]`	Displays help for the given command or all commands if none is specified.

3.5. jobtracker

Runs the MapReduce job Tracker node.

```
Usage: hadoop jobtracker
```

3.6. namenode

Runs the namenode. More info about the upgrade, rollback and finalize is at Upgrade Rollback

```
Usage: hadoop namenode [-format] | [-upgrade] | [-rollback]
```

```
| [-finalize] | [-importCheckpoint]
```

COMMAND_OPTION	Description
`-format`	Formats the namenode. It starts the namenode, formats it and then shut it down.
`-upgrade`	Namenode should be started with upgrade option after the distribution of new hadoop version.
`-rollback`	Rollsback the namenode to the previous version. This should be used after stopping the cluster and distributing the old hadoop version.
`-finalize`	Finalize will remove the previous state of the files system. Recent upgrade will become permanent. Rollback option will not be available anymore. After finalization it shuts the namenode down.
`-importCheckpoint`	Loads image from a checkpoint directory and save it into the current one. Checkpoint dir is read from property fs.checkpoint.dir

3.7. secondarynamenode

Runs the HDFS secondary namenode. See Secondary Namenodefor more info.

```
Usage: hadoop secondarynamenode [-checkpoint [force]]
| [-geteditsize]
```

COMMAND_OPTION	Description
`-checkpoint [force]`	Checkpoints the Secondary namenode if EditLog size >= fs.checkpoint.size. If -force is used, checkpoint irrespective of EditLog size.
`-geteditsize`	Prints the EditLog size.

3.8. tasktracker

Runs a MapReduce task Tracker node.

```
Usage: hadoop tasktracker
```

ΠΑΡΑΤΗΜΑ Β’

Demo Code In Java

```
1  import java.io.IOException;
2  import java.util.*;
3  import org.apache.hadoop.fs.Path;
4  import org.apache.hadoop.conf.*;
5  import org.apache.hadoop.io.*;
6  import org.apache.hadoop.mapreduce.*;
7  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
8  import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
9  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
10 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
11
12 publicclass Combiner {
13         publicstaticclass Map extends Mapper<Text, Text, Text, Text> {
14
15                 private Text word = new Text();
16                 publicvoid map(Text key, Text value, Context context) throws IOException,
17                 InterruptedException {
18
19                         StringTokenizer token = newStringTokenizer(value.toString());
20                         while (token.hasMoreTokens()){
21                                 word.set(token.nextToken());
22                         context.write(key, word);
23                          }
24
25                 }
26
27         }
28
29         publicstaticclass Reduce extends Reducer<Text, Text, Text, Text> {
```

