ΑΝΩΤΑΤΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΜΕΣΟΛΟΓΓΙΟΥ

ΤΜΗΜΑ ΤΗΛΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ & ΔΙΚΤΥΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Κατασκευή εφαρμογών που χειρίζονται γρήγορα και παράλληλα τεράστιες ποσότητες δεδομένων σε μεγάλες ομάδες υπολογιστικών κόμβων με χρήση Hadoop Map/Reduce»

Πτυχιακή εργασία των

Δαβίδ Γκογκριτσιάνι
Α.Μ: 0673

Γιονίντ Αλίαϊ
Α.Μ: 0629

ΕΠΙΒΛΕΠΩΝ: ΧΡΗΣΤΟΣ ΤΡΑΝΩΡΗΣ, Ερευνητής στο Τμήμα Ηλεκτρολόγων Μηχανικών

ΝΑΥΠΑΚΤΟΣ, 2013

«Δηλώνουμε υπεύθυνα ότι το παρόν κείμενο αποτελεί προϊόν προσωπικής μας μελέτης και εργασίας και πως όλες οι πηγές που χρησιμοποιήθηκαν για τη συγγραφή της δηλώνονται σαφώς είτε στις παραπομπές είτε στη βιβλιογραφία. Γνωρίζουμε πως η λογοκλοπή αποτελεί σοβαρότατο παράπτωμα και είμαστε ενήμεροι για την επέλευση των νομίμων συνεπειών»

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

Ναύπακτος …/…../2013

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1.

2.

3.

Ευχαριστίες

Είναι μεγάλη χαρά για μας να αναγνωρίσουμε, στην προσπάθειά μας αυτή, την βοήθεια και τη συμβολή ενός πλήθους ανθρώπων. Πρώτα θέλουμε να ευχαριστήσουμε, τον επιβλέποντα καθηγητή μας κ. Χρήστο Τρανώρη για την υποστήριξη, την καθοδήγηση και τις συμβουλές που μας προσέφερε καθ' όλη τη διάρκεια αυτής της Πτυχιακής Εργασίας καθώς και για την επίτευξη μιας άψογης συνεργασίας.

Στη συνέχεια θα θέλαμε να ευχαριστήσουμε όλους τους καθηγητές μας που στάθηκαν αρωγοί στις προσπάθειές μας και μας μεταλαμπάδευσαν τις πολύτιμες γνώσεις τους.

Τις ιδιαίτερες ευχαριστίες μας θα θέλαμε να δώσουμε και στους φίλους μας για την κατανόηση και την ψυχολογική συμπαράσταση που μας προσέφεραν όλο αυτό το διάστημα.

Τέλος είμαστε ευγνώμονες στις οικογένειές μας, για την αμέριστη συμπαράστασή (οικονομική και ηθική), την ενθάρρυνση και την υπομονή τους σε όλα αυτά τα χρόνια των σπουδών μας.

Περίληψη

Ο στόχος της παρούσας πτυχιακής εργασίας είναι η παρουσίαση ενός framework συστήματος, όπως το Hadoop, καθώς και ενός προγραμματιστικού μοντέλου, όπως το MapReduce, μέσω του οποίου θα μπορούμε να επεξεργαζόμαστε παράλληλα και αποτελεσματικά ένα τεράστιο όγκο δεδομένων ο οποίος παράγεται καθημερινά στο διαδίκτυο (όπως π.χ. στα social media, σε τεράστια ιδιόκτητα δίκτυα), προσπαθώντας να αυξήσει την επεκτασιμότητα σε όλες τις διαδικασίες (του όγκου δεδομένων, αύξηση αριθμού συστημάτων μελών, αύξηση του αριθμού χρηστών). Η προσπάθεια θα επικεντρωθεί στην υλοποίηση του ανωτέρω συστήματος με τρόπο τέτοιο, ώστε η όλη διαδικασία να λαμβάνει χώρα τοπικά στον υπολογιστή/συσκευή του χρήστη. Η πλήρης ανάπτυξη του συστήματος θα βασίζεται στην τεχνολογία Hadoop MapReduce μέσω ενός δοκιμαστικού περιβάλλοντος που θα δημιουργηθεί με τη χρήση Εικονικών Μηχανών (Virtual Machines) σε συσκευή Φορητού Υπολογιστή (Laptop Computer/Notebook). Οι συσκευές συμμετέχουν σε ένα ομαδοποιημένο σύνολο (cluster) αποτελούμενο απο 4 εικονικές μηχανές κάθε μια εκ των οποίων θα επεξεργάζεται συγκεκριμένο όγκο δεδομένων, εκτελώντας συγκεκριμένες εργασίες. Το σύστημα θα είναι δυναμικό, γεγονός το οποίο σημαίνει ότι οι διασυνδεδεμένες στο σύστημα συσκευές θα εισέρχονται και θα αποχωρούν από το σύστημα, αναλόγως με τον όγκο των προς επεξεργασία δεδομένων. Ένας κατάλληλος αλγόριθμος θα επιλέγει τον κατάλληλο αριθμό συσκευών που απαιτούνται για το σύνολο της επεξεργασίας των δεδομένων, καθώς και για το δυναμικό καταμερισμό των πόρων του συστήματος, ώστε να επικρατεί δικαιοσύνη μέταξύ των χρηστών. Για την επεξεργασία και υλοποίηση των δεδομένων, θα χρησιμοποιηθεί το προγραμματιστικό μοντέλο MapReduce.

***Λέξεις Κλειδιά*:** Hadoop, MapReduce, Cluster, Virtual Machines.

Abstract

The objective of this thesis is the presentation of a system framework like Hadoop, as much as of a programming model such as MapReduce, which enables a user to process in parallel and effectively an enormous volume of data produced daily in the internet (as e.g. in social media, in huge private network etc.), trying trying to increase scalability in all processes (data volumes, increasing number of systems members, increasing number of users). Our effort concentrates on the implementation of the above system such that the whole process takes place localy on a user's machine. The full implementation of the our system is based on Hadoop MapReduce technology and on a test environment which will be created using Virtual Machines (*Virtual Machines*) on the notebook devices (Laptop Computer / Notebook), which are called nodes. The nodes are participating in a cluster consisting of four virtual machines, each of which will process a given amount of data, performing specific tasks. The system is dynamic, meaning that all nodes having access to the cloud are able to connect or disconnect based on the total volume of data to be processed. A specific algorithm will be employed in order to decide about the optimal number of nodes connected to the system/cluster, depending on the total amount of data, as long as for the dynamic allocation of the system's resources, maintaining the system's justice. As previously mentioned, the MapReduce programmable model will be employed for data processing.

***Key – Words*:** Hadoop, MapReduce, Cluster, Virtual Machines.

Σύντομη περιγραφή διπλωματικής

Στην παρούσα πτυχιακή εργασία παρουσιάζεται ενα open source framework του Apache Foundation όπως το Hadoop, καθώς και το μοντέλο προγραμματισμού όπως το MapReduce, μέσω του οποίου που θα μπορούμε να επεξεργαζόμαστε παράλληλα και αποτελεσματικά ένα τεράστιο όγκο δεδομένων, συνήθως σε κλίμακα πολλών terabytes, ο οποίος παράγεται καθημερινά στο διαδίκτυο (όπως π.χ. στα social media, σε τεράστια ιδιόκτητα δίκτυα). Η προσπάθεια θα επικεντρωθεί στην υλοποίηση του ανωτέρω συστήματος με τρόπο τέτοιο, ώστε η όλη διαδικασία να λαμβάνει χώρα τοπικά στον υπολογιστή/συσκευή του χρήστη, βάση της τεχνολογίας Hadoop MapReduce. Οι συσκευές συμμετέχουν σε ένα ομαδοποιημένο σύνολο (cluster) αποτελούμενο απο 4 εικονικές μηχανές κάθε μια εκ των οποίων επεξεργάζεται συγκεκριμένο όγκο δεδομένων, εκτελώντας συγκεκριμένες εργασίες. Ένας κατάλληλος αλγόριθμος θα επιλέγει τον κατάλληλο αριθμό συσκευών που απαιτούνται για το σύνολο της επεξεργασίας των δεδομένων. Για την υλοποίηση του προγράμματος και την επεξεργασία των δεδομένων, όπως αναφέρθηκε προηγουμένως, θα χρησιμοποιηθεί το προγραμματιστικό μοντέλο MapReduce και συγκεκριμένα η έκδοση 0.20.2 του Hadoop, μίας open-source υλοποίησης του MapReduce.

Στην παρούσα εργασία, παρουσιάζονται τα εξής: το Κεφάλαιο 1, όπου περιγράφει την έννοια του μεγάλου όγκου δεδομένων (Big Data), το Κεφάλαιο 2 που εστιάζει σε γενικές πληροφορίες του Hadoop. Έπειτα το 3^{ο} Κεφάλαιο περιγράφει ομαδοποιημένο σύνολο (cluster) του Hadoop, και το 4^{ο} ακολουθεί με το κατανεμημένο σύστημα αρχείων του Hadoop, ενώ το Κεφάλαιο 5 δίνει μια λεπτομερή επισκόπηση του προγραμματιστικού μοντέλου MapReduce. Στο Κεφάλαιο 6 περιγράφεται η εκτέλεση των πειραμάτων. Επιπλέον, στο Κεφάλαιο 7 απεικονίζονται τα web interfaces του Hadoop cluster. Η εργασία κλείνει με τον επίλογο και τα συμπεράσματα και προτείνονται ιδέες για μελλοντικές επεκτάσεις. Τέλος, γίνεται αναφορά στις οδηγίες εγκατάστασης του Hadoop, το οποίο παρατίθεται στο Παράρτημα Α'. Στο Παράρτημα Β' περιλαμβάνεται ο κώδικας που υλοποιήθηκε κάνοντας χρήση του MapReduce σε Java.

Πίνακας Περιεχομένων

Ι. Πίνακας πινάκων

ΙΙ. Πίνακας σχημάτων

ΚΕΦΑΛΑΙΟ 1ο

ΕΙΣΑΓΩΓΗ ΣΤΙΣ ΈΝΝΟΙΕΣ

1.1 Big Data

Τα Big Data είναι μια συλλογή πολλών και σύνθετων δεδομένων, τα οποία είναι δύσκολα να υποβληθούν σε επεξεργασία με τη χρήση κοινών εργαλείων διαχείρισης βάσης δεδομένων ή των παραδοσιακών εφαρμογών επεξεργασίας δεδομένων. Οι προκλήσεις που προκύπτουν αφορούν τα παρακάτω: συλλογή, διόρθωση, αποθήκευση, αναζήτηση, διανομή, μεταφορά, ανάλυση, και απεικόνιση των δεδομένων. Η τάση που παρατηρείται προς τα τεράστια σύνολα δεδομένων οφείλεται στις πρόσθετες πληροφορίες, οι οποίες εξάγονται από την ανάλυση ενός ενιαίου μεγάλου συνόλου συσχετιζόμενων δεδομένων, σε σύγκριση με τα χωριστά μικρότερα σύνολα, με το ίδιο συνολικό ποσό δεδομένων. Η χρήση μεγάλης ποσότητας δεδομένων επιτρέπει να γίνουν οι συσχετισμοί, προκειμένου να βρεθούν τα «σημεία επιχειρησιακών τάσεων, να καθορίσει την ποιότητα της έρευνας, να αποτρέψει τις ασθένειες, να συνδέσει τις νομικές παραπομπές, να καταπολεμήσει το έγκλημα, και να καθορίσει σε πραγματικό χρόνο τις συνθήκες κυκλοφορίας στους δρόμους.»

Τα δεδομένα είναι εθιστικά. Η επανάσταση στον τομέα των υπολογιστών και του Διαδικτύου έχει αυξήσει σε μεγάλο βαθμό την ικανότητα συλλογής και αποθήκευσης δεδομένων. Παλαιότερα, η βιβλιοθήκη του Κογκρέσου των ΗΠΑ, ήταν μία από τις μεγαλύτερες συλλογές δεδομένων στον κόσμο (περίπου 10 terabytes Πληροφορίας). Σήμερα οι μεγάλες εταιρείες του Διαδικτύου συλλέγουν μεγάλο μέρος δεδομένων σε καθημερινή βάση, και δεν είναι μόνο οι εφαρμογές του Internet, οι οποίες παράγουν δεδομένα σε τεράστια ποσά. Τα δεδομένα προέρχονται από παντού: από αισθητήρες που χρησιμοποιούνται για τη συλλογή κλιματικών δεδομένων, από δημοσιεύσεις στις ιστοσελίδες social media, από δεδομένα ψηφιακής φωτογραφίας και βίντεο, καθώς και από την καταγραφή οικονομικών συναλλαγών, από το σήμα GPS του κινητό τηλέφωνου. Όλα αυτά τα δεδομένα αποτελούν τα λεγόμενα «Big Data».

Μία εκτίμηση της IDC υπολόγισε το μέγεθος του "Digital Universe" στα 0,18 zettabytes το 2006, ενώ η πρόβλεψη έως το 2011 δείχνει μία δεκαπλάσια αύξηση έως τα 1,8 zettabytes.

1 Zettabytes = 10^{21} bytes = 1000 exabytes = 1 εκατομμύριο petabytes (PB) = 1 δισ. terabytes (TB)

Τεράστια ποσότητα, έτσι δεν είναι;

Ορισμένες πηγές που προκαλούν αυτή τη μαζική αύξηση στην ποσότητα των δεδομένων είναι:

- Το χρηματιστήριο της Νέας Υόρκης, όπου παράγει περίπου 1 TB νέων δεδομένων συναλλαγών ανά ημέρα.
- Το Facebook, όπου φιλοξενεί περίπου 10 δισεκατομμύρια φωτογραφίες, και καταλαμβάνει πάνω από 1 PB αποθηκευτικού χώρου.
- Τα online καταστήματα, όπου αποθηκεύουν περίπου 2 PB δεδομένων. Η ποσότητα αυτή αυξάνεται με ρυθμό 20 TB ανά μήνα.

Τα Big Data σχετίζονται με κάτι περισσότερο από το μέγεθος. Αποτελούν μια ευκαιρία για να βρεθούν ιδέες σε νέους και αναδυόμενους τύπους των δεδομένων και των περιεχομένων τους, για να γίνει μια επιχείρησή πιο ευέλικτη, και να απαντηθούν ερωτήσεις που προηγουμένως θεωρούνταν ακατόρθωτες. Μέχρι τώρα, δεν υπήρχε πρακτικός τρόπος για κάτι τέτοιο. Το καλό νέο είναι ότι με τη βοήθεια εργαλείων όπως τα Hadoop και MapReduce, ανοίγουν πόρτες σε έναν κόσμο δυνατοτήτων**.** [1]

ΑΝΑΦΟΡΕΣ

[1]. Big data - http://en.wikipedia.org/wiki/Big_data

ΚΕΦΑΛΑΙΟ 2ο

ΗADOOP

2.1 **Η**adoop

Το Hadoop ξεκίνησε ως ένα υποπρόγραμμα (*subprojects*) Nutch, το οποίο με τη σειρά του ήταν ένα υποπρόγραμμα του Apache Lucene. Ο Doug Cutting δημιούργησε και τα τρία προγράμματα, καθένα εκ των οποίων αποτέλεσε μία λογική εξέλιξη των προηγούμενων.

Το Lucene είναι μια βιβλιοθήκη για ευρετηριακή ταξινόμηση κείμενου και αναζήτηση. Λαμβάνοντας υπ' όψιν μία συλλογή κειμένων, ένας προγραμματιστής μπορεί εύκολα να προσθέσει την ικανότητα αναζήτησης στα έγγραφα χρησιμοποιώντας τη μηχανή Lucene. Το Desktop search, enterprise search, και πολλές *domain-specific* μηχανές αναζήτησης έχουν κατασκευαστεί χρησιμοποιώντας το Lucene. Το Nutch είναι η πιο φιλόδοξη επέκταση του Lucene, όπου προσπαθεί να δημιουργήσει μια πλήρης μηχανή αναζήτησης στον Ιστό χρησιμοποιώντας το Lucene ως βασικό του στοιχείο.

Το Nutch έχει τους συντακτικούς αναλυτές για το HTML, έναν crawler ιστού, μια βάση δεδομένων συνδεδεμένου γράφου, και άλλα πρόσθετα στοιχεία απαραίτητα για μια μηχανή αναζήτησης Ιστού. Ο Doug Cutting οραματίστηκε το Nutch ως μια εναλλακτική λύση ανοικτού κώδικα, απέναντι στις εμπορικές τεχνολογίες, όπως επί παραδείγματι της Google.

Εκτός από την προσθήκη των εξαρτημάτων (*components*), όπως ένα crawler, και έναν parser, μια μηχανή αναζήτησης Ιστού διαφέρει από μια βασική μηχανή αναζήτησης εγγράφων ως προς την κλιμάκωση. Εκτιμώντας ότι το Lucene στοχεύει στην ευρετηριακή ταξινόμηση (*indexing*) εκατομμυρίων εγγράφων, το Nutch πρέπει να είναι σε θέση να χειριστεί δισεκατομμύρια ιστοσελίδες, χωρίς να γίνει εξωφρενικά κοστοβόρο για να λειτουργήσει. Το Nutch θα πρέπει να τρέξει σε ένα κατανεμημένο cluster από υπολογιστές γενικής χρήσης (*commodity hardware*). Η πρόκληση για την ομάδα Nutch είναι να αντιμετωπιστούν τα ζητήματα επεκτασιμότητας σε θέματα λογισμικού. Το Nutch επίσης χρειάζεται ένα επίπεδο για να χειριστεί την κατανεμημένη επεξεργασία, τον πλεονασμό, την αυτόματη επανάκαμψη αποτυχίας, και την εξισορρόπηση φορτίου (*load balancing*), όπου αυτές οι προκλήσεις δεν είναι καθόλου αναξιόλογες **(***trivial***).**

Κατά το **2004**, η Google δημοσίευσε δύο έγγραφα περιγράφοντας το σύστημα αρχείων Google (GFS) και το *MapReduce framework*. Η Google υποστήριξε ότι

χρησιμοποιούν αυτές τις δύο τεχνολογίες για την αλλαγή κλίμακας στο δικό τους σύστημα αναζήτησης. Ο Doug Cutting, αμέσως αντιλήφθηκε την εφαρμοστικότητα αυτών των τεχνολογιών σε Nutch, και έτσι η ομάδα του υλοποίησε το νέο framework και μετάφερε το Nutch σε αυτό. Η νέα υλοποίηση ενίσχυσε αμέσως την επεκτασιμότητα του Nutch. Άρχισε να χειρίζεται αρκετές εκατοντάδες εκατομμύρια ιστοσελίδες και επιπρόσθετα θα μπορούσε να τρέξει σε clusters δεκάδων κόμβων.

Ο Doug συνειδητοποίησε ότι ήταν απαραίτητο ένα αφιερωμένο *project* για να διανθίσει τις δύο τεχνολογίες, ώστε να έχουμε κλιμάκωση στον Ιστό, και έτσι το Hadoop γεννήθηκε. Η Yahoo! προσέλαβε τον Doug τον Ιανουάριο του **2006,** ώστε να συνεργαστεί με μια ειδική ομάδα για την ανάπτυξη του Hadoop ως ***open source project.*** Δύο χρόνια αργότερα, το Hadoop κατόρθωσε την εγκατάσταση ενός κορυφαίου Apache Project. Αργότερα, στις 19 Φεβρουαρίου **2008,** η Yahoo! ανακοίνωσε ότι το Hadoop που έτρεχε σε 10.000+ cluster με πυρήνα Linux ήταν το σύστημα παραγωγής της για την ευρετηριακή ταξινόμηση Ιστού. Το Hadoop είχε χτυπήσει αληθινά την web κλιμάκωση!

2.2 Η Προέλευση του Ονόματος "Hadoop"

Κατά την ονοματοδοσία των projects λογισμικού, ο Doug Cutting φαίνεται να εμπνέεται από την οικογένειά του. Το **Lucene** είναι το μεσαίο όνομα της γυναίκας του, καθώς και το όνομα της μητέρας της γιαγιάς της. Ο γιος του επίσης, όταν άρχισε να περπατά χρησιμοποιούσε τη λέξη **Nutch** σαν μια λέξη παντός σκοπού για το γεύμα, και επιπρόσθετα φώναζε το κίτρινο ελεφαντάκι του με το «όνομα» Hadoop. Ο Doug είπε ότι "έψαχνα για ένα όνομα που δεν ήταν ήδη ένα domain web αλλά και ούτε εμπορικό σήμα, και έτσι δοκίμασα διάφορες λέξεις που υπήρχαν στη ζωή μου, αλλά δεν χρησιμοποιούνταν από κανέναν άλλον. Τα παιδιά είναι πολύ καλά στο να επινοούν λέξεις ." [1]

2.3 Τι Είναι Το Hadoop;

Τυπικά μιλώντας, το Hadoop είναι ένα ανοικτού κώδικα framework για τη συγγραφή και εκτέλεση κατανεμημένων εφαρμογών που επεξεργάζονται παράλληλα μεγάλες ποσότητες δεδομένων εισόδου, επιταχύνοντας έτσι τις διαδικασίες. Ο κατανεμημένος υπολογισμός είναι ένα ευρύ και ποικίλο πεδίο, αλλά οι βασικές διακρίσεις το Hadoop είναι ότι είναι:

Προσπελάσιμο: Το Hadoop εκτελείται σε μεγάλα clusters εμπορικών μηχανών ή στις υπηρεσίες cloud computing όπως το Elastic Compute Cloud (EC2) της Amazon.

Ανθεκτικό: Επειδή προορίζεται να τρέξει σε εμπορικό υλικό, έχει δομηθεί με την υπόθεση των συχνών δυσλειτουργιών υλικού. Έτσι μπορεί να διαχειριστεί τις περισσότερες από αυτές τις αποτυχίες.

Κλιμακούμενο: Κλιμακώνεται γραμμικά, για να χειριστεί τα περισσότερα δεδομένα με την προσθήκη περισσότερων κόμβων στο cluster**.**

Αξιόπιστο: Σε περίπτωση βλάβης γίνεται ανάθεση των εργασιών σε νέους κόμβους. Επίσης παρέχει τη δυνατότητα αυτόματης αποθήκευσης των δεδομένων σε πολλαπλά αντίγραφα.

Απλό: Το Hadoop επιτρέπει στους χρήστες να γράφουν γρήγορα έναν αποδοτικό, παράλληλο κώδικα**.**

Οικονομικό: Τα δεδομένα κατανέμονται σε clusters αποτελούμενα απο υπολογιστές γενικής χρήσης.

Η δυνατότητα προσπέλασης και η απλότητα του Hadoop του δίνουν επιπλέον πλεονεκτήματα εκτός από το γράψιμο και την εκτέλεση μεγάλων κατανεμημένων προγραμμάτων. Ακόμη και οι φοιτητές πανεπιστημίων μπορούν γρήγορα και οικονομικά να δημιουργήσουν το δικό τους Hadoop cluster. Αφετέρου, η σθεναρότητα-ανθεκτικότητα και η κλιμάκωσή του, το καθιστούν κατάλληλο για ακόμη και τις πιο απαιτητικές εργασίες της Yahoo και του Facebook. Αυτά τα χαρακτηριστικά γνωρίσματα λοιπόν, καθιστούν το Hadoop δημοφιλές τόσο στον ακαδημαϊκό χώρο όσο και στη βιομηχανία. [2]

2.4 Γιατί Hadoop ;

Γιατί το Hadoop είναι η τελευταία buzz-word (ευρύτατα χρησιμοποιούμενη λέξη ειδικής ή τεχνικής ορολογίας, "καραμέλα") **;** Στον «Ευρωπαϊκό Οργανισμό Πυρηνικών Ερευνών» (CERN) οι επιστήμονες αξιολογούν το HDFS ως ένα σύστημα αρχείων για την αποθήκευση τεράστιων δεδομένων (περίπου 40TB / ημέρα (15PB/έτος)) που παράγονται από Large Hadron Collider (*LHC*) για να αποθηκεύονται και να επεξεργάζονται το HDFS.

Στην Ασία, η China Mobile, η μεγαλύτερη εταιρεία τηλεπικοινωνιών παγκοσμίως χρησιμοποιεί Hadoop για την επεξεργασία ενός τεράστιου όγκου δεδομένων. Και στη Silicon Valley (*περιοχή υψηλής τεχνολογίας, όπου δραστηριοποιούνται στις τεχνολογίες της Πληροφορικής και των Επικοινωνιών*) κάθε εταιρεία *Web* χρησιμοποιεί ήδη Hadoop (Yahoo!, Facebook, LinkedIn, Netflix, IBM, Zynga, Amazon).

Η Microsoft ανακοίνωσε ότι υποστηρίζει Hadoop σε Azure {*πλατφόρμα Υπολογιστικού Νέφους με MS* (*Microsoft Cloud Computing*)}, η οποία ανταγωνίζεται με τα Amazon EC2 και S3. Το επιχείρημα ότι το Hadoop κέρδισε αυτήν τη κεκτημένη ταχύτητα επειδή είναι ένα open source project δεν ευσταθεί, και τα εύσημα πηγαίνουν στη Yahoo**!,** αν και το Hadoop ήταν πνευματικό δημιούργημα του Dough Cutting (πρώην υπάλληλος της Yahoo!!). Η Yahoo! δαπάνησε τεράστιους πόρους κατά την ανάπτυξη του σημερινού Hadoop, όπως επίσης συμβάλλει σε περισσότερο από το **80%** του κώδικα Hadoop. [3]

2.5 **A**pache **H**adoop **P**roject

Το Hadoop project περιλαμβάνει τα ακόλουθα subprojects, τα οποία σχηματικά φαίνονται στο Σχήμα 2.1:

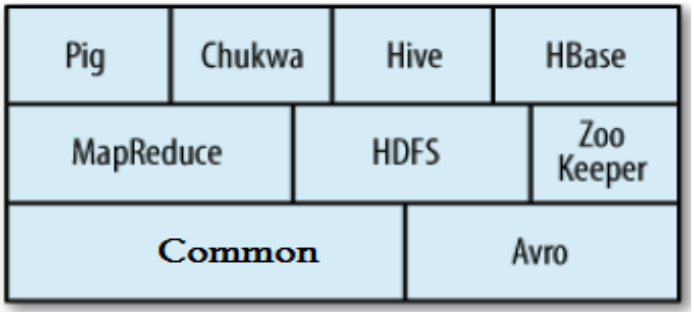
- **Hadoop Common:** Κοινά βοηθητικά προγράμματα (*common utilities*) για υποστήριξη άλλων Hadoop subprojects. Περιλαμβάνει ένα σύνολο από εξαρτήματα (components) και interfaces για κατανεμημένα συστήματα αρχείων και υπηρεσίες I/O (serialization**,** Java RPC (*Κλήσεις Απομακρυσμένων Διαδικασιών*)**,** ανθεκτικές δομές δεδομένων).
- **Hadoop Distributed File System (HDFS):** Κατανεμημένο σύστημα αρχείων που παρέχει υψηλό throughput στην πρόσβαση στα δεδομένα της εφαρμογής.
- **Hadoop YARN (MapReduce 2.0):** Ένα framework για χρονοπρογραμματισμό/παρακολούθηση των εργασιών και διαχείριση των πόρων του cluster.
- **Hadoop MapReduce**: Είναι ένα σύστημα βασισμένο στο Hadoop Yarn για παράλληλη επεξεργασία των μεγάλων συνόλων δεδομένων σε υπολογιστικά clusters.

Άλλα Hadoop **-** projects σχετικά με το Apache είναι:

- **Ambari:** Ένα Web-based εργαλείο για την επίβλεψη και διαχείριση των clusters το οποίο περιλαμβάνει υποστήριξη για τα Hadoop HDFS, MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig και Sqoop.
- **Avro:** Ένα σύστημα serialization (αλληλουχίας-ανάκτησης) δεδομένων το οποίο παρέχει δυναμική ολοκλήρωση με γλώσσες σεναρίων (*scripting languages*).
- **Cassandra:** Μια κλιμάκωση πολλαπλών κυρίων σταθμών βάσης δεδομένων χωρίς κανένα μοναδικό σημείο αστοχίας.

- **Chukwa:** Ένα σύστημα συλλογής-ανάλυσης δεδομένων για τη διαχείριση μεγάλων κατανεμημένων συστημάτων. Τρέχει συλλέκτες που αποθηκεύουν τα δεδομένα σε HDFS και χρησιμοποιεί MapReduce για την παραγωγή αναφορών.
- **HBase:** Μια κλιμακούμενη, κατανεμημένη βάση δεδομένων που υποστηρίζει δομημένη αποθήκευση δεδομένων για μεγάλους πίνακες.
- **Hive:** Μια υποδομή αποθήκης δεδομένων (*data warehouse*) που παρέχει συνοπτικά δεδομένα (*summarization*) και σύνθετες αναζητήσεις (*ad hoc querying*) των δεδομένων. Διαχειρίζεται τα δεδομένα που είναι αποθηκευμένα στην HDFS και παρέχει μια γλώσσα ερωτημάτων που βασίζεται σε SQL.
- **Pig:** Μια υψηλού επιπέδου γλώσσα ροής δεδομένων και πλαίσιο εκτέλεσης για παράλληλους υπολογισμούς. Τρέχει σε HDFS και σε MapReduce clusters.
- **Mahout:** Παρέχει βιβλιοθήκες για εξόρυξη δεδομένων και για machine learning
- **ZooKeeper :** Μια κατανεμημένη**,** υψηλής διαθεσιμότητας (*highly available*) υπηρεσία συντονισμού (*coordination service*) για κατανεμημένες εφαρμογές**.** [4]

Η ανάλυση περισσότερων από αυτών των subprojects ξεφεύγει από τα όρια της πτυχιακής, άρα περισσότερες λεπτομέρειες θα πρέπει να αναζητηθούν σε σχετική βιβλιογραφία.



Σχήμα 2.1 *Hadoop subprojects*

2.6 Καταστάσεις Ενός Hadoop Cluster

- Standalone (Local) Mode
- Pseudo-Distributed Mode
- Fully-Distributed (Cluster) Mode

Standalone Operation

Σε αυτή την κατάσταση, όλοι οι daemons (*διεργασίες*) για map-reduce τρέχουν σε ένα single JVM (**Java Virtual Machine**). Αυτό είναι χρήσιμο για την ανάπτυξη και τον εντοπισμό σφαλμάτων (*debugging*).

Επειδή δεν υπάρχει καμία ανάγκη να επικοινωνήσει με άλλους κόμβους, η αυτόνομη λειτουργία δεν χρησιμοποιεί HDFS, ούτε θα ξεκινήσει οποιαδήποτε από τις Hadoop daemons.

Pseudo-Distributed Operation

Σε αυτή την κατάσταση, daemons τρέχουν πάνω σε διαφορετικούς JVMs σε ένα τοπικό μηχάνημα, σαν ένα "ψευδό" cluster.

Αυτή η κατάσταση συμπληρώνει την αυτόνομη λειτουργία για τον εντοπισμό σφαλμάτων στον κώδικα, επιτρέποντας στο χρήστη να εξετάσει τη χρήση της μνήμης, ζητήματα σχετικά με HDFS input/output, και άλλες daemon αλληλεπιδράσεις.

Fully-Distributed Operation

Τονίζονται τα οφέλη της κατανεμημένης αποθήκευσης και του κατανεμημένου υπολογισμού. Σε αυτή την κατάσταση, daemons τρέχουν πάνω σε διαφορετικούς JVMs και σε διαφορετικές μηχανές. Παρακάτω θα χρησιμοποιούνται τα ακόλουθα ονόματα κεντρικών υπολογιστών**:**

- Master - κύριος κόμβος του cluster και host του NameNode και του JobTracker daemons.
- Secondary NameNode daemon - για τη δημιουργία αντιγράφων ασφαλείας (*backup*) .
- slave1, slave2, slave3**,...**—Τα slave boxes του cluster που τρέχουν: DataNode και TaskTracker daemons.

ΑΝΑΦΟΡΕΣ

[1]. Chuck Lam, (2010): *Hadoop in Action*, Manning Publications, (σελ.19-20)

[2]. Chuck Lam, (2010): *Hadoop in Action*, Manning Publications, (σελ.4-5)

[3]. Why Hadoop; - http://bigdata.wordpress.com/2010/05/16/why-hadoop/

[4]. Hadoop Page - http://hadoop.apache.org/

[5]. Tom White, (2010): *Hadoop: The Definitive Guide*, 2nd Edition. O'Reilly Media/Yahoo Press, (σελ.12-13)

[6]. Chuck Lam, (2010): *Hadoop in Action*, Manning Publications, (σελ.29-31),

ΚΕΦΑΛΑΙΟ 3ο
HADOOP CLUSTER

3.1 Hadoop Cluster;

Ένα **Hadoop cluster,** είναι ένα cluster υπολογιστών που χρησιμοποιούνται για την εκτέλεση MapReduce εργασιών. Όλοι οι υπολογιστές που συμμετέχουν στο συγκεκριμένο cluster είναι προφανώς συνδεδεμένοι μεταξύ τους μέσω δικτύου, και έχουν όλοι τους εγκατεστημένη μια έκδοση του Hadoop.

3.2 HDFS LAYER

3.2.1 NameNode

Η αρχιτεκτονική του HDFS είναι της μορφής master/slave. Το ρόλο του master έχει ο NameNode, ο οποίος διαχειρίζεται το namespace του συστήματος αρχείων και ρυθμίζει την πρόσβαση των clients στα αρχεία του HDFS. Επιπρόσθετα, ο NameNode έχει τις πληροφορίες για την αντιστοίχηση (*mapping*) των blocks στους DataNodes, και για όλα τα metadata του HDFS. Οποιαδήποτε αλλαγή στο namespace ή τις ιδιότητες του συστήματος αρχείων καταγράφεται από τον NameNode.

Πιο αναλυτικά, ο NameNode χρησιμοποιεί ένα αρχείο στο τοπικό σύστημα αρχείων, ώστε να αποθηκεύει το *EditLog*, ένα transaction log με ξεχωριστή εγγραφή για κάθε αλλαγή που συμβαίνει στο HDFS. Ενδεικτικά, τέτοιες αλλαγές μπορεί να είναι η δημιουργία ενός αρχείου/directory, το άνοιγμα, το κλείσιμο ή η αλλαγή του replication factor του αρχείου. Εκτός από το EditLog, υπάρχει και ένα άλλο αρχείο που ονομάζεται *FsImage* και αποθηκεύεται και αυτό στο τοπικό σύστημα αρχείων του NameNode. Στο αρχείο αυτό, αποθηκεύεται ολόκληρο το namespace του συστήματος αρχείων, μαζί με τις ιδιότητες του τελευταίου και τις αντιστοιχίσεις των blocks σε αρχεία.

Ο NameNode κρατάει μία εικόνα του namespace του συστήματος αρχείων και το αρχείο BlockMap στη μνήμη. Ο σχεδιασμός έχει γίνει με τρόπο τέτοιο, ώστε 4GB RAM να επιτρέπουν στον NameNode να διαχειρίζεται ένα πολύ μεγάλο αριθμό αρχείων και φακέλων. Όταν ξεκινάει ο NameNode, διαβάζει το FsImage και το EditLog από το δίσκο, και

εφαρμόζει όλες τις συναλλαγές του EditLog στο FsImage που βρίσκεται στη μνήμη. Αφού αποθηκευτεί το ενημερωμένο FsImage στο δίσκο, ο NameNode "κόβει" το EditLog. Η διαδικασία αυτή λοιπόν, ονομάζεται checkpoint και υλοποιείται, προς το παρόν, μόνο κατά την εκκίνηση του NameNode.

3.2.2 DataNode

Οι DataNodes αποτελούν τους slaves ή workers του HDFS, και συνήθως τρέχει ένας σε κάθε μηχάνημα εκτός από αυτό που τρέχει τον NameNode. Οι DataNodes ικανοποιούν τις αιτήσεις των clients του συστήματος για ανάγνωση και εγγραφή και επιπλέον δημιουργούν, και διαγράφουν blocks, όπως και φροντίζουν για τα αντίγραφά τους, με βάση τις οδηγίες του NameNode. Επίσης, ενημερώνουν τον τελευταίο για τα blocks που έχει ο καθένας, ενώ του στέλνουν και ανά τακτά διαστήματα "παλμό" που δηλώνει ότι λειτουργούν κανονικά.

Κάθε DataNode αποθηκεύει HDFS δεδομένα σε αρχεία (1 block ανά αρχείο) στο τοπικό σύστημα αρχείων, ενώ δε γνωρίζει τίποτα για τα HDFS αρχεία. Η αποθήκευση των αρχείων στο τοπικό σύστημα γίνεται σε διαφορετικούς φακέλους, ο αριθμός των οποίων καθορίζεται από ευριστική συνάρτηση που δίνει το βέλτιστο αριθμό αρχείων ανά φάκελο. Αυτό συμβαίνει επειδή το τοπικό σύστημα αρχείων ενδέχεται να μην μπορεί να διαχειριστεί αποδοτικά πολύ μεγάλο αριθμό αρχείων ανά φάκελο.

Κατά την εκκίνηση οι DataNodes, σκανάρουν το τοπικό σύστημα αρχείων και δημιουργούν μία λίστα με όλα τα HDFS block δεδομένων που αντιστοιχούν στα τοπικά αρχεία. Η λίστα αυτή αποτελεί το Blockreport και την αποστέλλουν στο NameNode. Κατά το διάστημα αυτό που οι DataNodes δημιουργούν τις αναφορές και τις στέλνουν στον NameNode, ο τελευταίος βρίσκεται σε safemode.

3.2.3 Secondary NameNode

Ο NameNode αποθηκεύει τις αλλαγές του συστήματος αρχείων σε ένα log αρχείο στο τοπικό σύστημα. Όταν ξεκινάει, διαβάζει την κατάσταση του HDFS από το FsImage και εφαρμόζει σε αυτό τις εγγραφές του EditLog. Αφού αποθηκευτεί το ανανεωμένο FsImage, και αρχίζει η κανονική λειτουργία του NameNode με άδειο το EditLog. Επειδή, όμως, η διαδικασία αυτή γίνεται μόνο κατά την εκκίνηση του NameNode, μπορεί το EditLog να γίνει πολύ μεγάλο σε μέγεθος με αποτέλεσμα να καθυστερεί η επανεκκίνηση του κόμβου. Για το λόγο αυτό υπάρχει ο Secondary NameNode, ο οποίος συνενώνει το FsImage με το EditLog

ανά τακτά χρονικά διαστήματα (η διαδικασία αυτή ονομάζεται checkpoint) και κρατάει το μέγεθος του EditLog εντός κάποιου ορίου. Οι απαιτήσεις σε μνήμη του Secondary NameNode είναι το ίδιο υψηλές με αυτές του NameNode κι έτσι, συνήθως, τρέχουν σε διαφορετικά μηχανήματα.

Η διαδικασία checkpoint στον Secondary NameNode καθορίζεται από τις εξής παραμέτρους:

- **fs.checkpoint.period**, η οποία καθορίζει το μέγιστο χρόνο που μεσολαβεί μεταξύ 2 checkpoints και
- **fs.checkpoint.size**, η οποία αποτελεί το μέγεθος του EditLog, που αν ξεπεραστεί πρέπει να γίνει άμεσα checkpoint.

Το directory στο οποίο αποθηκεύεται το τελευταίο checkpoint στον Secondary NameNode έχει την ίδια δομή με αυτό του Name Node, έτσι ώστε να μπορεί άμεσα να χρησιμοποιηθεί σε περίπτωση που αυτός αποτύχει.

3.3 MAP/REDUCE LAYER

3.3.1 Jobtracker

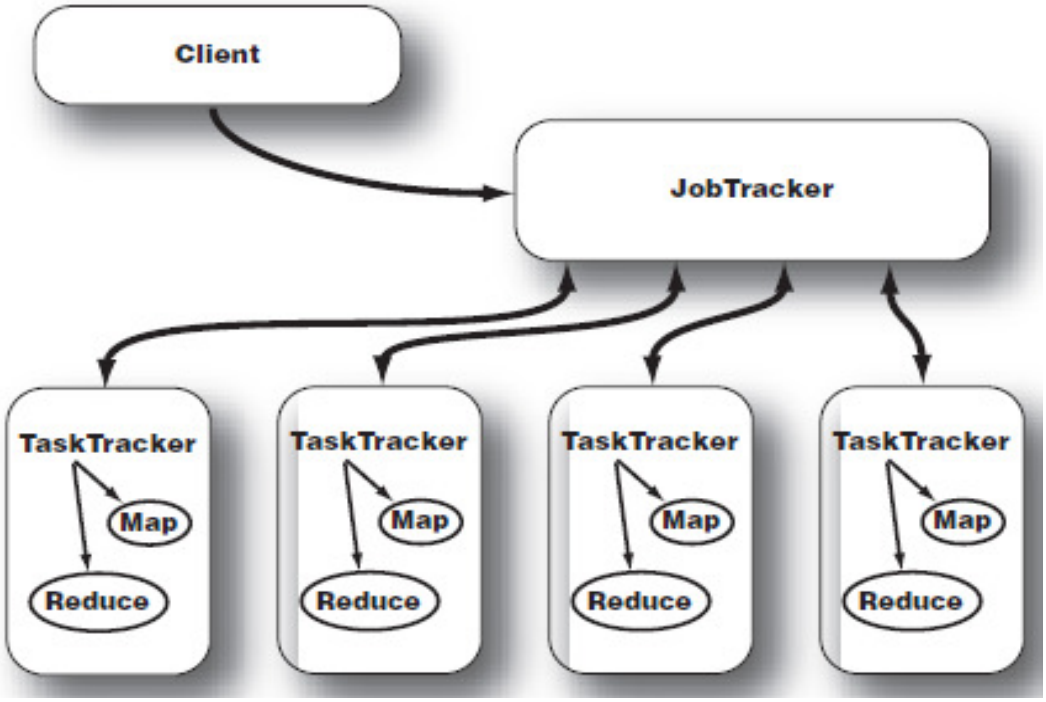
Ο JobTracker οργανώνει την εκτέλεση των MapReduce εργασιών (jobs) που υποβάλλουν οι clients. Συγκεκριμένα, τις χωρίζει σε μικρότερα tasks και προσπαθεί να τα αναθέσει σε TaskTrackers με τρόπο τέτοιο ώστε να ελαχιστοποιείται η κίνηση στο δίκτυο. Προκειμένου να κάνει αυτή την ανάθεση, ο JobTracker επικοινωνεί με τον NameNode για να καθορίσει πού βρίσκονται τα δεδομένα, και να αναθέσει τα tasks σε κόμβους που έχουν αυτά τα δεδομένα ή είναι κοντά σε κόμβους που τα έχουν, δηλαδή σε κόμβους στο ίδιο rack. Ο JobTracker δέχεται παλμούς από τους TaskTrackers και αυτοί τον ενημερώνουν αν κάποιο task απέτυχε. Αν σταματήσει κάποιο TaskTracker να δίνει παλμό, ο JobTracker θεωρεί ότι απέτυχε και αναθέτει το task σε άλλον TaskTracker. Αν αναφερθεί στο JobTracker ότι κάποιο task απέτυχε, τότε μπορεί να το αναθέσει εκ νέου σε άλλο κόμβο ή να σημειώσει ότι η συγκεκριμένη εγγραφή πρέπει να αποφεύγεται ή να βάλει σε blacklist τον TaskTracker που απέτυχε. Όταν τελειώσει η εργασία, ο JobTracker ανανεώνει την κατάστασή του.