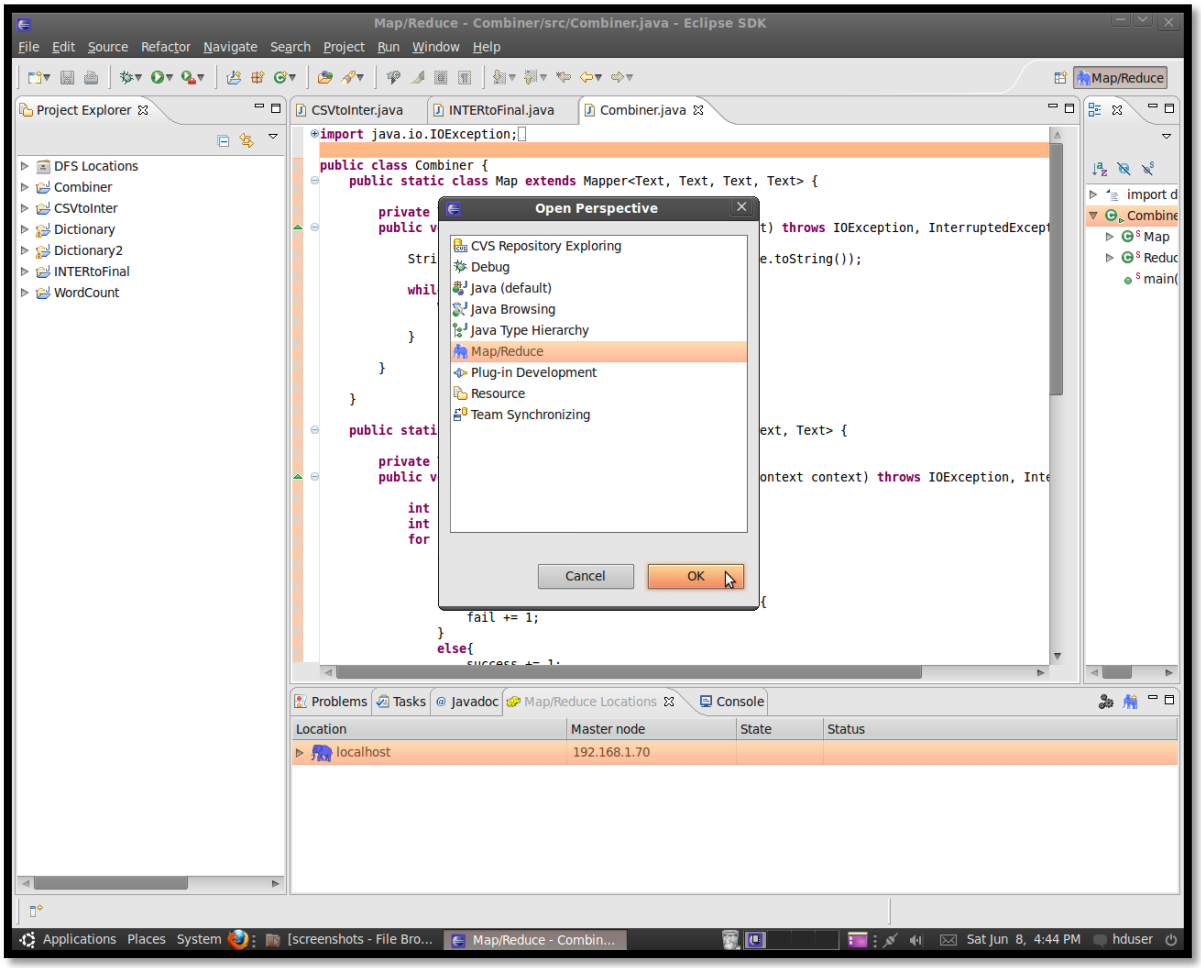
```
30
31                  private Text result = new Text();
32                  publicvoid reduce(Text key, Iterable<Text> values,Context context) throws
33                  IOException, InterruptedException {
34
35                          int success = 0;
36                          int fail = 0;
37                          for (Text val : values){
38                                  if (val.toString().equals("NULL")){
39                                          val.set("0");
40                                  }
41                                  if(Double.parseDouble(val.toString()) < 0.5){
42                                          fail += 1;
43                                  }
44                                  else{
45                                          success += 1;
46                                  }
47
48                          }
49                          result.set(" Success :" + Integer.toString(success)+" Fail :"+
50                          Integer.toString(fail));
51                          context.write(key, result);
52                  }
53          }
54
55          publicstaticvoid main(String[] args) throws Exception {
56                  Configuration conf = newConfiguration();
57                  //chabge the dilimiter from deafult to ";".
58                  conf.set("mapreduce.input.keyvaluelinerecordreader.key.value.separator", ";");
59
60                  Job job = newJob(conf, "Combiner");
61                  job.setJarByClass(Combiner.class);
62
63                  job.setOutputKeyClass(Text.class);
64                  job.setOutputValueClass(Text.class);
65
66                  job.setMapperClass(Map.class);
67                  job.setReducerClass(Reduce.class);
68
69                  job.setInputFormatClass(KeyValueTextInputFormat.class);
70                  job.setOutputFormatClass(TextOutputFormat.class);
71
72                  FileInputFormat.addInputPath(job, new Path("/user/hduser/studcour"));
73                  FileOutputFormat.setOutputPath(job, new Path("/user/hduser/CombinerOut"));
74
75                  job.waitForCompletion(true);
76          }
77
78  }
```

ΟΔΗΓΟΣ ΧΡΗΣΗΣ ΛΟΓΙΣΜΙΚΟΥ

Όσον αφορά το στήσιμο του Eclipse για να μπορούμε να γράψουμε κώδικα σε Java έτσι ώστε να έχουμε και τη βοήθεια του compiler ακολουθήθηκε η παρακάτω διαδικασία.