Τα Apache Hadoop 0.20 και 0.21, δεν έχουν καμία υψηλή διαθεσιμότητα για τον JobTracker. Εάν ο JobTracker αποτύχει, όλες οι τρέχουσες εργασίες (*jobs*) τερματίζονται και η πρόοδός τους χάνεται.

3.3.2 TaskTracker

Οι TaskTrackers είναι εφαρμογές που εκτελούν τα tasks στα οποία έχει χωριστεί μία εργασία (job). Τα tasks ανατίθενται στους TaskTrackers από τον JobTracker, στον οποίο:

- Δίνουν αναφορά για την επιτυχία ή αποτυχία εκτέλεσης του task και
- Αποστέλλουν παλμό που δηλώνει την ορθή λειτουργία τους. Ο παλμός αυτός αποτελεί τον τρόπο επικοινωνίας μεταξύ των TaskTrackers και του JobTracker: μέσω αυτού γίνεται η ενημέρωση του τελευταίου για την κατάσταση του TaskTracker, καθώς και η ανάθεση νέου task. [1]



Σχήμα 3.1 *Το παραπάνω σχήμα απεικονίζει την αλληλεπίδραση του JobTracker με τον TaskTracker. Αφότου καλεί ένας client (εφαρμογή πελάτη) το JobTracker για να υποβάλει μια MapReduce εργασία (job) επεξεργασίας δεδομένων, ο JobTracker χωρίζει την εργασία και αναθέτει διαφορετικά map και reduce tasks σε κάθε TaskTracker στο cluster. (**Πηγή:** Chuck Lam, (2010): Hadoop in Action, Manning Publications)*

ΑΝΑΦΟΡΕΣ

[1]. http://casablanca.dblab.ece.ntua.gr/cloudwiki/index.php/Main_Page

ΚΕΦΑΛΑΙΟ 4ο

HADOOP DISTRIBUTED FILE SYSTEM

4.1 Τι είναι το HDFS;

Το **HDFS** είναι ένα κατανεμημένο σύστημα διαχείρισης αρχείων (*distributed computing*) για την αποθήκευση μεγάλων αρχείων, ιδανικά με μέγεθος πολλαπλάσιο των 64Mb (Σχήμα 4.2). Σχεδιάζεται για χρήση από τις εργασίες MapReduce που διαβάζουν μεγάλα τμήματα (*chunks*) της εισόδου, τα επεξεργάζονται, και ενδεχομένως τα αποθηκεύουν ή πραγματοποιούν εγγραφές σε αυτά. Το HDFS δεν χειρίζεται την τυχαία προσπέλαση ιδιαίτερα καλά. Για λόγους αξιοπιστίας, το αρχείο δεδομένων, είναι απλά ένα αντίγραφο (*mirrored*) αποθηκευμένο σε πολλούς κόμβους. Αυτό αναφέρεται ως αντιγραφή *(replication)* στην κοινότητα Hadoop**.** Όσο τουλάχιστον ένα αντίγραφο ενός τμήματος δεδομένων (*data chunkd*) είναι διαθέσιμο, ο καταναλωτής εκείνων των δεδομένων δεν ξέρει για τις αποτυχίες που σχετίζονται με τον κεντρικό κόμβο αποθήκευσης (*server*).

Οι υπηρεσίες **HDFS** παρέχονται από δύο διαδικασίες:

- Το NameNode (*master*)
- Το DataNode (*workers*)

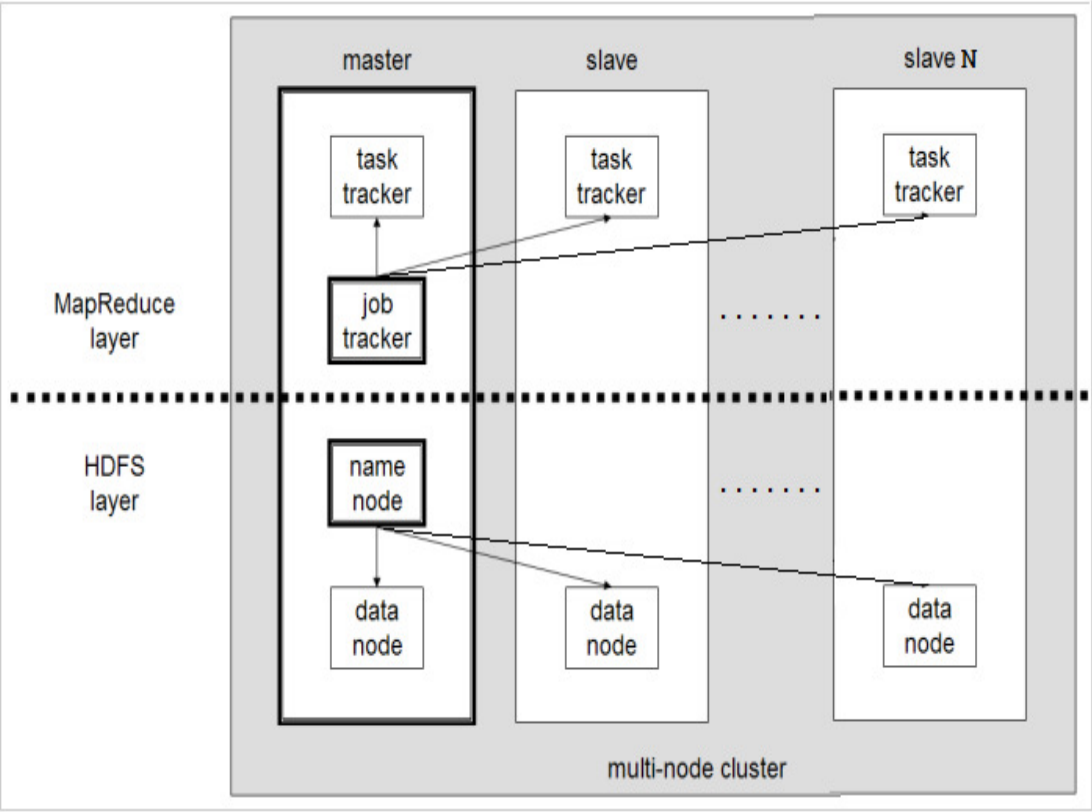
Το σύστημα αρχείων HDFS χρησιμοποιεί ένα κεντρικό κόμβο, τον NameNode, ο οποίος είναι και το μοναδικό σημείο αποτυχίας (*single point of failure- SPOF*), όπου κρατά τις πληροφορίες σχετικά με το που βρίσκεται κάθε δεδομένο στο HDFS.

Αν αυτός δεν είναι διαθέσιμος τότε δεν υπάρχει πρόσβαση στο σύστημα αρχείων. Οι άλλοι κόμβοι είναι οι DataNodes, οι οποίοι αποθηκεύουν τα δεδομένα.

Το framework παρέχει δύο διεργασίες (*processes*) που χειρίζονται τη διαχείριση των εργασιών MapReduce (*jobs*)**:**

- Ο *JobTracker* δέχεται τις υποβολές των εργασιών**,** παρέχει παρακολούθηση και έλεγχο**,** και διαχειρίζεται την κατανομή MapReduce εργασιών (*tasks*) στους κόμβους *TaskTracker* μέσα σε ένα cluster .
- Ο *TaskTracker* διαχειρίζεται την εκτέλεση των επιμέρους διεργασιών map και reduce σε έναν υπολογιστικό κόμβο στο cluster.

Γενικά, υπάρχει μια διεργασία *JobTracker* ανά *cluster* και μία ή περισσότερες διεργασίες *TaskTracker* ανά κόμβο στο *cluster*. [1]



Σχήμα 4.1 *Αρχιτεκρονική Hadoop Cluster (**Πηγή:**http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/)*

4.2 Αρχιτεκτονική του **HDFS**

4.2.1 Εισαγωγή

Το Hadoop Distributed File System (HDFS), Σχήμα 4.2, είναι ένα κατανεμημένο σύστημα αρχείων σχεδιασμένο να τρέχει σε κάθε υπολογιστή γενικής χρήσης. Επίσης έχει και κάποιες ομοιότητες με τα ήδη υπάρχοντα κατανεμημένα συστήματα αρχείων. Εντούτοις, οι διαφορές από τα άλλα κατανεμημένα συστήματα αρχείων είναι σημαντικές. Το HDFS είναι πολύ ανεκτικό στα σφάλματα (*fault-tolerant*) και σχεδιάστηκε για να τοποθετηθεί σε χαμηλού κόστους υλικό. Το HDFS προσφέρει μεγάλη ρυθμαπόδοση (*Throughput*) πρόσβασης σε εφαρμογές δεδομένων, και είναι κατάλληλο για εφαρμογές που έχουν μεγάλο σύνολο δεδομένων. Το HDFS μετριάζει μερικές προϋποθέσεις για να ενεργοποιήσει την

πρόσβαση στο σύστημα αρχείων δεδομένων. Αρχικά δημιουργήθηκε ως μια υποδομή για το σχέδιο μηχανής αναζήτησης Apache Nutch στο web. Το HDFS αποτελεί πλέον ένα υποπρότζεκτ του Apache Hadoop**.**

4.3 **Π**ροϋποθέσεις και **Σ**τόχοι

4.3.1 **Α**στοχία **Υ**λικού (Hardware Failure)

Η Αστοχία Υλικού αποτελεί μάλλον ένα μέτρο απόδοσης παρά μια εξαίρεση. Ένα HDFS στιγμιότυπο μπορεί να απαρτίζεται από εκατοντάδες ή χιλιάδες servers, οπού ο καθένας αποθηκεύει κομμάτι του συστήματος αρχείων δεδομένων. Λόγω του γεγονότος ότι υπάρχει μεγάλος αριθμός εξαρτημάτων και ότι το κάθε εξάρτημα έχει μια σημαντική πιθανότητα αστοχίας, κάποια εξαρτήματα είναι συνήθως μη – λειτουργικά. Επομένως, η γρήγορη ανίχνευση βλαβών και η αυτόματη επαναφορά από αυτά, είναι ένας από τους σκοπούς της αρχιτεκτονικής του πυρήνα του HDFS.

4.3.2 **Π**ρόσβαση **Ρ**οής **Δ**εδομένων (Data Stream Access)

Οι εφαρμογές που τρέχουν στο HDFS χρειάζονται συνεχή ροή πρόσβασης στα δικά τους σύνολα δεδομένων. Δεν είναι εφαρμογές γενικού σκοπού οπότε τυπικά τρέχουν σε συστήματα δεδομένων γενικού σκοπού. Το HDFS σχεδιάστηκε πιο πολύ για επεξεργασία κατά δέσμες (*batch processing*), και όχι για διαδραστική χρήση από τους χρήστες. Επίσης, δίνει πρόσβαση στα δεδομένα με υψηλό ρυθμό, και όχι με χαμηλή καθυστέρηση.

4.3.3 **Δ**εδομένα **Μ**εγάλου **Ό**γκου **(**Data Sets**)**

Οι εφαρμογές που τρέχουν στο HDFS περιέχουν μεγάλα σύνολα δεδομένων. Ένα συνήθες αρχείο στο HDFS έχει μέγεθος από μερικά GigaBytes μέχρι ακόμα και TeraBytes/Petabytes. Άρα, το HDFS έχει σχεδιαστεί και ρυθμιστεί για να υποστηρίζει μεγάλα αρχεία. Θα πρέπει να παρέχει μεγάλο σύνολο εύρος ζώνης δεδομένων και να κλιμακώνεται σε εκατοντάδες κόμβους σε ένα cluster. Επίσης, Θα πρέπει να υποστηρίζει δεκάδες εκατομμύρια αρχεία σε μια μόνο περίοδο.

4.3.4 **Μ**οντέλο **Σ**υνοχής (Coherency Model)

Οι HDFS εφαρμογές χρειάζονται ένα μοντέλο πρόσβασης μίας εγγραφής-πολλών αναγνώσεων στα αρχεία (*write-once/read-many* – WORM). Ένα αρχείο αφού δημιουργηθεί, εγγραφτεί, δεν επιτρέπεται να τροποποιηθεί. Αυτή η παραδοχή απλοποιεί τα δεδομένα συνοχής που μοιράζονται και ενεργοποιεί την υψηλή ρυθμαπόδοση δεδομένων πρόσβασης. Μια MapReduce εφαρμογή ή μια εφαρμογή περιήγησης web ταιριάζει εξαιρετικά στο μοντέλο αυτό.

4.3.5 **Μ**εταφορά της **Ε**κτέλεσης Επεξεργασίας «**Κ**οντά» στα **Δ**εδομένα

Μια εκτέλεση επεξεργασίας είναι πολύ πιο αποδοτική οταν εκτελούνται κοντά στα δεδομένα που χρησιμοποιούν. Αυτό ισχύει ειδικά όταν το μέγεθος του data set είναι τεράστιο. Αυτή η δυνατότητα ελαχιστοποιεί τη συμφόρηση του δικτύου και αυξάνει τη ρυθμαπόδοση του συστήματος. Έτσι,αποφεύγεται η μεταφορά του data set μέσα απο το δίκτυο, η οποία είναι ιδιαίτερα χρονοβόρα, γεγονός που θα καθυστερούσε σημαντικά την εκτέλεση της εφαρμογής. Η φιλοσοφία είναι ότι συχνά είναι προτιμότερη η μεταφορά υπολογισμών πιό κοντά στο σημείο όπου βρίσκονται τα δεδομένα παρά η μεταφορά των δεδομένων στο σημείο όπου η εφαρμογή εκτελείται. Το HDFS παρέχει interfaces για τις εφαρμογές για να μεταφέρονται πιό κοντά στα δεδομένα.

4.3.6 **Φ**ορητότητα σε **Ε**τερογενές **Π**λατφόρμες **Υ**λικού και **Λ**ογισμικού**.**

Το HDFS έχει σχεδιαστεί για να είναι εύκολα μεταφέρσιμο από μια πλατφόρμα στην άλλη. Αυτές οι διευκολύνσεις είναι ευρύτατα διαδεδομένες και υιοθετούνται από το HDFS ως μια πλατφόρμα επιλογής για ένα μεγάλο σύνολο εφαρμογών.

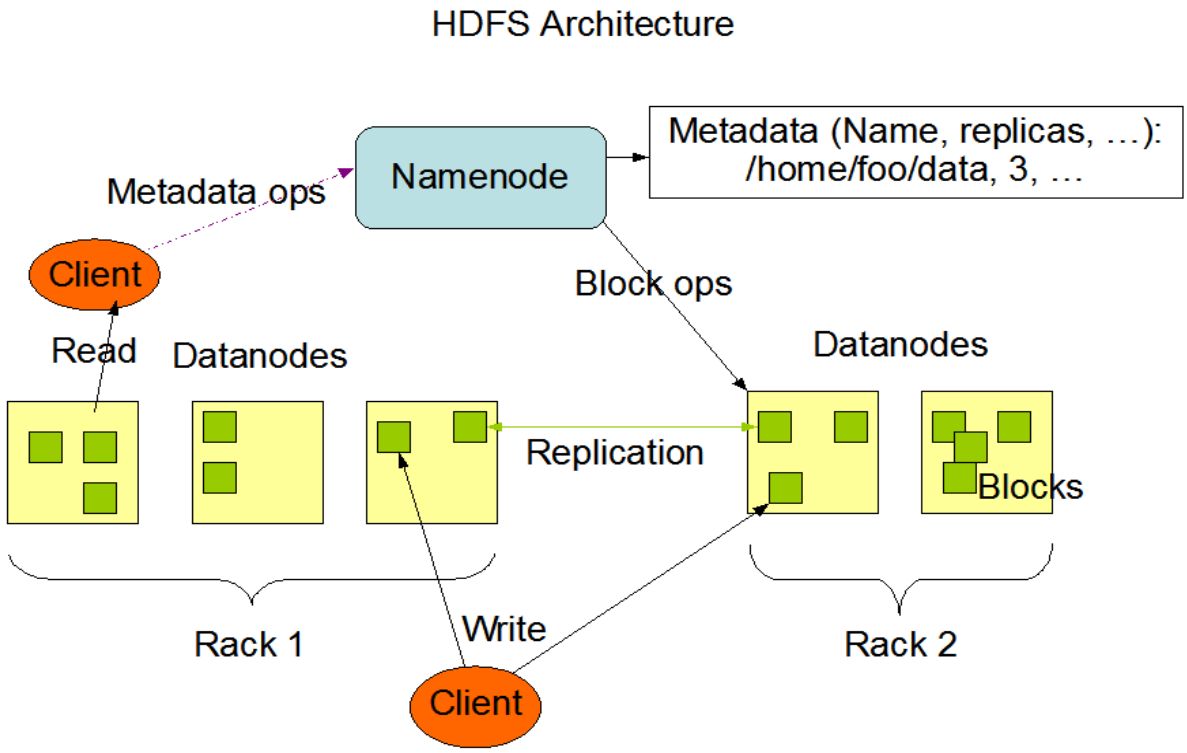
4.4 **N**ame**N**ode και **D**ata**N**odes

Το HDFS αποτελεί μια master/slave αρχιτεκτονική. Ένα HDFS cluster αποτελείται από ένα single NameNode, ένα master server, το οποίο διαχειρίζεται το σύστημα αρχείων namespace και ελέγχει την πρόσβαση των πελατών στα αρχεία. Συμπληρωματικά, υπάρχει ένας αριθμός από DataNodes, συνήθως ένα για κάθε κόμβο του cluster, οι οποίοι διαχειρίζονται την αποθήκευση αποκλειστικά για τους κόμβους στους οποίους εκτελούνται. Το HDFS παρουσιάζει ένα σύστημα αρχείων namespace και επιτρέπει στα δεδομένα των χρηστών να αποθηκευτούν σε αρχεία. Εσωτερικά, ένα αρχείο διασπάται σε ένα ή

περισσότερα blocks και αυτά σε ένα σύνολο από DataNodes. Το NameNode εκτελεί λειτουργίες συστήματος αρχείου namespace όπως άνοιγμα, κλείσιμο, μετονομασία αρχείων και καταλόγων. Αυτό επίσης προσδιορίζει το mapping των blocks στα DataNodes. Τα DataNodes είναι υπεύθυνα για την εξυπηρέτηση των αιτημάτων ανάγνωσης και εγγραφής από τα συστήματα αρχείων των πελατών. Τα DataNodes επίσης, εκτελούν blocks δημιουργίας, διαγραφής, και αναπαραγωγή επί εντολής από το NameNode.

Τα NameNode και DataNode είναι κομμάτια λογισμικού, σχεδιασμένα να τρέχουν σε μηχανήματα εμπορίου. Αυτά τα μηχανήματα συνήθως τρέχουν GNU/Linux λειτουργικό σύστημα (*Operating System* - OS). Το HDFS κατασκευάστηκε με χρήση της γλώσσας Java, οπότε κάθε μηχάνημα που υποστηρίζει Java μπορεί να τρέχει το NameNode ή το DataNode λογισμικό. Η υψηλή φορητότητα (*portability*) της Java σημαίνει, ότι το HDFS μπορεί να τοποθετηθεί σε μια μεγάλη ποικιλία μηχανημάτων. Μια τυπική υλοποίηση έχει ένα αποκλειστικό μηχάνημα στο οποίο εκτελείται μόνο το NameNode. Το κάθε ένα από τα άλλα μηχανήματα στο cluster εκτελεί ένα στιγμιότυπο από το DataNode λογισμικό. Η αρχιτεκτονική δεν αποκλείει το να εκτελούνται πολλαπλά DataNodes στο ίδιο μηχάνημα, αλλά σε ένα αληθινό περιβάλλον, όπου κάτι τέτοιο αποτελεί σπάνια περίπτωση.

Η ύπαρξη ενός NameNode σε ένα cluster απλοποιεί σε μεγάλο βαθμό την αρχιτεκτονική του συστήματος. Ο NameNode είναι ο διαιτητής και η αποθήκη όλων των μεταδεδομένων HDFS. Επιπρόσθετα το σύστημα είναι σχεδιασμένο με τέτοιο τρόπο, όπου τα δεδομένα του χρήστη ποτέ δεν ρέουν από το NameNode.

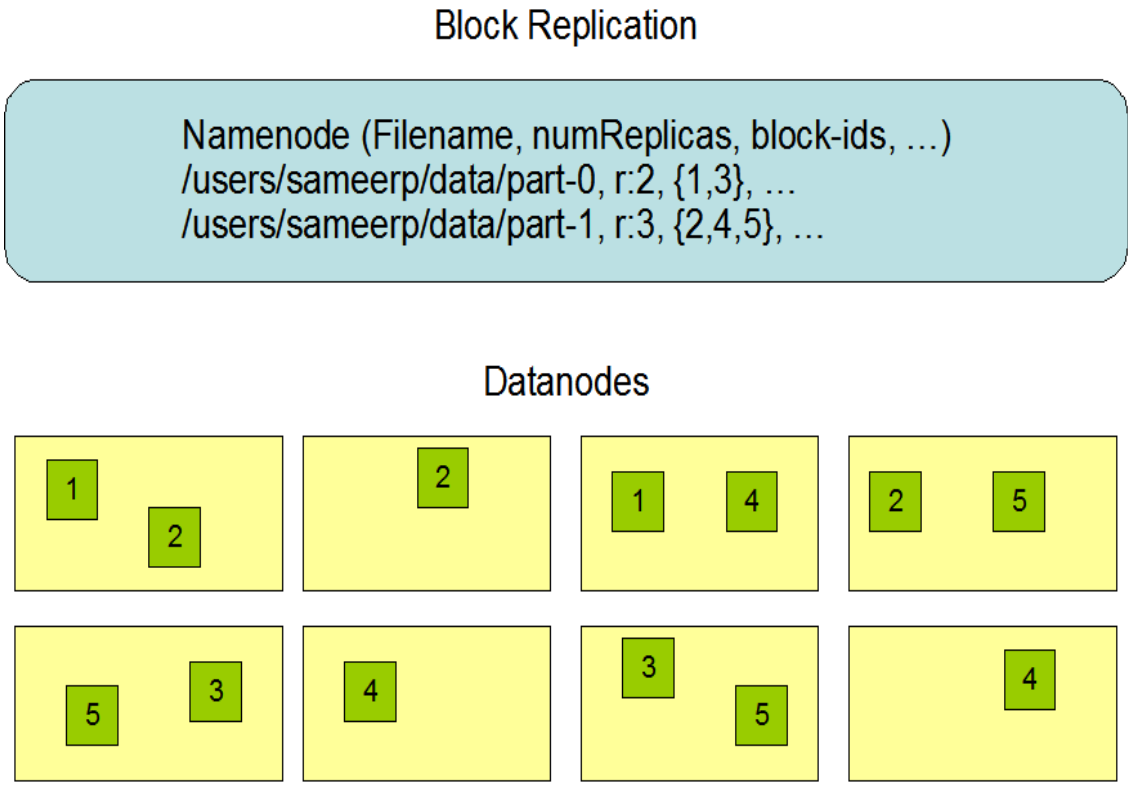


Σχήμα 4.2: *Η Αρχιτεκτονική του HDFS*

4.5 Αντίγραφα Δεδομένων στο HDFS

Το HDFS είναι σχεδιασμένο να αποθηκεύει αξιόπιστα πολύ μεγάλα αρχεία στις συσκευές σε ένα μεγάλο cluster (Σχήμα 4.3). Αποθηκεύει το κάθε αρχείο σαν μια ακολουθία από blocks, εκ των οποίων όλα εκτός του τελευταίου έχουν το ίδιο μέγεθος. Τα blocks του αρχείου αντιγράφονται σε πολλές μηχανές για να εξασφαλιστεί η ανοχή σε σφάλματα (*fault tolerance*). Το μέγεθος του block και ο παράγοντας αντιγραφής (*replication factor*) μπορει να ρυθμιστεί για κάθε αρχείο χωριστά. Επίσης μια εφαρμογή μπορεί να καθορίσει τον αριθμό των αντιγράφων του αρχείου. Ο παράγοντας αντιγραφής μπορεί να καθοριστεί τη στιγμή δημιουργίας του αρχείου και να αλλαχτεί αργότερα.

Ο NameNode παίρνει όλες τις αποφάσεις που αφορούν την αντιγραφή των blocks. Περιοδικά λαμβάνει ένα Heartbeat και ένα Blockreport από το κάθε DataNode στο cluster. Η λήψη του Heartbeat υποδηλώνει ότι το DataNode είναι σε κανονική λειτουργία, καθώς και ένα Blockreport περιλαμβάνει μια λίστα από όλα τα blocks σε DataNode.



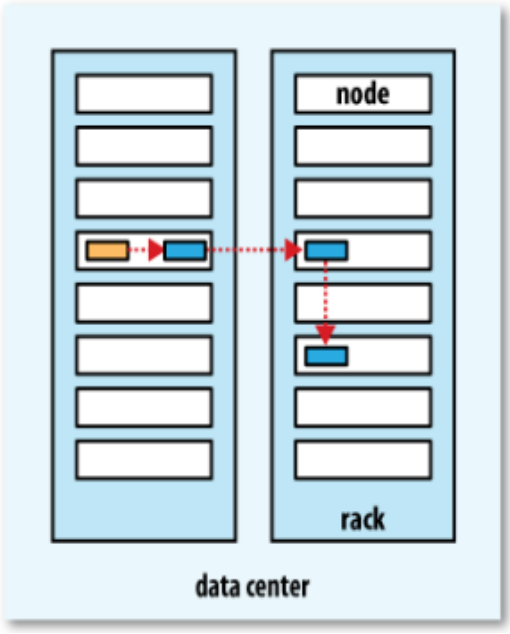
Σχήμα 4.3: *Διατήρηση αντιγράφων αρχείων*

4.5.1 **Α**ντιγραφή **Δ**εδομένων στο **H**DFS

Η τοποθέτηση των αντιγράφων είναι σημαντική για την αξιοπιστία και την απόδοση του HDFS συστήματος. Η βελτιστοποίηση της αντιγραφής των blocks διακρίνει το HDFS από τα περισσότερα κατανεμημένα συστήματα αρχείων. Ο σκοπός ενός rack-aware πολιτικής τοποθέτησης αντιγράφων είναι να βελτιώσει την αξιοπιστία των δεδομένων, τη διαθεσιμότητα, και τη χρήση εύρους ζώνης του δικτύου.

Μεγάλα HDFS στιγμιότυπα που τρέχουν σε ένα cluster υπολογιστών συνήθως κατανέμονται σε πολλά racks, η επικοινωνία των οποίων γίνεται μέσω Switches. Στις περισσότερες περιπτώσεις, το εύρος ζώνης (*bandwidth*) του δικτύου μεταξύ των μηχανών στον ίδιο rack είναι μεγαλύτερη από το εύρος ζώνης του δικτύου μεταξύ των μηχανών σε διαφορετικά racks.

Η NameNode μέσω μιας διαδικασίας καθορίζει τον αριθμό του rack (*rack id*) του κάθε κόμβου. Όπως αναφέρθηκε, ο παράγοντας αντιγραφής έχει την προεπιλεγμένη τιμή 3. Όπως φαίνεται στο Σχήμα 4.4, η προεπιλεγμένη στρατηγική του HDFS είναι να τοποθετεί το πρώτο αντίγραφο στον ίδιο κόμβο. Το δεύτερο αντίγραφο τοποθετείται σε διαφορετικό rack. Το τρίτο αντίγραφο τοποθετείται στο ίδιο rack. Περαιτέρω αντίγραφα τοποθετούνται σε τυχαίους κόμβους του cluster, παρόλο που το σύστημα προσπαθεί να αποφύγει την τοποθέτηση πάρα πολλών αντιγράφων στον ίδιο rack. Η πολιτική αυτή βελτιώνει την απόδοση των εγγραφών χωρίς να θέτει σε κίνδυνο την αξιοπιστία των δεδομένων ή την επίδοση ανάγνωσης.



Σχήμα 4.4: *Στρατηγική τοποθέτησης αντιγράφων του HDFS (**Πηγή:** Tom White, (2010): Hadoop: The Definitive Guide, 2nd Edition. O'Reilly Media/Yahoo Press)*

4.5.2 **Ε**πιλογή **Α**ντιγράφων

Για να μειωθεί η κατανάλωση του bandwidth και η καθυστέρηση της ανάγνωσης, το HDFS προσπαθεί να ικανοποιήσει την αίτηση ανάγνωσης, δίνοντας το αντίγραφο το οποίο είναι πιο κοντά στον κόμβο που ζήτησε να διαβάσει. Εάν υπάρχει ήδη ένα αντίγραφο στο ίδιο rack, τότε αυτό προτιμάται. Αν ένα cluster εκτείνεται σε πολλά υπολογιστικά κέντρα (*data center*), τότε το αντίγραφο που παραμένει στο τοπικό υπολογιστικό κέντρο προτιμάται περισσότερο σε σχέση με κάποιο απομακρυσμένο αντίγραφο.

4.5.3 **R**ack **A**wareness

Συνηθισμένη αρχιτεκτονική για Hadoop Cluster αποτελεί η τοπολογία δύο επιπέδων. Για παράδειγμα, σε κάθε rack υπάρχουν 30-40 κόμβοι που συνδέονται με 1GB switch και τα switches αυτά συνδέονται σε ένα κεντρικό switch >=1GB. Το βασικό χαρακτηριστικό είναι ότι το συνολικό bandwidth μεταξύ κόμβων στο ίδιο rack είναι πολύ μεγαλύτερο από αυτό μεταξύ διαφορετικών racks και για αυτό το λόγο το Hadoop προτιμάει να κάνει μεταφορές μέσα στο ίδιο rack παρά μεταξύ διαφορετικών racks, όταν κάνει ανάθεση MapReduce εργασιών στους κόμβους. Η τοπολογία του δικτύου καθορίζει και τον τρόπο με τον οποίο το Hadoop τοποθετεί τα αντίγραφα στους κόμβους.

4.5.4 **S**afe **M**ode

Κατά την εκκίνηση του HDFS, ο NameNode τίθεται στην κατάσταση **SafeMode** (Κατάσταση Ασφαλούς Λειτουργίας). Αναπαραγωγή των blocks δεδομένων δεν συμβαίνει όταν το NameNode βρίσκεται σε κατάσταση SafeMode. Ο NameNode λαμβάνει μηνύματα Heartbeat και Blockreport από τα DataNodes. Ένα Blockreport περιλαμβάνει τη λίστα των blocks δεδομένων τα οποία φιλοξενεί ο DataNode. Το κάθε block έχει έναν ελάχιστο προκαθορισμένο αριθμό αντιγράφων. Ένα block θεωρείται ασφαλώς κατασκευασμένο όταν ο ελάχιστος αριθμός από αντίγραφα αυτού του block έχει ελεγχθεί μαζί με το NameNode. Μετά τον υπολογισμό του ποσοστού των ασφαλώς κατασκευασμένων blocks δεδομένων ελέγχεται μαζί με το NameNode (επιπρόσθετα 30 δευτερόλεπτα), το NameNode βγαίνει από την κατάσταση SafeMode. Αυτό καθορίζει τη λίστα από blocks δεδομένων τα οποία έχουν ακόμη λιγότερα αντίγραφα από ότι ο προκαθορισμένος αριθμός αντιγράφων. Ο NameNode τότε αντιγράφει αυτά τα blocks στα υπόλοιπα DataNodes.

4.6 **Π**ρωτόκολλα **Ε**πικοινωνίας

Όλα τα HDFS πρωτόκολλα επικοινωνίας είναι διαστρωματωμένα στην κορυφή των TCP/IP πρωτοκόλλων. Κάθε πελάτης καθιερώνει μια σύνδεση για να ρυθμίσει μια TCP θύρα στο NameNode μηχάνημα. Αυτό συνδέει το ClientProtocol με το NameNode. Οι DataNodes μιλάνε με το NameNode χρησιμοποιώντας το DataNode πρωτόκολλο. Μια κλήση απομακρυσμένων διαδικασιών (*Remote Procedure Call - RPC*) αφαίρεσης τυλίγει και το Client Protocol και το DataNode Protocol. Εκ προθέσεως, ο NameNode ποτέ δεν εκκινεί κάποια RPCs. Αντ' αυτού, απαντάει μόνο στα RPC αιτήματα που εξέρχονται από τα DataNodes ή τους πελάτες.

4.7 **Ε**υρωστεία (Robustness)

Ο κύριος στόχος της HDFS είναι να αποθηκεύουν αξιόπιστα τα δεδομένα ακόμη και σε παρουσία των αποτυχιών. Οι τρεις κοινοί τύποι των αστοχιών είναι οι NameNode αστοχίες, οι DataNode αστοχίες και οι κατατμήσεις του δικτύου.

4.7.1 **D**ata **D**isk **Α**ποτυχία, **H**eartbeats και **R**e-**R**eplication

Κάθε DataNode στέλνει περιοδικά ένα μήνυμα Heartbeat στο NameNode . Ενα τμήμα του δικτύου μπορεί να είναι αίτιο σε ένα υποσύνολο των DataNodes να διακόψουν τη σύνδεση με τον NameNode. Ο NameNode εντοπίζει αυτή την κατάσταση από την απουσία ενός μηνύματος Heartbeat. Ο NameNode χωρίς πρόσφατα σήματα Heartbeats θεωρεί τα DataNodes ως 'νεκρά' και δεν προωθεί νέα IO αιτήματα σε αυτούς. Όλα τα δεδομένα που είχαν καταγραφεί στο 'νεκρό' DataNode δεν είναι διαθέσιμο πλέον στο HDFS. Ο 'θάνατος' του DataNode ενδέχεται να αναγκάσει τον παράγοντα αντιγραφής ορισμένων blocks να μειώσει την καθορισμένη τιμή του προς τα κάτω . Ο NameNode ελέγχει διαρκώς ποια blocks θα πρέπει να αντιγραφούν και ξεκινά την αντιγραφή όποτε είναι απαραίτητο. Η ανάγκη για την εκ νέου αντιγραφή ενδέχεται να προκύψει για πολλούς λόγους: ένας DataNode μπορεί να καταστεί μη διαθέσιμος, ένα αντίγραφο ενδέχεται να καταστραφεί, ένας σκληρός δίσκος σε έναν DataNode μπορεί να αποτύχει, ή ο παράγοντας αντιγραφής ενός αρχείου μπορεί να αυξηθεί.

4.7.2 **Α**νακατανομή του **Φ**όρτου **C**luster (Cluster Rebalancing)

Η HDFS αρχιτεκτονική είναι συμβατή με τα συστήματα εξισορρόπησης δεδομένων. Το σύστημα πρέπει αυτόματα να μετακινήσει δεδομένα από ένα DataNode σε ένα άλλο, εάν ο ελεύθερος χώρος στο DataNode πέφτει κάτω από το προκαθορισμένο κατώφλι. Στην περίπτωση μιας ξαφνικής αύξησης της απαίτησης για ένα συγκεκριμένο αρχείο, το σύστημα δημιουργεί δυναμικά επιπλέον αντίγραφα και εξισορροπεί τα άλλα δεδομένα στο cluster.

4.7.3 **Α**κεραιότητα **Δ**εδομένων (Data Integrity)

Είναι πιθανό ένα block των δεδομένων που φορτώνεται από ένα DataNode να φτάνει κατεστραμμένο. Η αλλοίωση μπορεί να συμβεί λόγω σφαλμάτων σε μια συσκευή αποθήκευσης, βλάβες του δικτύου, η λάθη λογισμικού. Το λογισμικό client HDFS υλοποιεί checksum έλεγχο σχετικά με το περιεχόμενο των αρχείων HDFS. Όταν ένας πελάτης (client) δημιουργεί ένα αρχείο HDFS, τότε υπολογίζει το άθροισμα ελέγχου (*checksum*) του κάθε block ενός αρχείου και αποθηκεύει αυτά τα αθροίσματα ελέγχου σε ένα ξεχωριστό κρυφό αρχείο στον ίδιο HDFS namespace. Όταν ένας πελάτης ανακτά τα περιεχόμενα του αρχείου επαληθεύει ότι τα δεδομένα που έλαβε από κάθε DataNode ταιριάζει με το άθροισμα ελέγχου που αποθηκεύεται στο σχετικό αρχείο checksum. Αν όχι, τότε ο πελάτης μπορεί να επιλέξει για να ανακτήσει το εν λόγω block από έναν άλλο DataNode που έχει ένα αντίγραφο εκείνου του block.

4.8 **D**ata **O**rganization

4.8.1 **D**ata **B**locks

Το HDFS έχει σχεδιαστεί για να υποστηρίζει πολύ μεγάλα αρχεία. Εφαρμογές που είναι συμβατές με HDFS είναι εκείνες που εξετάζουν τα μεγάλα σύνολα δεδομένων. Αυτές οι εφαρμογές γράφουν τα δεδομένα τους μόνο μια φορά αλλά τα διαβάζουν μια ή περισσότερες φορές και απαιτούν από αυτές τις αναγνώσεις να ικανοποιούν τις ταχύτητες ροής. Το HDFS υποστηρίζει τη σημασιολογία μίας εγγραφής-πολλών αναγνώσεων στα αρχεία. Ένα τυπικό μέγεθος block που χρησιμοποιείται από το HDFS είναι τα 64 MB. Επομένως, ένα αρχείο HDFS τεμαχίζεται σε κομμάτια (*chunks*) των 64 MB, και ενδεχομένως, κάθε κομμάτι θα βρίσκεται σε διαφορετικό DataNode.

4.8.2 **Α**ντίγραφα **Δ**ιοχέτευσης (**Replication Pipelining**)

Όταν ένας χρήστης (*client*) γράφει τα δεδομένα του σε ένα αρχείο HDFS, αυτά γράφονται αρχικά σε ένα τοπικό αρχείο. Υποθέστε ότι το αρχείο HDFS έχει έναν παράγοντα αντιγραφής ίσο με τρία.

Όταν το τοπικό αρχείο συγκεντρώνει ένα πλήρες block των δεδομένων του χρήστη, ο client ανακτά μια λίστα με DataNodes από το NameNode. Αυτή η λίστα περιέχει τα DataNodes που θα φιλοξενήσει ένα αντίγραφο αυτού του block. Ο client αδειάζει το block δεδομένων στον πρώτο DataNode. Το πρώτο DataNode λαμβάνει τα δεδομένα σε μικρά κομμάτια (4 KB), γράφει κάθε κομμάτι (*portion*) στο τοπικό repository και μεταφέρει αυτό το κομμάτι στη λίστα του δεύτερου DataNode. Ο δεύτερος DataNode, αρχίζει να λαμβάνει το κάθε κομμάτι των block δεδομένων, γράφοντας το κάθε κομμάτι στο χώρο του και έπειτα αδειάζει εκείνο το κομμάτι στο τρίτο DataNode. Τέλος, το τρίτο DataNode γράφει τα δεδομένα στο τοπικό του repository. Επομένως, ένα DataNode μπορεί να λαμβάνει δεδομένα κατά την διοχέτευση από τον προηγούμενο και την ίδια στιγμή τα προωθεί στην επόμενη διοχέτευση. Συνεπώς, τα δεδομένα διοχετεύονται από έναν DataNode στο άλλο.

4.9 **Π**ροσβασιμότητα

Στο HDFS μπορεί κανείς να έχει πρόσβαση από εφαρμογές με πολλούς διαφορετικούς τρόπους. Εσωτερικά, το HDFS παρέχει το Java API για χρήση από τις εφαρμογές. Ένα περιτύλιγμα της γλώσσας C για το Java API είναι επίσης διαθέσιμο. Επιπρόσθετα, ένας φυλλομετρητής HTTP μπορεί επίσης να χρησιμοποιηθεί για την αναζήτηση των αρχείων μιας περίπτωσης HDFS. Η εργασία είναι υπό εξέλιξη για να παρουσιάσει το HDFS μέσω του πρωτοκόλλου WebDAV.