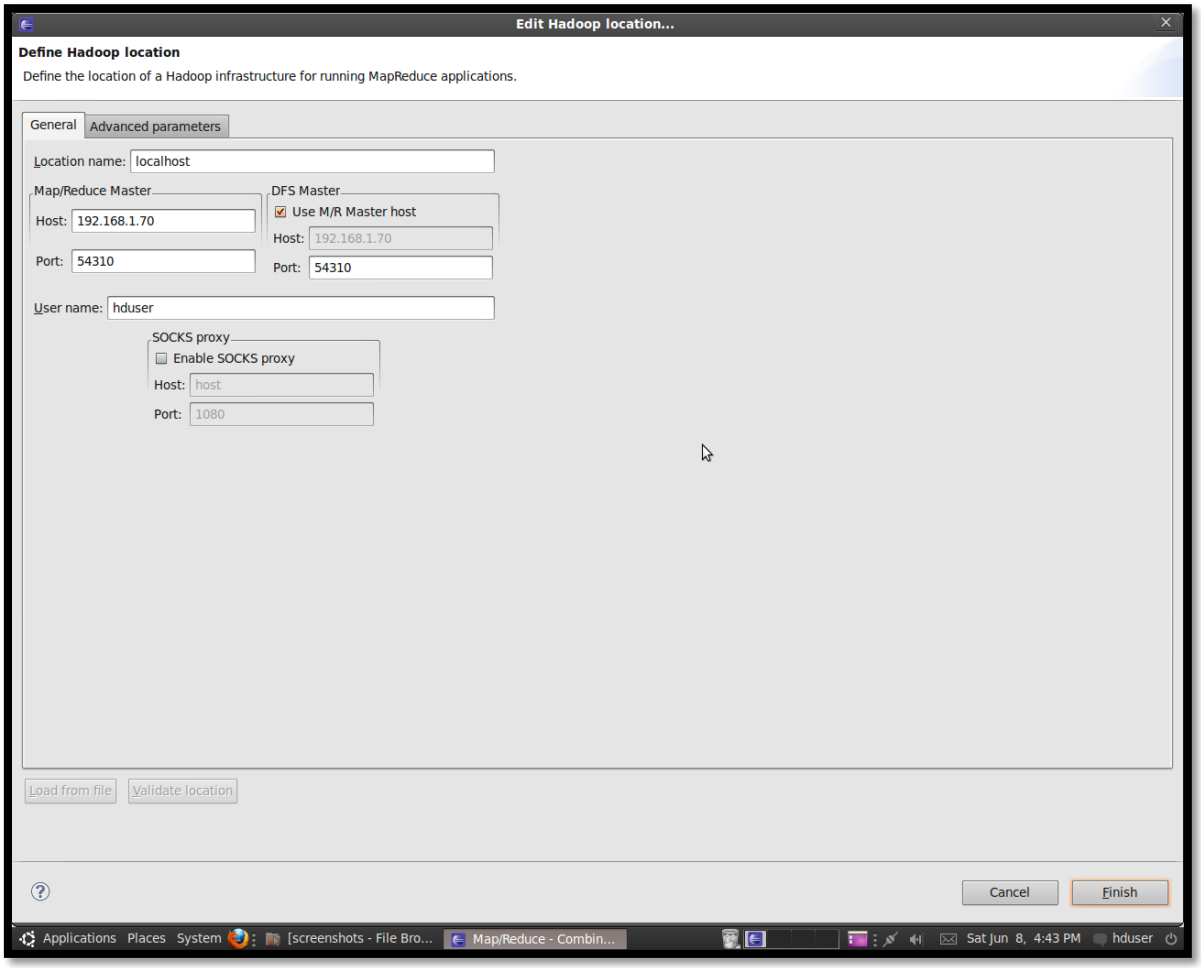
Πηγαίνουμε στο φάκελο που έχουμε εγκαταστήσει το Hadoop και στη συνέχεια στο φάκελο με το όνομα contrib., Μετά ανοίγουμε το φάκελο που λέγεται eclipse-plug-in. Αυτό το φάκελο τον αντιγράφουμε μέσα στο φάκελο plugins που βρίσκεται στον κατάλογο που έχουμε εγκαταστήσει το Eclipse.