4.9.1 **F**S **S**hell

Το HDFS επιτρέπει στα δεδομένα χρηστών να οργανωθούν υπό τη μορφή αρχείων και καταλόγων. Παρέχει τη διεπαφή command line, η οποία ονομάζεται FS Shell, και επιτρέπει σε έναν χρήστη να αλληλεπιδράσει με τα δεδομένα στο HDFS. Η σύνταξη αυτού του συνόλου εντολής είναι παρόμοια με άλλα shells (π.χ. bash, csh). Παρακάτω στον Πίνακα 4.1 παραθέτονται μερικά δείγματα από ενέργειες/εντολές:

Πίνακας 4.1

Action	Command
Δημιούργησε έναν κατάλογο με όνομα /foodir	bin/hadoop dfs –mkdir /foodir
Απομάκρυνε τον κατάλογο με όνομα /foodir	bin/hadoop dfs –rmr /foodir
Εμφάνισε τα περιεχόμενα του αρχείου με όνομα /foodir/myfile.txt	bin/hadoop dfs -cat /foodir/myfile.txt

Το FS Shell στοχεύει στις εφαρμογές που χρειάζονται μια scripting γλώσσα για να αλληλεπιδράσουν με τα αποθηκευμένα δεδομένα.

4.9.2 **D**FS**A**dmin

Το σύνολο εντολών DFSAdmin χρησιμοποιείται για τη διαχείριση μιας συστάδας HDFS. Αυτές είναι οι εντολές που χρησιμοποιούνται μόνο από έναν διαχειριστή HDFS. Στον πίνακα 4.2 παραθέτονται μερικά δείγματα από ενέργειες/εντολές:

Πίνακας 4.2

Action	Command
Βάλε τη cluster σε κατάσταση ασφαλούς λειτουργίας.	bin/hadoop dfsadmin -safemode enter
Δημιούργησε μια λίστα από DataNodes	bin/hadoop dfsadmin - report
Ανέθεσε ή αφόπλισε τα DataNode(s)	bin/hadoop dfsadmin -refreshNodes

4.9.3 **Δ**ιεπαφή **Φ**υλλομετρητή (Browser Interface)

Ένα χαρακτηριστικό του HDFS είναι ότι εγκαθιστά ρυθμίσεις σε έναν κεντρικό υπολογιστή δικτύου για να παρουσιάσει το HDFS namespace μέσω ενός διαμορφώσιμου

TCP port. Αυτό επιτρέπει σε έναν χρήστη να πλοηγηθεί στο HDFS namespace και να δει το περιεχόμενο των αρχείων του χρησιμοποιώντας έναν φυλλομετρητή ιστοσελίδων (*web browser*). [2]

ΑΝΑΦΟΡΕΣΣ

[1]. Jason Venner, (2009). *Pro Hadoop*. 1st ed. : Apress, (5-6 σελ.)

[2]. HDFS Architecture **-** http://hadoop.apache.org/docs/stable/hdfs_design.html#Browser+Interface

ΚΕΦΑΛΑΙΟ 5ο

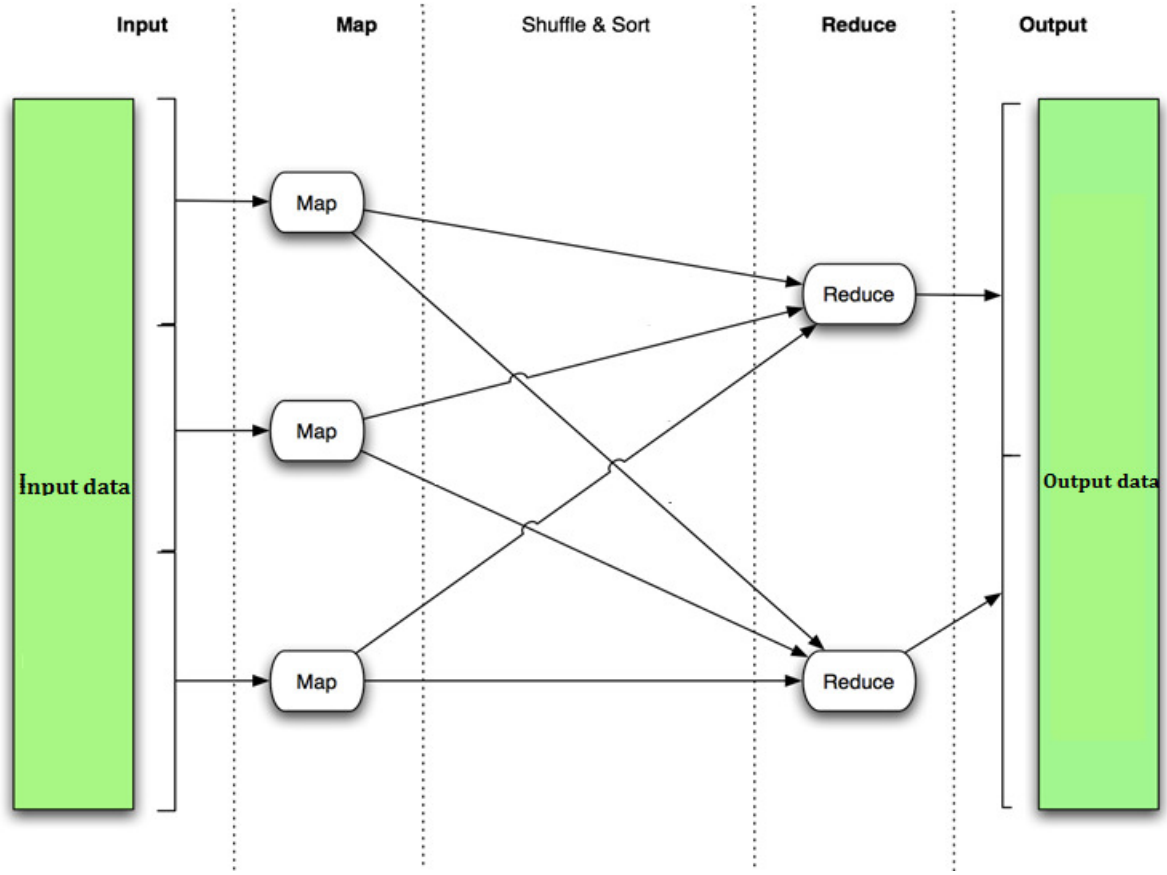
ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΟ ΜΟΝΤΕΛΟ MAPREDUCE

5.1 ΤΙ ΕΙΝΑΙ ΤΟ MAPREDUCE;

Το Hadoop MapReduce είναι ένα ένα framework λογισμικού (*software framework*), για την αποδοτική διαχείριση - επεξεργασία των τεράστιων όγκων δεδομένων που θεσπίστηκε από τη **Google** το 2004. Στόχος ήταν να παρέχει εύκολο γράψιμο εφαρμογών, οι οποίες επεξεργάζονται ένα τεράστιο όγκο δεδομένων (*multi-terabyte data-sets*) παράλληλα σε μεγάλα clusters (*thousands of nodes*) από υπολογιστές γενικής χρήσης, παρέχοντας ταυτόχρονα αξιοπιστία και ανοχή σε σφάλματα. Το Hadoop μπορεί να τρέξει προγράμματα MapReduce που είναι γραμμένα σε γλώσσες όπως *Java, Ruby, Python* και *C++,* καθώς και σε άλλες γλώσσες. Έχει εμπνευστεί από τις συναρτήσεις map και reduce στον συναρτησιακό (*functional*) προγραμματισμό.

Μια εργασία *(job)* MapReduce χωρίζει συνήθως το σύνολο δεδομένων εισαγωγής σε ανεξάρτητα blocks, τα οποία υποβάλλονται σε επεξεργασία από τους map tasks κατά τρόπο τελείως παράλληλο. Το πλαίσιο εργασίας ταξινομεί τα αποτελέσματα των maps, τα οποία στη συνέχεια εισάγουν reduce tasks. Τυπικά οι δυο είσοδοι και οι έξοδοι της εργασίας αποθηκεύονται σε ένα αρχείο συστήματος. Το framework φροντίζει για τον χρονοπρογραμματισμό εργασιών, την παρακολούθηση και την επάν-εκτέλεση αποτυχημένων εργασιών (*tasks*). Τυπικά οι κόμβοι που εκτελούν τους υπολογισμούς και την αποθήκευση είναι η ίδιοι, δηλαδή, το MapReduce framework και το ***Hadoop Distributed File System*** **(*HDFS*)** εκτελούνται στο ίδιο σύνολο κόμβων. Αυτή η διαμόρφωση επιτρέπει στο MapReduce framework να χρονοπρογραμματίσει αποτελεσματικά τις εργασίες στους κόμβους όπου τα δεδομένα είναι ήδη παρόντα, με αποτέλεσμα πολύ υψηλό συνολικό εύρος ζώνης σε όλο το cluster**.** [1]

Ο μηχανισμός του MapReduce λειτουργεί με βάση της αρχιτεκτονικής master/slave. Το πλαίσιο MapReduce αποτελείται από ένα master JobTracker και έναν slave TaskTracker ανά cluster-node**.** Ο master είναι υπεύθυνος για τον προγραμματισμό των εργασιών (*tasks*) στους slaves - TaskTrackers, την παρακολούθηση τους και την εκ νέου εκτέλεσή τους σε τυχόν περιστατικά αποτυχίας**.** Οι slaves - TaskTrackers εκτελούν τις εργασίες (*tasks*) σύμφωνα με τις οδηγίες του master - JobTracker.



Σχήμα 5.1: *MapReduce Ροή (**Πηγή**: Πήγη: Data-Intensive Text Processing with MapReduce. Jimmy Lin and Chris Dyer. Morgan & Claypool Publishers, 2010)*

"Map" βήμα: Ο κύριος κόμβος παίρνει την είσοδο, τη διαιρεί σε μικρότερα επιμέρους προβλήματα, και τα κατανέμει στους κόμβους των εργαζομένων. Ένας κόμβος εργαζομένων μπορεί να κάνει αυτό πάλι στη συνέχεια, που οδηγεί σε μια πολυεπίπεδη δενδρική δομή. Ο κόμβος εργαζόμενος επεξεργάζεται το μικρότερο πρόβλημα, και περνά την απάντηση στον κύριο κόμβο του.

"Reduce" βήμα: Ο κύριος κόμβος συλλέγει έπειτα τις απαντήσεις για όλα τα επιμέρους προβλήματα και τις συνδυάζει με κάποιο τρόπο για να διαμορφώσει την έξοδο - την απάντηση στο πρόβλημα που προσπαθούσε αρχικά να λύσει.

"Shuffle" βήμα: Η φάση shuffle ταξινομεί τα ζεύγη αποτελεσμάτων που προκύπτουν τοπικά από τα κλειδιά της map φάσης, μετά από την οποία, η MapReduce τους αναθέτει σε ένα reducer σύμφωνα με αυτά τα κλειδιά. Το framework εξασφαλίζει ότι όλα τα ζεύγη με το ίδιο κλειδί ανατίθενται στον ίδιο reducer. Επειδή η έξοδος της map φάσης μπορεί να κατανεμηθεί

αυθαίρετα στο cluster, η έξοδος από τη φάση θα πρέπει να μεταφερθεί σε όλο το δίκτυο στους σωστούς παραγωγούς της φάσης Shuffle.

"Sort" βήμα: Κάθε reduce task ευθύνεται για τη μείωση των τιμών που συνδέονται με αρκετά ενδιάμεσά κλειδιά. Το σύνολο των ενδιάμεσων κλειδιών για ένα μόνο κόμβο ταξινομείται αυτόματα από το Hadoop προτού υποβληθούν στην Reducer.

"Combiner" βήμα: Combiner ονομάζεται εκείνη η φάση που τρέχει μετά την Mapper και πριν από την Reducer. *Η χρήση του Combiner είναι προαιρετική*. Εάν αυτή η φάση είναι κατάλληλη για την εργασία σας, στοιχεία της Combiner class τρέχουν σε κάθε κόμβο που έχει τρέξει τα map tasks. Ο Combiner θα λαμβάνει σαν είσοδο όλα τα δεδομένα που εκπέμπονται από τα στοιχεία που Mapper από έναν συγκεκριμένο κόμβο. Η έξοδος από τον Combiner στη συνέχεια αποστέλνεται στους Reducers, αντί της εξόδου από τους Mappers. Ο Combiner είναι ένα "mini-reduce" διαδικασία η οποία λειτουργεί μόνο σε δεδομένα που προέρχονται από ένα μηχάνημα. [2]

5.2 Λογική Όψη

Οι συναρτήσεις **Map** και **Reduce** του MapReduce ορίζονται όσον αφορά τα δομημένα δεδομένα σε (*key, value*) ζεύγη**.** Η Map παίρνει ένα ζευγάρι των δεδομένων με έναν τύπο σε ένα πεδίο δεδομένων (*data domain*), και επιστρέφει μια λίστα από ζεύγη σε ένα διαφορετικό πεδίο**:**

$$\text{Map } (k1,v1) \rightarrow \text{list } (k2,v2)$$

Η συνάρτηση Map (απεικόνιση) εφαρμόζεται παράλληλα σε κάθε στοιχείο στο σύνολο δεδομένων εισόδου. Αυτό παράγει μια λίστα ζευγαριών (k2, v2) για κάθε κλήση. Μετά από αυτό, το MapReduce framework συλλέγει όλα τα ζευγάρια με το ίδιο κλειδί από όλες τις λίστες και τα συγκεντρώνει μαζί, δημιουργώντας έτσι μια ομάδα για κάθε ένα από τα διαφορετικά κλειδιά που δημιουργούνται.

Η συνάρτηση Reduce (συγχώνευση) εφαρμόζεται έπειτα παράλληλα σε κάθε ομάδα, η οποία με τη σειρά της παράγει μια συλλογή των τιμών στον ίδιο πεδίο (*domain*)**:**

$$\text{Reduce } (k2,\ \text{list } (v2)) \rightarrow \text{list } (v3)$$

Κάθε κλήση Reduce παράγει συνήθως είτε μια τιμή v3 είτε καμία, παρότι μία κλήση δεν επιτρέπεται να επιστρέψει περισσότερες από μία τιμές. Οι επιστροφές όλων των κλήσεων συλλέγονται σε λίστα επιθυμητού αποτελέσματος.

Έτσι το MapReduce framework μετασχηματίζει μια λίστα ζευγαριών (key, value) σε μια λίστα τιμών. Αυτή η συμπεριφορά, είναι διαφορετική από το τυπικό συναρτησιακό προγραμματισμό Map και Reduce συνδυασμού, δέχεται μια λίστα με αυθαίρετες τιμές (*arbitrary values*) και επιστρέφει μία μόνο τιμή, η οποία συνδυάζει όλες τις τιμές που επιστρέφονται από τον Map.

Είναι απαραίτητο λοιπόν, αλλά δεν αρκεί να υπάρξουν οι υλοποιήσεις των Map και Reduce αφαιρέσεων (*abstractions*) προκειμένου να υλοποιηθεί η MapReduce. Κατανεμημένες υλοποιήσεις του MapReduce απαιτούν ένα μέσο σύνδεσης στις διεργασίες που εκτελούν τα στάδια Map και Reduce. Αυτό μπορεί να είναι ένα κατανεμημένο σύστημα αρχείων, όπως το HDFS**.**

Το παρακάτω πρόβλημα παίρνει ως είσοδο διάφορα αρχεία κειμένου και μετρά τις εμφανίσεις κάθε λέξης μέσα σε μεγάλη συλλογή εγγράφων:

```
function map (String name, String document) {
 // name: document name
 // document: document contents
        for each word w in document:
           emitIntermediate (w, 1)  }
function reduce (String word, Iterator  partialCounts..) {
 // word: a word
 // partialCounts: a list of aggregated partial counts
        sum = 0
        for each pc in partialCounts:
                        sum += ParseInt(pc)
        emit (word, sum)
        }  [2]
```

Σχήμα 5.2: *Παράδειγμα Ψευδοκώδικα WordCount Εφαρμογής MapReduce (**Πηγή**: http://en.wikipedia.org/wiki/MapReduce)*

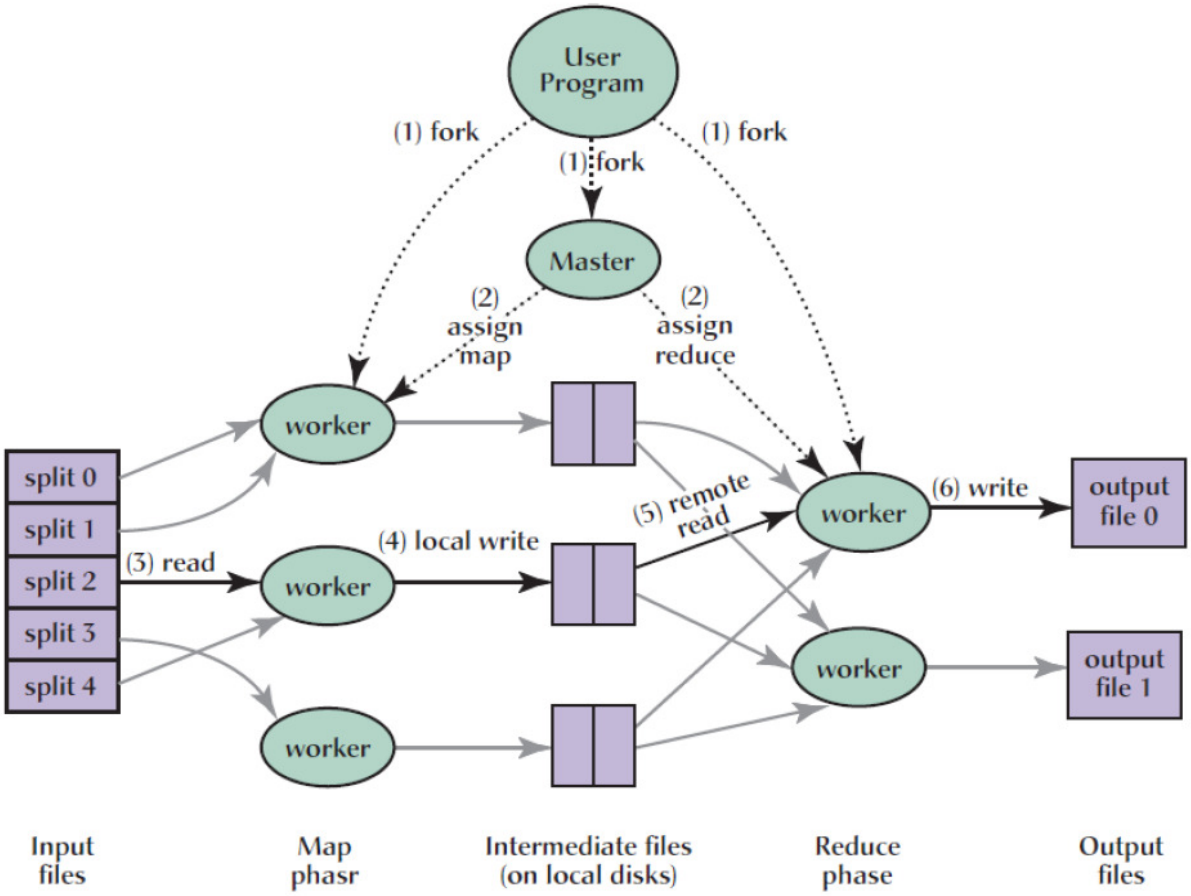
Εδώ, κάθε έγγραφο χωρίζεται σε λέξεις, και κάθε λέξη μετριέται από τη συνάρτηση map, χρησιμοποιώντας τη λέξη, ως κλειδί για το αποτέλεσμα. Το framework συγκεντρώνει όλα τα ζευγάρια με το ίδιο κλειδί και τα τροφοδοτεί στην ίδια reduce, κατά συνέπεια αυτή η συνάρτηση χρειάζεται μόνο για να αθροίσει όλες τις τιμές της εισόδου για να βρει το σύνολο των εμφανίσεων αυτής της λέξης. [3]

5.3 Γενική Επισκόπηση Εκτέλεσης MapReduce

Η κατανομή των κλήσεων Map γίνεται σε πολλαπλά μηχανήματα μέσω του αυτόματου διαχωρισμού δεδομένων εισόδου σε ένα σύνολο M διασπάσεων (*splits*). Οι διασπάσεις εισόδου μπορούν να υποστούν παράλληλη επεξεργασία σε διαφορετικές μηχανές.

Η κατανομή των κλήσεων Reduce γίνεται με κάποια συνάρτηση διαχωρισμού που χωρίζει τα ενδιάμεσα κλειδιά σε R κομμάτια. Ο αριθμός των διαχωρισμών (*partitions*) και η συνάρτηση διαχωρισμού είναι καθορισμένες από τον χρήστη.

Στο Σχήμα 5.3 παρουσιάζεται η συνολική ρέουσα κίνηση μιας MapReduce εργασίας. Όταν το πρόγραμμα του χρήστη καλεί την συνάρτηση **MapReduce**, εμφανίζεται η ακόλουθη σειρά ενεργειών (οι αριθμημένες ετικέτες στο σχήμα 5.3 αντιστοιχούν στους αριθμούς στην παρακάτω λίστα).



Σχήμα 5.3: *Στάδια εκτέλεσης μιας εργασίας MapReduce (**Πηγή**:Jeffrey Dean, Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters Commun.ACM51(1),2008)*

1. Η βιβλιοθήκη του Map/Reduce στο πρόγραμμα του χρήστη, πρώτα διασπά (*split*) τα δεδομένα εισόδου σε M κομμάτια, τυπικά μεγέθους των 64 MB ανά κομμάτι. Στη συνέχεια, ξεκινάει να τρέχει πολλαπλά αντίγραφα του προγράμματος σε ενα cluster από μηχανήματα.

2. Ένα απο τα αντίγραφα του προγράμματος είναι ιδιαίτερο και αντιστοιχεί στο master, ενώ τα υπόλοιπα είναι workers στους οποίους ανατίθενται εργασίες από τον master. Υπάρχουν M (map) και R (reduce) εργασίες για ανάθεση. Ο master επιλεγεί τους μη απασχολημένους workers και αναθέτει στον καθένα μια map ή μια reduce εργασία.

3. Ο εργαζόμενος στον οποίο έχει ανατεθεί μια map εργασία, διαβάζει το περιεχόμενο του αντίστοιχου τμήματος (*input split*) εισόδου. Αναλύει τα δεδομένα εισόδου σε ζευγάρια της μορφής κλειδί - τιμή και στέλνει το κάθε ζευγάρι στην καθορισμένη συνάρτηση Map του χρήστη. Η συνάρτηση δέχεται τα ζευγάρια αυτά και εκπέμπει

(*emit*) τα ενδιάμεσα ζευγάρια του τύπου κλειδί - τιμή, τα οποία αποθηκεύονται προσωρινά στη μνήμη.

4. Περιοδικά τα προσωρινά αποθηκευμένα ζευγάρια γράφονται στον τοπικό σκληρό δίσκο του μηχανήματος, όπου μέσω συνάρτησης διαχωρισμού (*partitioning function*) χωρίζονται σε R περιοχές. Η περιοχές αυτών των προσωρινά αποθηκευμένων ζευγαριών στον τοπικό δίσκο αποστέλλονται στον master ο οποίος είναι υπεύθυνος για την προώθηση αυτών των partitions στους workers στους οποίους έχουν ανατεθεί οι *reduce* εργασίες (*reduce workers*).
5. Όταν ένας reduce εργαζόμενος ενημερωθεί από τον master για αυτές τις περιοχές, χρησιμοποιεί απομακρυσμένες κλήσεις διαδικασιών (RPC) για να διαβάσει τα προσωρινά αποθηκευμένα δεδομένα από τους τοπικούς δίσκους των map εργαζομένων. Εφόσον διαβάσει όλα τα ενδιάμεσα δεδομένα από τη συνάρτηση διαχωρισμού, τα ταξινομεί με βάση τα ενδιάμεσα κλειδιά προκειμένου να ομαδοποιηθούν όλες οι εμφανίσεις με το ίδιο κλειδί. Η ταξινόμηση είναι αναγκαία επειδή συνήθως πολλά ζεύγη με διαφορετικά κλειδιά map κατευθύνονται στην ίδια reduce task.
6. Ο reduce εργαζόμενος εκτελεί επαναλήψεις για την ενδιάμεση ταξινομημένη λίστα για κάθε μοναδικό ενδιάμεσο κλειδί που βρίσκει, το περνάει μαζί με το αντίστοιχο ενδιάμεσο σύνολο τιμών που αντιστοιχούν σε αυτο το κλειδί στην Reduce συνάρτηση του χρήστη. Η έξοδος της Reduce συνάρτησης επισυνάπτεται σε ένα τελικό αρχείο εξόδου.
7. Αφού όλες οι εργασίες map και reduce έχουν ολοκληρωθεί, ο master επιστρέφει τον έλεγχο της εκτέλεσης στο πρόγραμμα του χρήστη. Σε αυτό το σημείο ολοκληρώνεται η κλήση *MapReduce*.

Μετά την επιτυχή ολοκλήρωση, η έξοδος από την MapReduce εκτέλεση είναι διαθέσιμη μέσω των R αρχείων εξόδου ένα για κάθε reduce εργασία, με ονόματα που προσδιορίζονται από τον χρήστη.

Για τον εντοπισμό της αποτυχίας, ο master ελέγχει περιοδικά την παρουσία του κάθε εργαζόμενου. Εάν καμία ανταπόκριση δεν ληφθεί από τον εργαζόμενο σε ένα ορισμένο συνολικό χρόνο, ο master θεωρεί τον εργαζόμενο ότι απέτυχε. Κάθε ολοκληρωμένη map εργασία από τους εργαζόμενους επαναφέρετε στην αρχική ανενεργή κατάσταση (*idle state*), και συνεπώς καθίστανται επιλέξιμες για χρονοπρογραμματισμό στους άλλους εργαζόμενους. Ομοίως, κάθε map ή reduce εργασία ενός αποτυχημένου εργαζόμενου επαναφέρεται στην

αρχική ανενεργή κατάσταση και γίνεται εκλέξιμο για προγραμματισμό. Οι ολοκληρωμένες map εργασίες επανεκτελούνται (hadoop v0.21) όταν η αποτυχία συμβαίνει επειδή οι έξοδοί τους αποθηκεύονται στον τοπικό δίσκο του αποτυχημένου μηχανήματος και συνεπώς δεν είναι προσπελάσιμη. Οι ολοκληρωμένες reduce εργασίες δεν χρειάζεται να επανεκτελεστούν από τη στιγμή που είναι αποθηκευμένες σε ένα συνολικό σύστημα αρχείων. [4]

Η ροή δεδομένων που ακολουθείται κατά την εκτέλεση μιας MapReduce εργασίας μπορεί να αποτυπωθεί ως εξής**:**

(input) < k1, v1 > → Map → list< k2, v2 > → shufflers & combiners → <k2, list (v2) > → Reduce → list< k3, v3 > (output)

Όπου τα k1...3 και v1...3 αναπαριστούν τον τύπο των keys και values.

5.4 Ο **Χ**ωρισμός της **Ε**ργασίας σε **Μ**aps και **R**educes

Η επιλογή του κατάλληλου μεγέθους για τα *tasks* της εργασίας (*job*) μπορεί να αλλάξει ριζικά την απόδοση του Hadoop. Η αύξηση του αριθμού των εργασιών (*tasks*) αυξάνει την επιβάρυνση του πλαισίου εργασίας (*framework overhead*), αλλά αυξάνει και την εξισορρόπηση του φόρτου (*load balancing*), ενώ μειώνει το κόστος των αποτυχιών.

5.4.1 **Α**ριθμός των **Μ**aps

Ο αριθμός των Maps καθοδηγείται συνήθως από τον αριθμό των blocks DFS στα αρχεία εισόδου. Επίσης αυτό οδηγεί τους χρήστες να αλλάξουν το μέγεθος των block DFS ούτως ώστε να ρυθμίσουν τον αριθμό των maps.

Το σωστό επίπεδο παραλληλισμού για τους Maps έχει βρεθεί να είναι περίπου από 10**-**100 Maps **/** node.

Στην πραγματικότητα ο έλεγχος του αριθμού των Maps είναι ευφυής. Η παράμετρος mapred.map.tasks είναι μόνο μια υπόδειξη της InputFormat για τον αριθμό των Maps. Η προεπιλεγμένη συμπεριφορά του InputFormat είναι να χωρίσει το συνολικό αριθμό των bytes σε σωστό αριθμό κομματιών (*fragments*). Ωστόσο, στην περίπτωση προεπιλογής το μέγεθος block DFS των αρχείων εισόδου θεωρείται ως το ανώτατο όριο (*upper bound*) για την είσοδο. Ένα κατώτερο όριο για τη διάσπαση μεγέθους (*lower bound*) μπορεί να ρυθμιστεί μέσω του mapred.min.split.size.

Κατά συνέπεια, εάν αναμένουμε 10TB δεδομένων εισόδου, και έχοντας ένα dfs block size των 128MB, θα καταλήξουμε με 82k Maps (*82,000 maps*), εκτός αν το mapred.map.tasks είναι ακόμα μεγαλύτερο. Τελικά το InputFormat καθορίζει τον αριθμό Maps.

Ο αριθμός των Map εργασιών (*tasks*) μπορεί επίσης να αυξηθεί χειροκίνητα χρησιμοποιώντας το conf.setNumMapTasks(int num) του JobConf. Αυτό μπορεί να χρησιμοποιηθεί για να αυξηθεί ο αριθμός των Map εργασιών (*tasks*), αλλά χωρίς θέσει τον αριθμό χαμηλότερο από εκείνο που καθορίζει το Hadoop, μέσω της διάσπασης των δεδομένων εισόδου**.**

5.4.2 **A**ριθμός των **R**educers

Ο σωστός αριθμός reduces θεωρείτε να είναι **0.95** ή **1.75** * (nodes * mapred.tasktracker.tasks.maximum). Για **0,95** όλα τα reduces μπορούν να ξεκινήσουν αμέσως μεταφέροντας τα αποτελέσματα Map ως τον τερματισμό των Maps. Στο **1.75** οι γρηγορότεροι κόμβοι θα τελειώσουν τον πρώτο γύρο των reduces και θα ξεκινήσει ένας δεύτερος γύρος των reduces που κάνει μια πολύ καλύτερη εργασία (*job*) της εξισορρόπησης φόρτου.

Σήμερα ο αριθμός των reduces περιορίζεται σε περίπου 1000 από το μέγεθος του buffer για τα αρχεία εξόδου (io.buffer.size * 2 * << numReduces heapSize). Αυτό μπορεί να καθοριστεί έως κάποιο βαθμό, μέχρι να παρέχει ένα άνω όριο αρκετά σταθερό.

Αύξηση του αριθμού των reduces αυξάνεται το framework overhead , αλλά αυξάνεται και η εξισορρόπηση φόρτου και μειώνεται το κόστος των αποτυχιών.

Ο αριθμός reduce εργασιών (*reduce tasks*) μπορεί επίσης να αυξηθεί με τον ίδιο τρόπο όπως οι map tasks, μέσω conf.setNumReduceTasks(int num) του JobConf**.** [5]

5.5 **M**ap**R**educe **Π**αραδείγματα

Εδώ παρατίθενται μερικά απλά παραδείγματα των ενδιαφερόντων προγραμμάτων που μπορούν να εκφραστούν εύκολα ως υπολογισμοί MapReduce.

- **Distributed Grep:**

Η συνάρτηση map εκπέμπει μια γραμμή (line), εάν ταιριάζει με ένα συγκεκριμένο πρότυπο (pattern) από εκείνα που παρέχονται, και τότε δίνει στην έξοδο την συγκεκριμένη γραμμή.

Η συνάρτηση reduce είναι μια συνάρτηση ταυτοποίησης (*identity function*), που αντιγράφει τα ενδιάμεσα δεδομένα που παρέχονται στην έξοδο της map.

- **Count of URL Access Frequency:**

  Η συνάρτηση map επεξεργάζεται τα αρχεία ιστορικού (*logs*) από web page αιτήσεις, και για κάθε πρόσβαση δίνει έξοδο <URL, 1>.

  Η συνάρτηση Reduce προσθέτει όλες τις τιμές για την ίδια διεύθυνση URL, και εκπέμπει ένα ζεύγος <URL, total count>

- **Reverse Web-Link Graph:**

  Η συνάρτηση Map εξάγει ζεύγη <target, source> για κάθε σύνδεσμο (*link*) σε ένα στόχο (target) URL που βρίσκεται σε μια σελίδα ονομαζόμενη πηγή "source".

  Οι reduce συνάρτηση συνενώνει τη λίστα όλων των διευθύνσεων πηγής URL (*source URLs*), που σχετίζονται με τη συγκεκριμένη διεύθυνση URL προορισμού και εκπέμπει το ζεύγος <target, list(source)>.

- **Term-Vector per Host:**

  Ένας όρος διάνυσμα συνοψίζει τις πιο σημαντικές λέξεις που συμβαίνουν σε ένα έγγραφο (*document*) ή ένα σύνολο εγγράφων, όπως μια λίστα από <word, frequency> ζεύγη.

  Η συνάρτηση Map εκπέμπει ένα ζεύγος <hostname, term vector> για κάθε έγγραφο εισόδου (όπου το όνομα εξάγεται από τη διεύθυνση URL του εγγράφου).

  Η συνάρτηση reduce περνάει από όλα τα per-document διανύσματα όρων μαζί (term vectors), πετάει τους σπάνιους όρους, και στη συνέχεια εκπέμπει ένα τελικό ζεύγος <hostname, term vector>

- **Inverted Index:**

  Η συνάρτηση Map αναλύει κάθε έγγραφο, και εκπέμπει μια ακολουθία από ζευγάρια <word, document ID>.

Η συνάρτηση reduce δέχεται όλα τα ζεύγη για μια συγκεκριμένη λέξη, ταξινομεί τις αντίστοιχες ταυτότητες των εγγράφων (document IDs) και εκπέμπει ένα ζεύγος <word, list(document ID)>.

Το σύνολο όλων των ζευγών εξόδου αποτελεί ένα απλό αντεστραμμένο ευρετήριο.

- **Distributed Sort:**

Η συνάρτηση Map για κάθε εγγραφή εξάγει το κλειδί (key) και εκπέμπει ένα <key,record>. Η συνάρτηση reduce δίνει στην έξοδο όλα τα ζεύγη που πήρε σαν είσοδο. [6]

5.6 Ανοχή σε Σφάλματα (Fault Tolerance)

Ένας από τους κύριους λόγους για να χρησιμοποιήσετε Hadoop, ώστε να εκτελέσετε τις εργασίες (*jobs*) σας, είναι λόγω του υψηλού βαθμού ανοχής του σε σφάλματα. Ακόμα και όταν τρέχει εργασίες (*jobs*) σε ένα μεγάλο cluster, όπου μεμονωμένοι κόμβοι ή στοιχεία του δικτύου μπορεί να εμφανίσουν υψηλά ποσοστά αποτυχίας, το Hadoop μπορεί να καθοδηγήσει τις εργασίες προς μια επιτυχή ολοκλήρωση.

Ο πρωταρχικός τρόπος του Hadoop επιτυγχάνει ανοχή σφαλμάτων, μέσω επανεκκίνησης εργασιών (*tasks*). Μεμονωμένοι κόμβοι εργασιών *(TaskTrackers)* βρίσκονται σε συνεχή επικοινωνία με τον επικεφαλή κόμβο του συστήματος, που ονομάζεται *JobTracker*. Εάν ένας TaskTracker αποτύχει να επικοινωνήσει με την JobTracker για ένα χρονικό διάστημα (από προεπιλογή, 1 λεπτό), ο JobTracker θα θεωρήσει ότι το εν λόγω TaskTracker έχει καταστραφεί. Ο JobTracker επίσης, γνωρίζει ποια map και reduce εργασίες είχαν ανατεθεί σε κάθε TaskTracker**.**

Αν η εργασία (*job*) είναι ακόμη στη φάση του mapping, τότε άλλα TaskTrackers θα κληθούν εκ νέου για να εκτελέσουν όλες τις map εργασίες (*tasks*) που προηγουμένως έτρεχαν στον αποτυχημένο TaskTracker**.** Εάν η εργασία (*job*) είναι στη reduce φάση, τότε άλλοι TaskTrackers θα επαν-εκτελέσουν όλες τις reduce εργασίες (*tasks*) που ήταν σε εξέλιξη για το αποτυχημένο TaskTracker.

Οι reduce εργασίες (*tasks*), μόλις ολοκληρωθούν, γράφονται στο HDFS. Έτσι, αν ένας TaskTracker έχει ήδη ολοκληρώσει δύο από τις τρεις reduce εργασίες (*tasks*) που του έχουν ανατεθεί, μόνο η τρίτη εργασία (*task*) πρέπει να εκτελεστεί αλλού. Οι map εργασίες είναι λίγο πιο περίπλοκες: αν ένας κόμβος έχει συμπληρώσει δέκα map εργασίες, οι reducers

μπορεί να μην έχουν αντιγράψει τις εισόδους τους από την έξοδο αυτών των map εργασιών. Αν ένας κόμβος έχει καταρρεύσει, τότε η κάθε map έξοδος είναι μη προσβάσιμο . Έτσι, τυχόν ήδη ολοκληρωμένες map εργασίες πρέπει να επαν-εκτελεστούν εκ νέου για να κάνουν τα αποτελέσματά τους διαθέσιμα στα υπόλοιπα reduce μηχανήματα. Όλο αυτό γίνονται αυτόματα από την πλατφόρμα Hadoop.

Αυτή η ανοχή σφαλμάτων δίνει έμφαση στην ανάγκη για την εκτέλεση του προγράμματος ώστε να είναι δωρεάν παρενέργεια. Αν οι Mappers και Reducers έχουν μεμονωμένες ταυτότητες και επικοινωνούσαν μεταξύ τους ή τον έξω κόσμο, τότε η επανεκκίνηση εργασίας θα απαιτήσει από τους άλλους κόμβους να επικοινωνούν με τα νέα στοιχεία του map και reduce εργασιών (*tasks*), και οι επαν-εκτελεσμένες εργασίες θα πρέπει να αποκαταστήσουν την ενδιάμεση κατάστασή τους. Αυτή η διαδικασία είναι εμφανώς πολύπλοκη και επιρρεπής στα λάθη σε γενική περίπτωση. Το MapReduce απλοποιεί δραστικά αυτό το πρόβλημα με την εξάλειψη των ταυτοτήτων εργασίας (*task*) ή την ικανότητα για τις διαχωρισμένες εργασίες (*task*) για να επικοινωνούν η μια με την άλλη. Μια μεμονωμένη εργασία (*task*) βλέπει μόνο τις δικές της άμεσες εισόδους και γνωρίζει μόνο τις δικές τις εξόδους, για να κάνει αυτή την αποτυχία και την διαδικασία επανεκκίνησης καθαρή και αξιόπιστη.

5.6.1 **Κ**ερδοσκοπική **Ε**κτέλεση (Speculative execution)

Ένα πρόβλημα με το σύστημα Hadoop είναι ότι διαιρώντας τις εργασίες (tasks) σε πολλούς κόμβους, είναι δυνατό για μερικούς αργούς κόμβους να έχουμε περιορισμένο ρυθμό για το υπόλοιπο του προγράμματος. Για παράδειγμα, εάν ένας κόμβος έχει έναν αργό ελεγκτή δίσκου (*disk controller*), τότε μπορεί να διαβάζει την είσοδό της μόνο κατά 10% της ταχύτητας όλων των άλλων κόμβων. Οπότε, όταν 99 map tasks έχουν ήδη ολοκληρωθεί, το σύστημα είναι ακόμα σε αναμονή για την τελική map εργασία για να κάνει check-in, το οποίο διαρκεί πολύ περισσότερο από όλους τους άλλους κόμβους.

Αναγκάζοντας τις εργασίες να εκτελούνται απομονωμένα το ένα από το άλλο, οι επιμέρους εργασίες δεν ξέρουν από πού προέρχονται οι είσοδοί τους. Οι εργασίες (*tasks*) εμπιστεύονται την πλατφόρμα Hadoop για να παραδώσει ακριβώς την κατάλληλη είσοδο. Ως εκ τούτου, η ίδια είσοδος μπορεί να υποβληθεί σε *επεξεργασία πολλές φορές παράλληλα*, για να αξιοποιήσουν τις διάφορες δυνατότητες του μηχανήματος. Δεδομένου ότι οι περισσότερες από τις εργασίες (*tasks*) σε μια εργασία (*job*) φτάνουν στο τέλος, η πλατφόρμα του Hadoop θα προγραμματίσει τα περιττά αντίγραφα από τις υπόλοιπες εργασίες σε αρκετούς κόμβους που δεν έχουν άλλη εργασία για να εκτελέσουν. Αυτή η διαδικασία είναι γνωστή ως

κερδοσκοπική εκτέλεση. Όταν οι εργασίες ολοκληρωθούν, θα ανακοινώσουν το γεγονός αυτό στον JobTracker. Όποιο αντίγραφο εργασίας τερματίσει πρώτο γίνεται ένα οριστικό αντίγραφο. Εάν άλλα αντίγραφα εκτελούνται κερδοσκοπικά, το Hadoop λέει στους TaskTrackers να εγκαταλείψουν τις εργασίες και απορρίψουν τις εξόδους τους. Οι Reducers τότε λαμβάνουν τις εισόδους τους από οποιοδήποτε Mapper που ολοκληρώθηκε πρώτο με επιτυχία.