Ανοίγουμε το Eclipse και πατάμε Window – Open Perspective – Other – Map/Reduce.



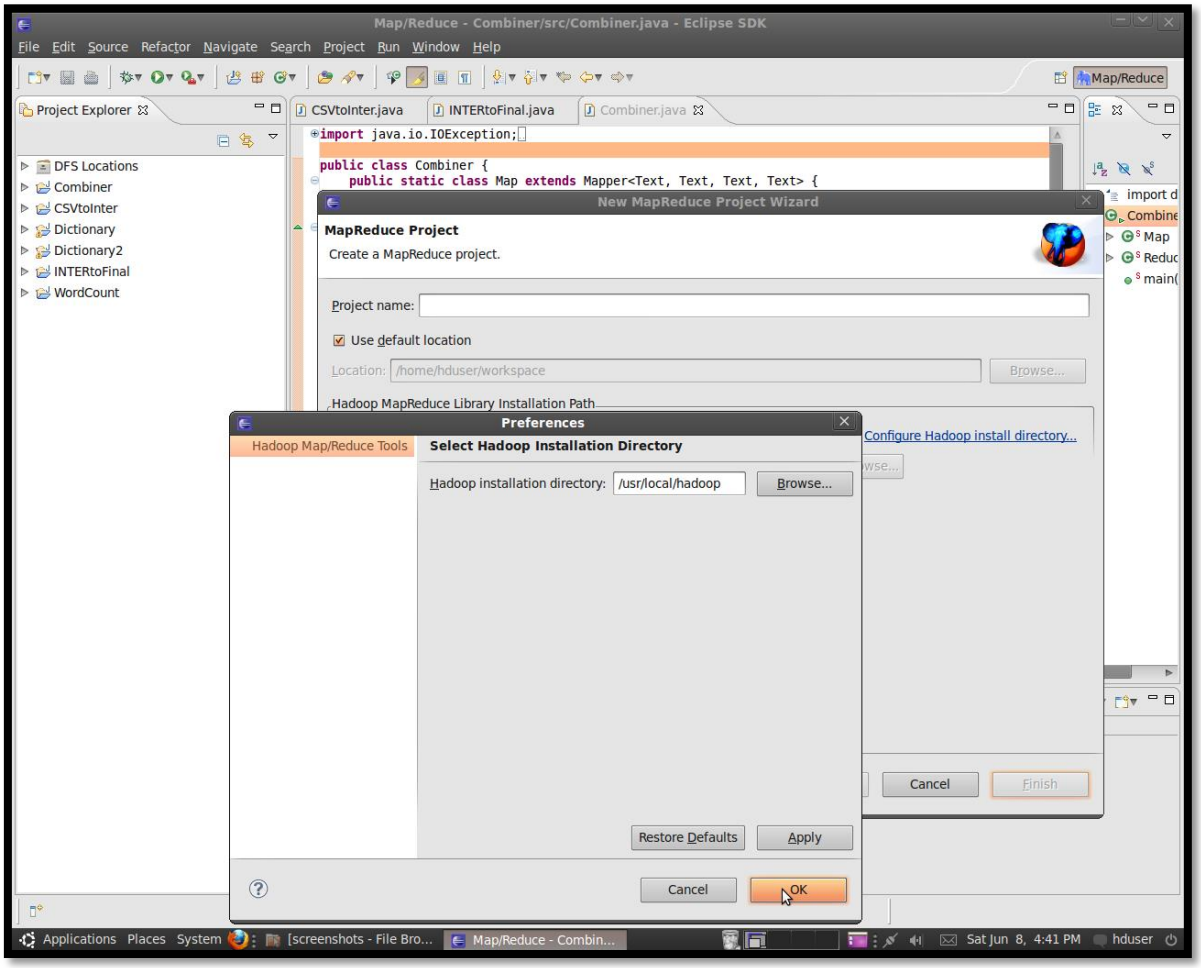
Εικόνα 1. Perspective MapReduce

Κάτω στην κονσόλα υπάρχει μία επιλογή που λέει new Hadoop Location. Το ανοίγουμε με δεξί κλικ για να θέσουμε κάποιες συγκεκριμένες ρυθμίσεις. Στο Location name βάζουμε ότι όνομα θέλουμε. Στο Host βάζουμε την IP του master υπολογιστή. Στο port βάζουμε 54310. Κάνουμε κλικ στην επιλογή που έχει Use M/R Master host και στο port βάζουμε 54311. Στο Username βάζουμε και πάλι ότι θέλουμε. Προτιμότερο είναι να χρησιμοποιήσουμε το όνομα χρήστη που δημιουργήσαμε σαν dedicated hadoop user.