Η κερδοσκοπική εκτέλεση είναι ενεργοποιημένη από προεπιλογή. Μπορείτε να απενεργοποιήσετε την κερδοσκοπική εκτέλεση για τους mappers και τους reducers με δύο τρόπους θέτοντας τις ρυθμίσεις, στο JobConf.set ή σε mapred-site.xml τους mapred.map.tasks.speculative.execution και mapred.reduce.tasks.speculative.execution αντίστοιχα, σε false. [7]

5.7 **M**ap**R**educe Σε **Σ**χέση Με Το **Σ**ύστημα **Δ**ιαχείρισης **Σ**χεσιακών **Β**άσεων **Δ**εδομένων (RDBMS)

Γιατί να μην μπορούμε να χρησιμοποιήσουμε τις βάσεις δεδομένων πολλούς δίσκους ώστε να κάνουμε μεγάλης κλίμακας σειρά ανάλυσης**;** Γιατί είναι το MapReduce χρήσιμο;

Η απάντηση σε αυτά τα ερωτήματα έρχεται από μια άλλη τάση σε δίσκους: ο χρόνος αναζήτησης βελτιώνεται πιο αργά από ό, τι ταχύτητα μεταφοράς. Η αναζήτηση είναι η διαδικασία στο να μεταβεί η κεφαλή του δίσκου σε ένα συγκεκριμένο σημείο στο δίσκο να διαβάσει ή να γράψει δεδομένα. Αυτό χαρακτηρίζει την καθυστέρηση της λειτουργίας του δίσκου, ενώ ο ρυθμός μεταφοράς αντιστοιχεί στο εύρος ζώνης ενός δίσκου.

Εάν το σχέδιο πρόσβασης δεδομένων κυριαρχήσει από τις αναζητήσεις, θα χρειαστεί περισσότερο χρόνο για να διαβάσει ή να γράψει μεγάλα τμήματα του συνόλου δεδομένων από το να υπάρχει μια συνεχής ροής μέσα από αυτό, το οποίο λειτουργεί στο ρυθμό μεταφοράς. Από την άλλη πλευρά, για την ενημέρωση ενός μικρού ποσοστού των εγγραφών σε μια βάση δεδομένων, ένα παραδοσιακό B-Tree *(η δομή δεδομένων που χρησιμοποιούνται σε σχεσιακές βάσεις δεδομένων, η οποία περιορίζεται από το ρυθμό που μπορεί να εκτελέσει αναζητήσεις)* λειτουργεί καλά. Για την ενημέρωση της πλειοψηφίας της βάσης δεδομένων, ένα B-Tree είναι λιγότερο αποτελεσματική από το MapReduce, το οποίο χρησιμοποιεί Ταξινόμηση / Συγχώνευση για την ανοικοδόμηση της βάσης δεδομένων.

Με πολλούς τρόπους, το MapReduce μπορεί να θεωρηθεί ως συμπλήρωμα μιας RDBMS. (Οι διαφορές μεταξύ των δύο συστημάτων φαίνεται στο Σχήμα 5.4). Το

MapReduce είναι ένα κατάλληλο αγαθό για τα προβλήματα τα οποία πρέπει να αναλύσει ένα μεγάλο σύνολο δεδομένων (*dataset*), σε μορφή batch, ιδίως για ad hoc ανάλυση. Ένα RDBMS είναι καλή για τα σημειακά ερωτήματα (*point queries*) ή για ενημερώσεις, όπου το σύνολο δεδομένων έχει ευρετήριο για να προσφέρει χαμηλό latency ανάκτησης και τους χρόνους ενημέρωσης του ένός σχετικά μικρού ποσοστού των δεδομένων. Το MapReduce ταιριάζει σε εφαρμογές, όπου τα δεδομένα είναι γραμμένα μια φορά, και διαβάζονται πολλές φορές, ενώ μια σχεσιακή βάση δεδομένων είναι καλή για τα σύνολα δεδομένων που ενημερώνονται συνεχώς.

	Traditional RDBMS	MapReduce
Data size	Gigabytes	Petabytes
Access	Interactive and batch	Batch
Updates	Read and write many times	Write once, read many times
Structure	Static schema	Dynamic schema
Integrity	High	Low
Scaling	Nonlinear	Linear

Σχήμα 5.4: *Διαφορές μεταξύ κλασικών σχεσιακών Βάσεων Δεδομένων με το MapReduce*

Μια άλλη διαφορά μεταξύ ενός MapReduce και ενός RDBMS είναι η ποσότητα της δομής του συνόλου των δεδομένων που χειρίζονται. Δομημένα δεδομένα είναι τα δεδομένα που είναι οργανωμένα σε οντότητες και που έχουν καθορισμένη μορφή, όπως XML έγγραφα ή πίνακες της βάσης δεδομένων, που προσαρμόζονται σε ένα συγκεκριμένο προκαθορισμένο σχήμα. Τα ημι-δομημένα δεδομένα, από την άλλη πλευρά, είναι ασαφής, αν και μπορεί να υπάρχει ένα σχήμα, το οποίο συχνά αγνοείται, έτσι ώστε να μπορεί να χρησιμοποιηθεί μόνο ως οδηγός για τη δομή των στοιχείων: για παράδειγμα, ένα λογιστικό φύλλο, στο οποίο η δομή είναι ένα πλέγμα με κελιά, όπως και τα ίδια τα κελιά μπορεί να κατέχει οποιαδήποτε μορφή των δεδομένων. Τα μη δομημένα δεδομένα δεν έχουν καμία ιδιαίτερη εσωτερική δομή: για παράδειγμα, απλό κείμενο ή εικόνες. Το MapReduce λειτουργεί καλά για τα μη δομημένα ή ημι-δομημένα δεδομένα, δεδομένου ότι έχει σχεδιαστεί για να διερμηνεύει τα δεδομένα κατά το χρόνο επεξεργασίας. Με άλλα λόγια, τα κλειδιά εισόδου και οι τιμές για το

MapReduce δεν είναι μια εσωτερική ιδιότητα των δεδομένων, αλλά επιλέγονται από το άτομο που αναλύει τα δεδομένα.

Τα σχεσιακά δεδομένα συχνά *κανονικοποιούνται* για να διατηρήσουν την ακεραιότητά τους και αφαιρέσουν τον πελονασμό. Η κανονικοποίηση δημιουργεί προβλήματα για MapReduce, δεδομένου ότι κάνει ανάγνωση ένα αρχείο με μια μη- τοπική λειτουργία, και μία από τις κεντρικές παραδοχές που το MapReduce κάνει είναι ότι είναι δυνατό να εκτελέσει (υψηλής ταχύτητας) συνεχούς ροής (*streaming*) ανάγνωση και εγγραφή.

Ένα αρχείο καταγραφής ενός web server, είναι ένα καλό παράδειγμα για ένα σύνολο εγγραφών που δεν κανονικοποιείται (για παράδειγμα, τα ονόματα κεντρικών υπολογιστών πελατών που ορίζονται χωρίς περικοπές κάθε φορά, έστω και αν ο ίδιος πελάτης μπορεί να εμφανιστεί πολλές φορές), και αυτός είναι ο λόγος για τον οποίο όλα τα είδη αρχείων καταγραφής είναι ιδιαίτερα καλά προσαρμοσμένα για την ανάλυση με MapReduce.

Το MapReduce είναι ένα γραμμικά κλιμακωτό μοντέλο προγραμματισμού. Ο προγραμματιστής γράφει δύο λειτουργίες - μια συνάρτηση *map* και μια συνάρτηση Reduce – όπου το καθένα ορίζει ένα mapping από ένα σύνολο ζευγαριών κλειδί-τιμή σε ένα άλλο. Αυτές οι λειτουργίες αγνοούν το μέγεθος των δεδομένων ή το cluster που χειρίζονται, έτσι ώστε να μπορούν να χρησιμοποιηθούν αμετάβλητες, για ένα μικρό σύνολο δεδομένων, όπως και για ένα μεγάλο. Πιο σημαντικό, αν διπλασιάσεις το μέγεθος των δεδομένων εισόδου, μια εργασία θα εκτελείται δύο φορές πιο αργά. Αλλά αν διπλασιάσεις το μέγεθος του cluster, μια εργασία θα τρέξει τόσο γρήγορα όσο το αρχικό. Γενικά αυτό δεν ισχύει για τα ερωτήματα SQL.

Με τον καιρό, όμως, οι διαφορές μεταξύ των σχεσιακών βάσεων δεδομένων και MapReduce συστημάτων, είναι πιθανόν να μειωθούν όσο οι σχεσιακές βάσεις δεδομένων ξεκινήσουν να ενσωματώνουν μερικές από τις ιδέες του MapReduce (όπως Aster Data, και βάσεις δεδομένων Greenplum) και από την άλλη κατεύθυνση, όσο χτίζονται γλώσσες υψηλού επιπέδου βασισμένες στο MapReduce (όπως το Pig και το Hive) κάνουν τα συστήματα MapReduce πιο προσιτά στους παραδοσιακούς προγραμματιστές βάσεων δεδομένων. [8]

ΑΝΑΦΟΡΕΣ

[1].http://hadoop.apache.org/docs/r0.18.3/mapred_tutorial.html#Map%2FReduce+-+User+Interfaces

[2]. http://www.admin-magazine.com/HPC/Articles/MapReduce-and-Hadoop

[3]. MapReduce **-** http://en.wikipedia.org/wiki/MapReduce

[4]. Jeffrey Dean, Sanjay Ghemawat: *MapReduce: Simplified Data Processing on Large Clusters* Commun.ACM51(1),2008

[5]. Hadoop Wiki - http://wiki.apache.org/hadoop/HowManyMapsAndReduces

[6]. Jeffrey Dean, Sanjay Ghemawat: *MapReduce: Simplified Data Processing on Large Clusters* Commun.ACM51(1),2008

[7]. http://developer.yahoo.com/hadoop/tutorial/module4.html#tolerence

[8]. Tom White, (2010): *Hadoop: The Definitive Guide*, 2nd Edition. O'Reilly Media/Yahoo Press, (σελ. 28)

ΚΕΦΑΛΑΙΟ 6ο

ΠΕΙΡΑΜΑΤΙΚΟΣ ΣΧΕΔΙΑΣΜΟΣ

6.1 Περιβάλλον Λειτουργίας και Δοκιμών

Το περιβάλλον εργασίας που επιλέξαμε ήταν του Ubuntu Linux Desktop (10.04).

Η εγκατάσταση μηχανημάτων πραγματοποιηθηκε μέσω Oracle VM VirtualBox (4.2.12) που τρέχει εικονικές μηχανές πάνω σε Windows 7. Κατεβάζουμε την σωστή έκδοση για το λειτουργικό μας από εδώ: https://www.virtualbox.org/wiki/Downloads

Για εγκατάσταση του κύριου κόμβου του master, από τον wizard επιλέγουμε New Machine → master-Linux-Ubuntu. Του δίνουμε 2 GB RAM και στην επόμενη οθόνη επιλέγουμε Startup Disk και create new hard disk (10GB), τύπο VDI, και dynamically allocated χώρο. Σε αυτό τον κόμβο εγκαταστάθηκε η έκδοση 0.20.2 του Hadoop, για λεπτομέρειες που αφορούν τις εγκαταστάσεις αναφέρονται στο Παράρτημα "Α".

Για τα υπόλοιπα μηχανήματα όπως φαίνονται στον πίνακα 6.1 χρησιμοποιήσαμε το master που είχαμε ήδη εγκατεστημένη την έκδοση του Hadoop 0.20.2. Αυτό που έγινε είναι να δημιουργήσουμε έναν κλώνο του master κόμβου και έπειτα να το διαμορφώσουμε με τα παρακάτω χαρακτηριστικά έτσι ώστε να μην απαιτηθεί ξανά η εγκατάσταση του Hadoop 0.20.2 απ'την αρχή.

Η παρεχόμενη RAM και ο ελεύθερος χώρος μπορούν αργότερα να αλλάξουν εάν ο χρήστης το επιθυμεί.

Κατόπιν κατεβάζουμε το Ubuntu Linux Desktop 10.04. (~699MB) .

http://old-releases.ubuntu.com/releases/10.04.0/

Τύπος φυσικού μηχανήματος (Laptop) που χρησιμοποιήθηκε προκειμένου να «στηθούν» τα VMs έχει τις εξής προδιαγραφές: *Intel Core i5-3230M/4GB/750GB CPU @ 2,60GHZ*, ενώ τα χαρακτηριστικά που χρησιμοποιήθηκαν για την πραγματοποίηση των πειραμάτων για το Hadoop φαίνονται στον πίνακα 6.1.

Πίνακας 6.1: Προδιαγραφές μηχανημάτων στο cluster

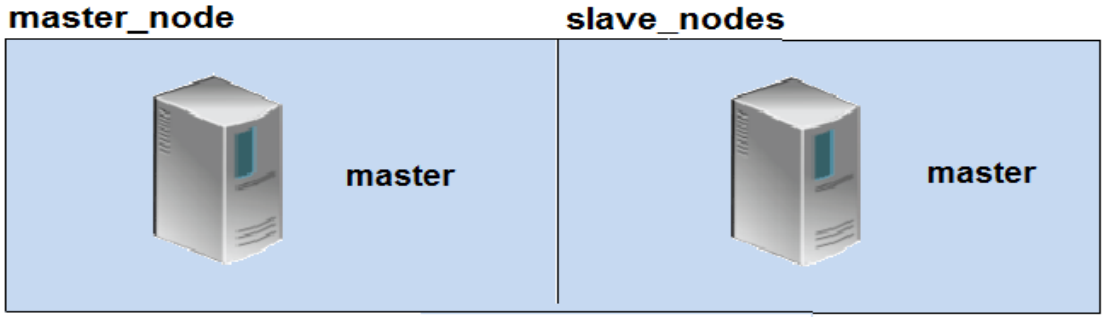
	Master/slave	**slave1**	**slave2**	**slave3**
Μνήμη RAM εικονικής μηχανής	2 GB	600 MB	600 MB	**600 MB**
Πλήθος CPU εικονικής μηχανής	3	1	1	**1**

6.2 Εκτέλεση Πειραμάτων

Αρχικά η διεξαγωγή των πειραμάτων και στα τέσσερα στάδια πραγματοποιήθηκε πανω σε Cluster των τεσσάρων κόμβων που υποστήριζαν το Hadoop, σε περιβάλλον Ubuntu_Linux. Το καθένα απο αυτά περιγράφεται αναλυτικά με εικόνες.

Στο **1ο** στάδιο της πειραματικής διαδικασίας έγινε σε ένα μηχάνημα. Τα μηχανήματα που επιλέχθηκαν ως master και ως slave φαίνονται στο σχήμα 6.1.

Το μηχάνημα master όπως βλέπουμε και στο παρακάτω σχήμα 6.1. λειτουργεί και σαν master και σαν slave.

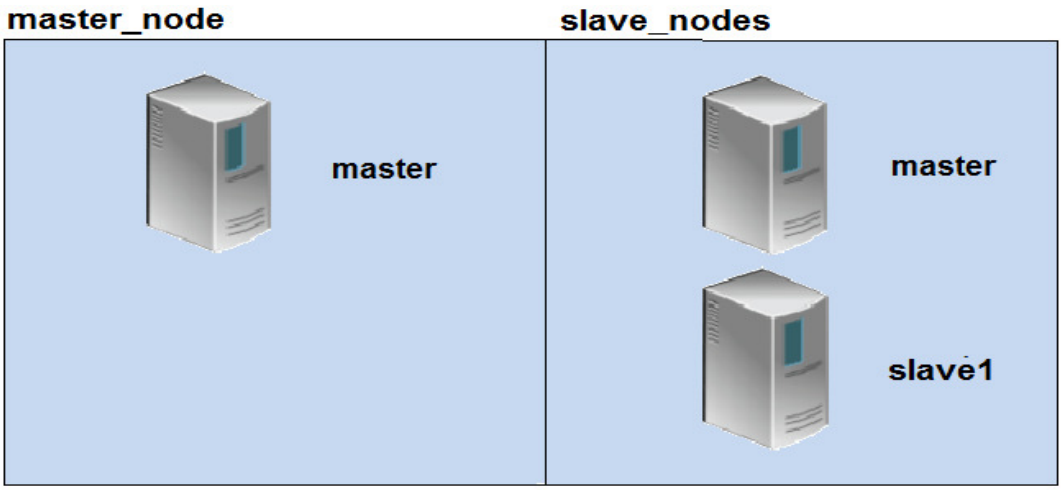


Σχήμα 6.1: *Επιλογή ενός master_node και ενός slave_node*

Στο **2ο** στάδιο της εκτέλεσης του πειράματος έγινε σε δύο μηχανήματα.

Ύπαρξη ενος master_node και δύο slave_nodes.

Τα μηχανήματα που επιλέχθηκαν ως master και ως slave φαίνονται στο σχήμα 6.2.

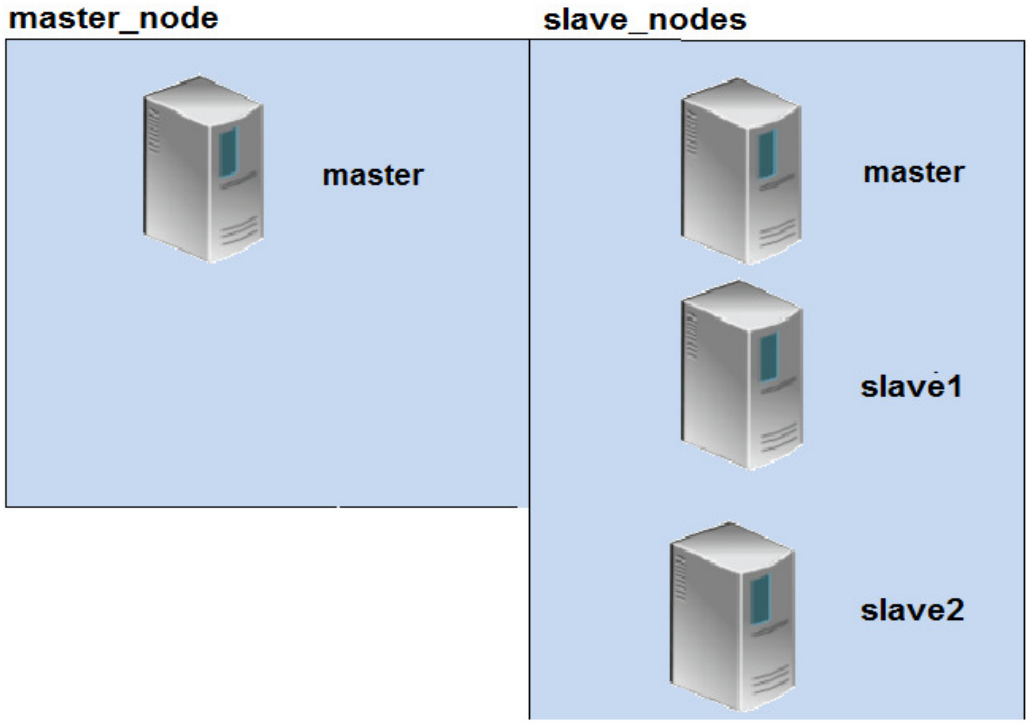


Σχήμα 6.2: *Επιλογή ενός master_node και δύο slave_nodes*

Στο **3ο** στάδιο της εκτέλεσης του πειράματος έγινε σε τρία μηχανήματα.

Ύπαρξη ενος master_node και τριών slave_nodes.