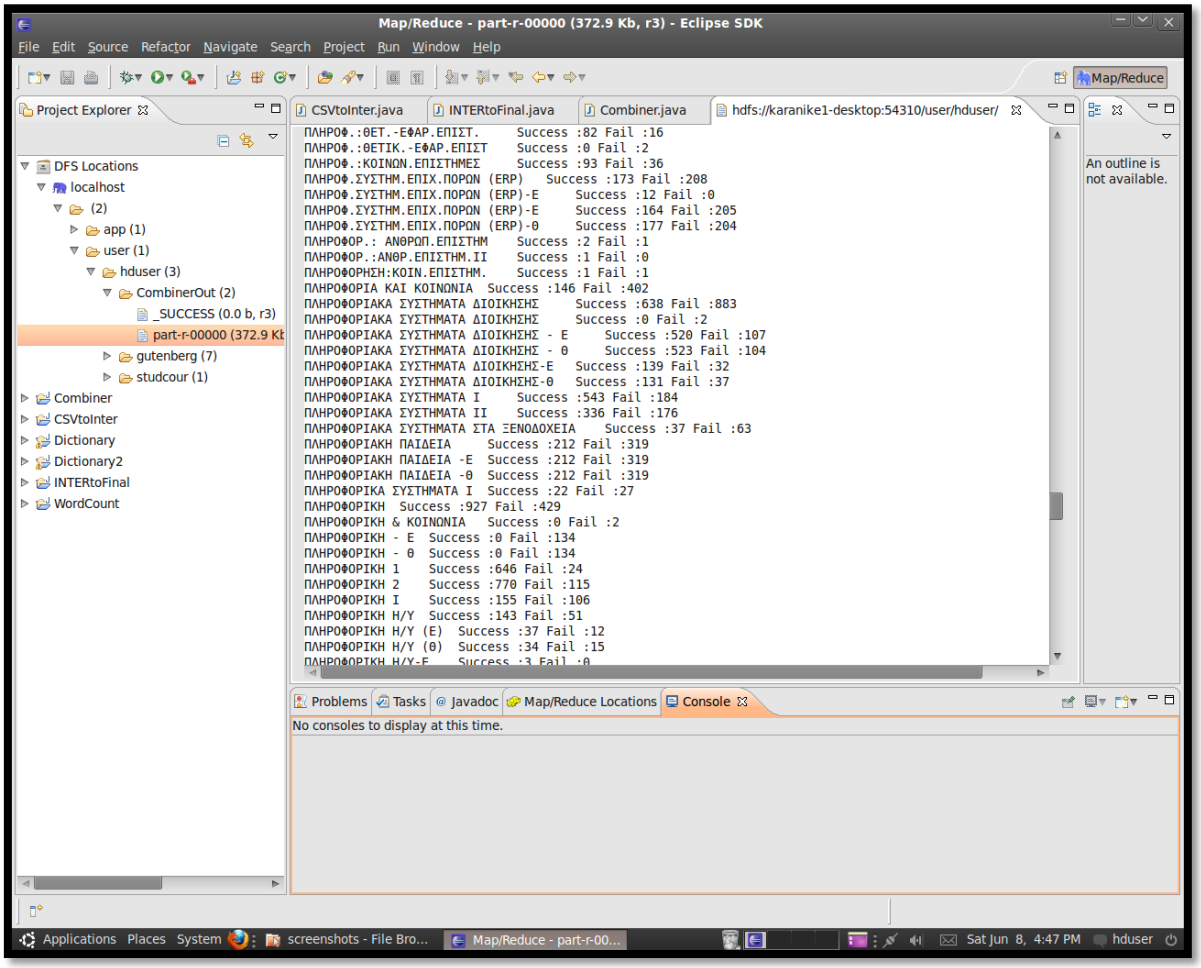
Εικόνα 2. New Hadoop Location

Στη συνέχεια πρέπει να δημιουργήσουμε ένα project το οποίο θα φιλοξενήσει τον κώδικα μας. Αφού επιλέξουμε από το μενού τη δημιουργία ενός νέου MapReduce project και του δώσουμε το επιθυμητό όνομα πρέπει να κάνουμε κλικ στην επιλογή Configure Hadoop Install directory και να δηλώσουμε στο Eclipse τον κατάλογο στον οποίο είναι εγκατεστημένο το Hadoop στον υπολογιστή μας.



Εικόνα 3. Configure Hadoop directory

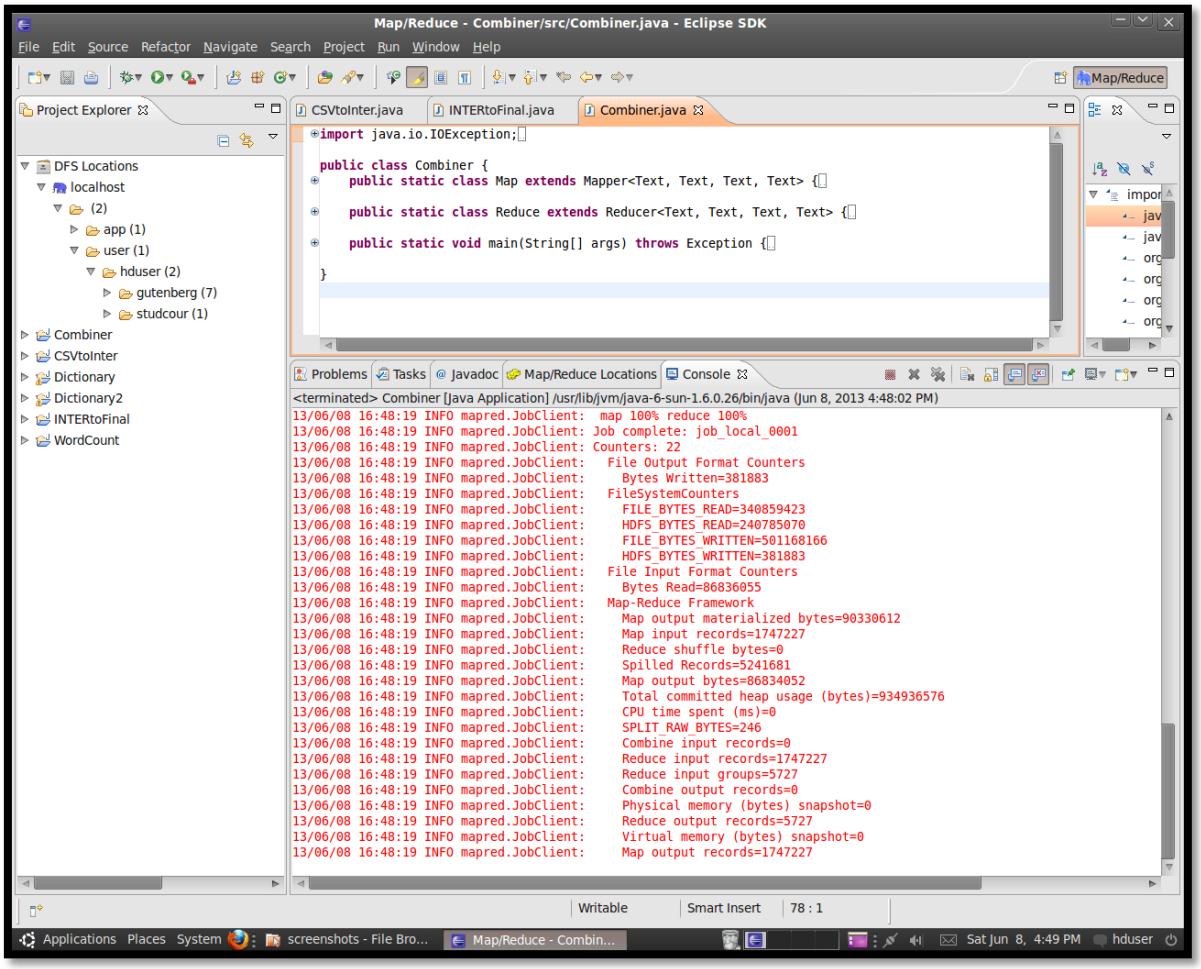
Τώρα για να τρέξουμε τον κώδικα που θέλουμε ανοίγουμε ένα αρχείο Java και γράφουμε τον κώδικά μας. Πρέπει να δηλώσουμε όμως στο πρόγραμμα έναν κατάλογο εισαγωγής δεδομένων (input-directory) και έναν κατάλογο που θα αποθηκευτούν τα δεδομένα που θα προκύψουν από την εκτέλεση του προγράμματος (output-directory). Ο κατάλογος από τον οποίο θα πάρει τα δεδομένα το πρόγραμμά μας θα πρέπει να δημιουργηθεί από εμάς κάνοντας δεξί κλικ αριστερά στον Project Explorer όπως φαίνεται παρακάτω.



Εικόνα 4. Input and Output directories

Αφού εκτελέσουμε τον κώδικα το πρόγραμμα θα δημιουργήσει αυτόματα τον κατάλογο με τα αποτελέσματα. Ο κατάλογος αυτός είναι απαραίτητο να μην υπάρχει αλλιώς θα δημιουργηθεί run-time error. Πρέπει δηλαδή να διαγράφουμε τον κατάλογο κάθε φορά που θέλουμε να εκτελέσουμε το πρόγραμμα.

Τώρα θα δούμε ένα παράδειγμα εκτέλεσης στο Hadoop και τα αποτελέσματα που θα μας εμφανίσει.



Εικόνα 5. Αποτελέσματα εκτέλεσης κώδικα

Εάν η εκτέλεση του προγράμματος τερματιστεί χωρίς λάθη θα εμφανιστεί στην οθόνη του Eclipse το παραπάνω κείμενο. Αυτό περιέχει πληροφορίες σχετικά με τις Mapping και Reducing διαδικασίες. Επίσης κατά την εκτέλεση του προγράμματος στην συγκεκριμένη κονσόλα μπορούμε να παρακολουθήσουμε την εξέλιξη των διεργασιών καθώς αυτές συμβαίνουν. Στο τέλος πάντοτε βρίσκεται η πληροφορία Map output records που μας δείχνει πόσες εγγραφές συνολικά παρήγαγε το πρόγραμμά μας.