Τα μηχανήματα που επιλέχθηκαν ως master και ως slave φαίνονται στο σχήμα 6.3

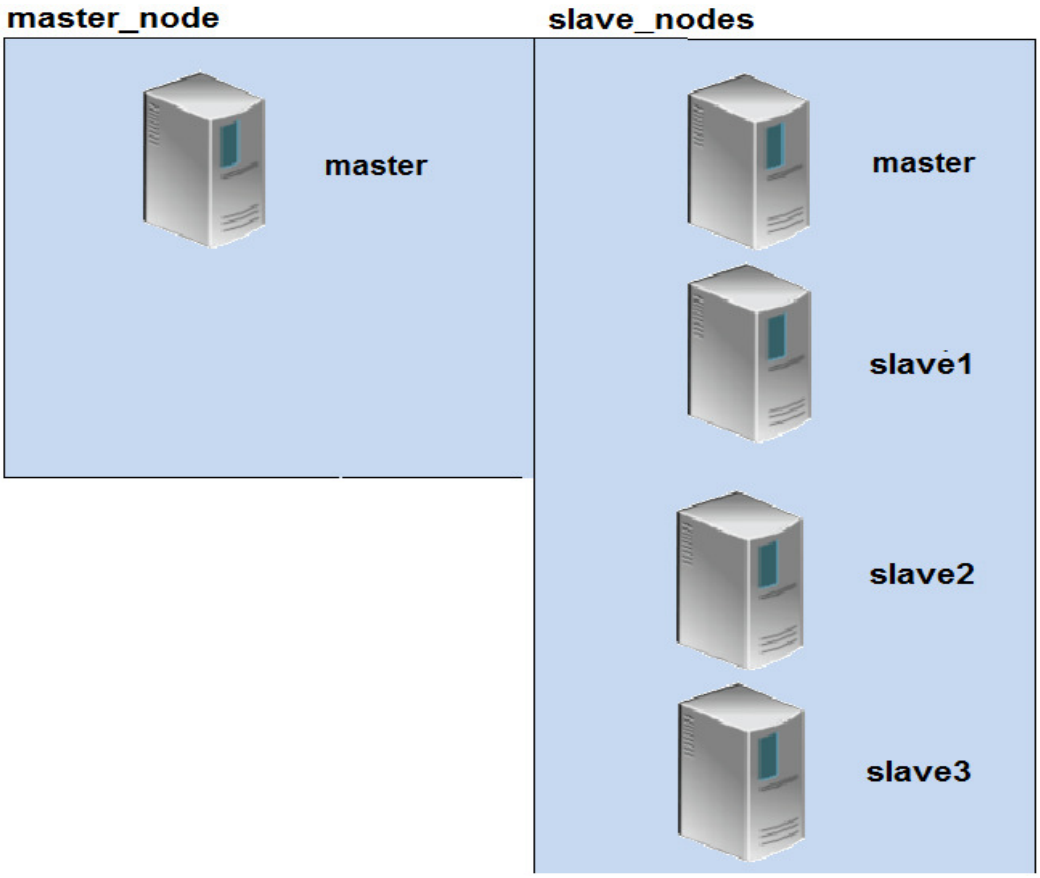


Σχήμα 6.3: *Επιλογή ενός master_node και τριών slave_nodes*

Στο **4ο** στάδιο της εκτέλεσης του πειράματος έγινε σε τέσσερα μηχανήματα.

Ύπαρξη ενος master_node και τεσσάρων slave_nodes.

Τα μηχανήματα που επιλέχθηκαν ως master και ως slave φαίνονται στο σχήμα 6.4.



Σχήμα 6.4: *Επιλογή ενός master_node και τεσσάρων slave_nodes*

6.3 Αποτελέσματα Εκτέλεσης Πειραμάτων

ΑΠΟΤΕΛΕΣΜΑΤΑ 1ου ΠΕΙΡΑΜΑΤΟΣ

Πίνακας 6.2: *1ο στάδιο εκτέλεσης Title_Finder (min)*

1 Μηχάνημα	**Job**	**Είσοδος (source code of urls)**	**Χρόνοι εκτέλεσης (min)**
	Title_Finder	100	2.95
		500	13.10
		1000	25.75
		3000	1h.25 min
2 Μηχανήματα	**Job**	**Είσοδος (source code of urls)**	**Χρόνοι εκτέλεσης (min)**
	Title_Finder	100	1,80
		500	6,29
		1000	17,20
		3000	37
3 Μηχανήματα	**Job**	**Είσοδος (source code of urls)**	**Χρόνοι εκτέλεσης (min)**
	Title_Finder	100	1,45
		500	5,89
		1000	13
		3000	30
4 Μηχανήματα	**Job**	**Είσοδος (source code of urls)**	**Χρόνοι εκτέλεσης (min)**
	Title_Finder	100	
		500	
		1000	
		3000	

Πίνακας 6.3: *2ο στάδιο εκτέλεσης Word_Finder*

1 Μηχάνημα	**Job**	**Είσοδος (source code of urls)**	**Χρόνοι εκτέλεσης (min)**
	Word_Finder	100	4.75
		500	20.5
		1000	59
		3000	2h.54min
2 Μηχανήματα	**Job**	**Είσοδος (source code of urls)**	**Χρόνοι εκτέλεσης (min)**
	Word_Finder	100	4,20
		500	8,85
		1000	24
		3000	1h.15min
3 Μηχανήματα	**Job**	**Είσοδος (source code of urls)**	**Χρόνοι εκτέλεσης (min)**
	Word_Finder	100	3,05
		500	8,50
		1000	18
		3000	52
4 Μηχανήματα	**Job**	**Είσοδος (source code of urls)**	**Χρόνοι εκτέλεσης (min)**
	Word_Finder	100	
		500	
		1000	
		3000	

Για την πειραματική αξιολόγηση του συστήματος έγιναν δύο υπολογισμοί (αναζήτησης) που εκτελέστηκαν σε ένα μικρό cluster μηχανών.

Στους πίνακες του παραπάνω παραδείγματος παρατίθενται τα αποτελέσματα που λάβαμε, τρέχοντας το πρόγραμμα με χρήση αρχείου (fileName).

Κατά την εκτέλεση των πειραμάτων στο $1^{ο}$ και $2^{ο}$ στάδιο παρατηρείτε μείωση του απαιτούμενου χρόνου εκτέλεσης των εφαρμογών όσο αυξάνουν οι υπολογιστικοί κόμβοι.

Η απόδοση στο πεδίο του χρόνου καθε φορα εξαρταται από διάφορες παραμέτρους, ανάμεσα τους το μέγεθος της εισόδου και ο αριθμός των κόμβων επεξεργασίας που τρέχουν παραλληλα την εφαρμογή.

Συμπερασματικά προκύπτει ότι η επίλυση των προβλημάτων με τη χρήση πολλαπλών υπολογιστικών συστημάτων στο Hadoop είναι αποδοτικότερη από ότι σε ένα απλό υπολογιστή. Αφού, χρησιμοποιώντας περισσότερους κόμβους για μια εργασία Hadoop MapReduce, η εργασία τεμαχίζεται σε πολλά υποπροβλήματα και εκτελείται παράλληλα σε πολλούς κόμβους του cluster προσφέροντας δυναμική εξισορρόπηση του φόρτου και επιλύοντας με αυτό το τρόπο όλο το πρόβλημα πιο γρήγορα.

Τα πειραματικά αποτελέσματα έδειξαν ότι η εφαρμογή μπορεί να μεταλλαχθεί επιτυχώς με την ύπαρξη ισχυρότερου Hardware, για να είναι σε θέση να επωφεληθεί πλήρως από τους διαθέσιμους πόρους του υπολογιστικού συστήματος.

Προσομοίωση με 4 μηχανήματα δεν πραγματοποιήθηκε , αφου δεν υπήρχε αρκετή υπολογιστική ισχύς στο σύστημα.

ΚΕΦΑΛΑΙΟ 7ο

HADOOP WEB INTERFACES

Hadoop έρχεται με πολλά web interfaces που είναι από προεπιλογή διαθέσιμα σε αυτές τις τοποθεσίες:

http://localhost:50070/ – web UI του NameNode daemon

http://localhost:50030/ – web UI του JobTracker daemon

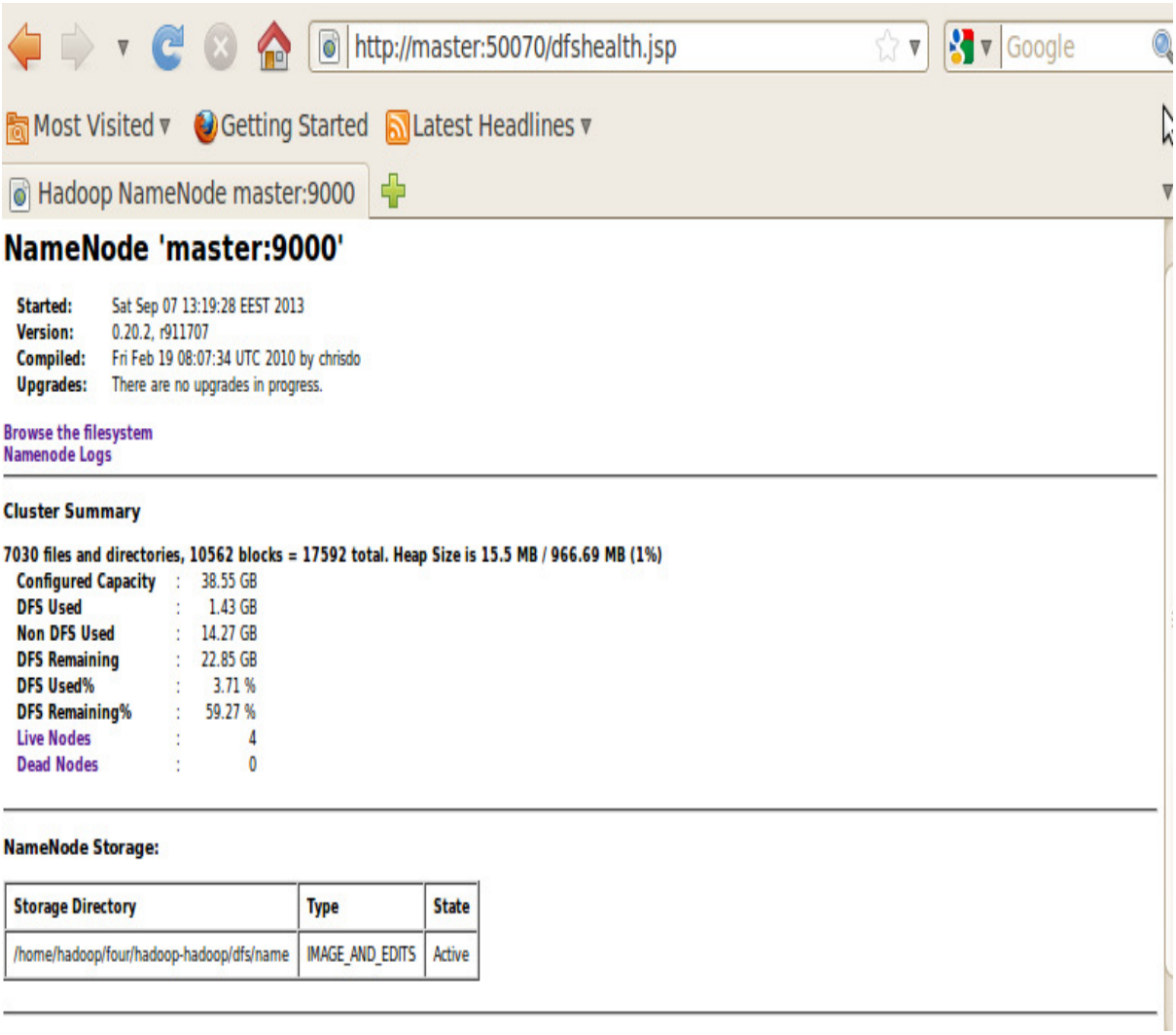
http://localhost:50060/ – web UI του TaskTracker daemon

Αυτά τα web interfaces παρέχουν συνοπτικές πληροφορίες σχετικά με το τι συμβαίνει στο Hadoop cluster μας.

7.1 NameNode Web Interface (HDFS LAYER)

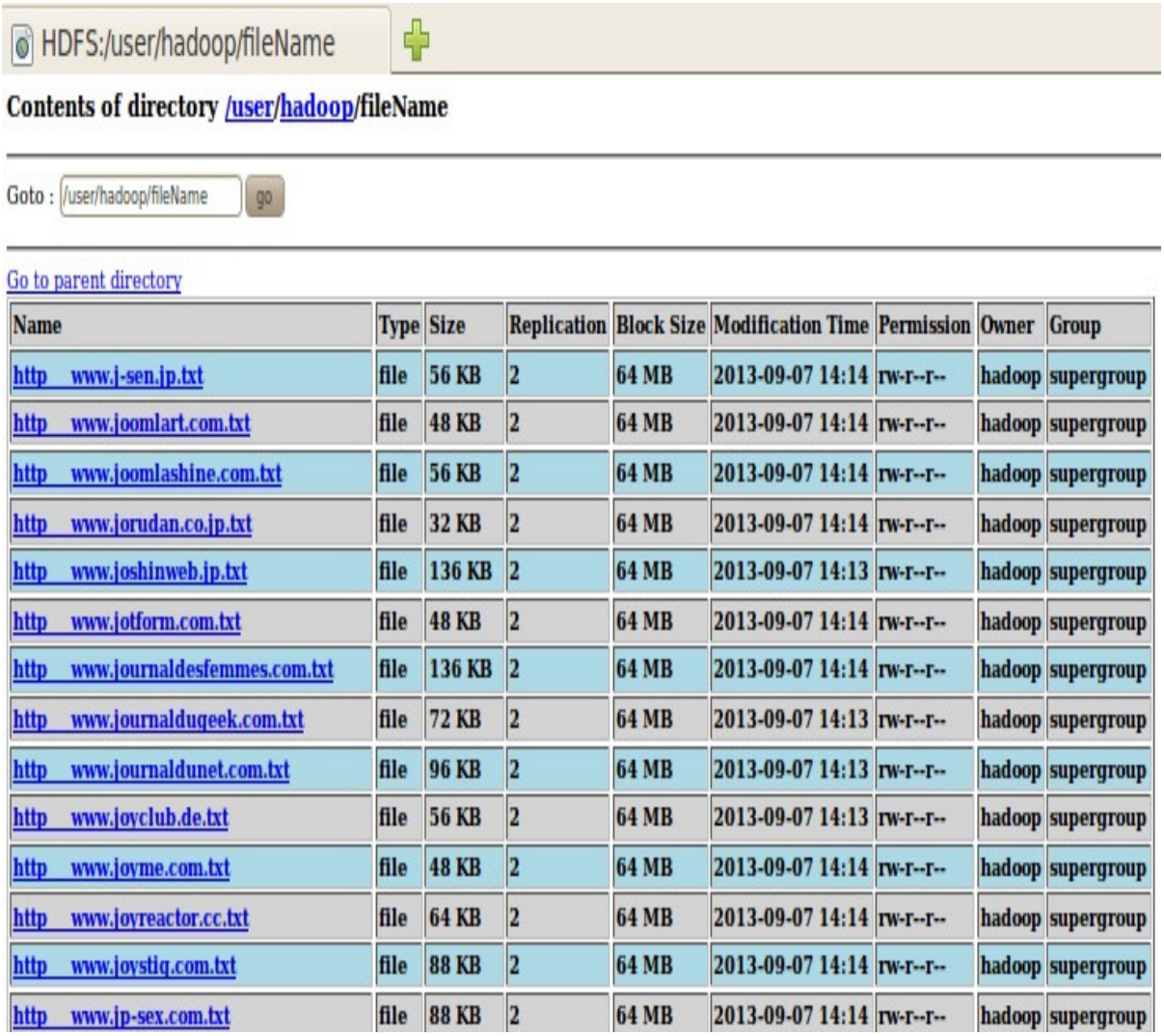
Ο Name node web UI (User interface) μας δείχνει μια σύνοψη του cluster περιλαμβάνοντας πληροφορίες σχετικά με τη συνολική / υπολειπόμενη χωρητικότητα, ενεργούς και μη ενεργούς κόμβους. Επιπλέον, μας επιτρέπει να περιηγηθούμε στο HDFS namespace και να δούμε τα περιεχόμενα των αρχείων του στο web browser. Μας δίνει, επίσης πρόσβαση στα log files του Hadoop στο τοπικό μηχάνημα.

Τη στιγμή που αρχίσουν όλες οι διαδικασίες Hadoop, μπορείτε να προβάλετε την υγεία και την κατάσταση του HDFS από ένα web interface. Χρησιμοποιήστε το http:// {your-Hadoop-server-ip}: **50070/**dfshealth.jsp. Από προεπιλογή, είναι διαθέσιμο στο **http://master:50070/** όπως φαίνεται στο Σχήμα 7.1**.**



Σχήμα 7.1: *Name Node web UI*

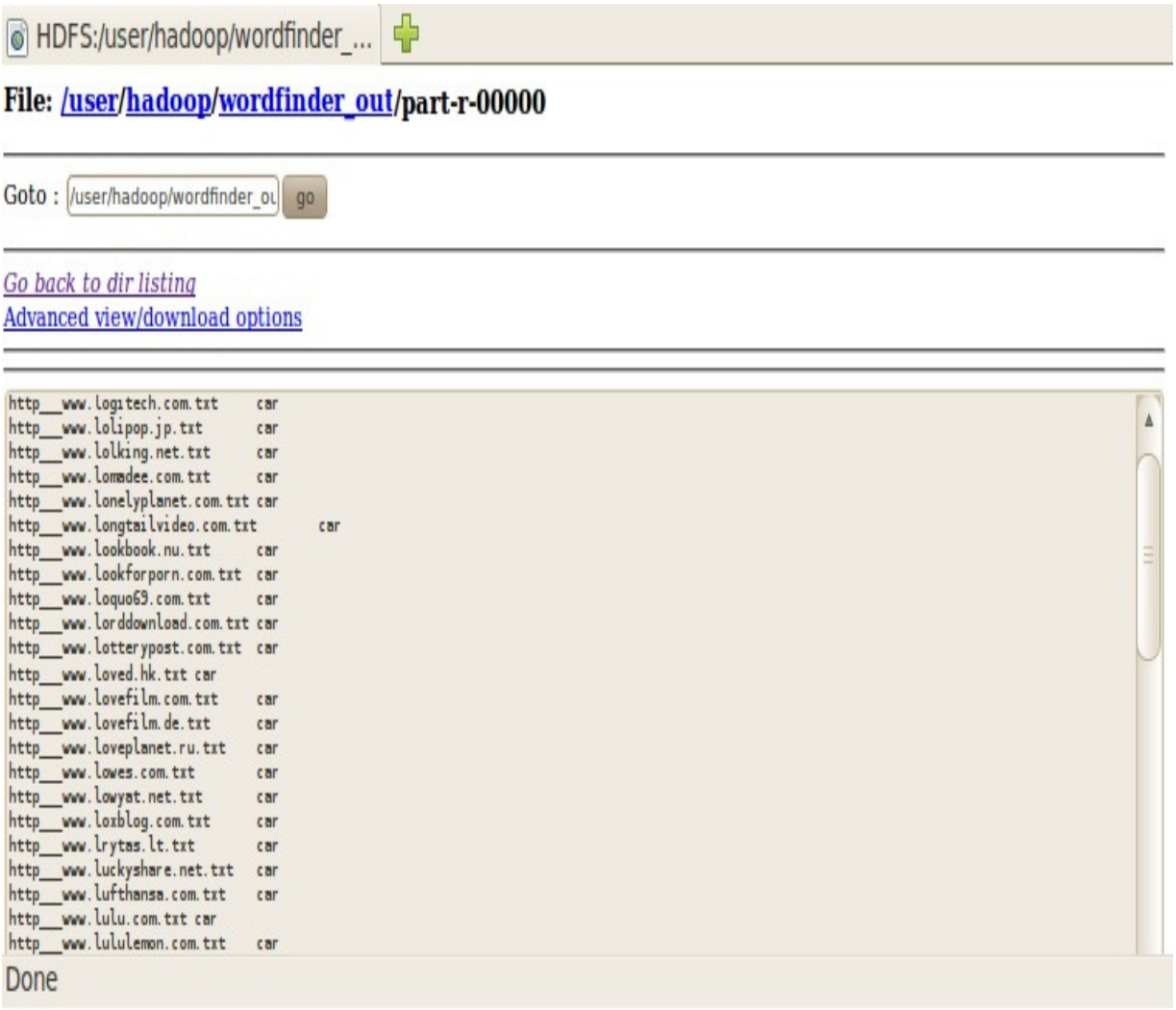
Μέσα από το Namenode, μπορείτε να επιθεωρήστε το HDFS, όπως φαίνεται στο Σχήμα 7.2, όπου επιθεωρούμε τον κατάλογο εισόδου (fileName) (το οποίο περιέχει τα δεδομένα εισόδου σας).



Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
http www.j-sen.jp.txt	file	56 KB	2	64 MB	2013-09-07 14:14	rw-r--r--	hadoop	supergroup
http www.joomlart.com.txt	file	48 KB	2	64 MB	2013-09-07 14:14	rw-r--r--	hadoop	supergroup
http www.joomlashine.com.txt	file	56 KB	2	64 MB	2013-09-07 14:14	rw-r--r--	hadoop	supergroup
http www.jorudan.co.jp.txt	file	32 KB	2	64 MB	2013-09-07 14:14	rw-r--r--	hadoop	supergroup
http www.joshinweb.jp.txt	file	136 KB	2	64 MB	2013-09-07 14:13	rw-r--r--	hadoop	supergroup
http www.jotform.com.txt	file	48 KB	2	64 MB	2013-09-07 14:14	rw-r--r--	hadoop	supergroup
http www.journaldesfemmes.com.txt	file	136 KB	2	64 MB	2013-09-07 14:14	rw-r--r--	hadoop	supergroup
http www.journalduqeek.com.txt	file	72 KB	2	64 MB	2013-09-07 14:13	rw-r--r--	hadoop	supergroup
http www.journaldunet.com.txt	file	96 KB	2	64 MB	2013-09-07 14:13	rw-r--r--	hadoop	supergroup
http www.joyclub.de.txt	file	56 KB	2	64 MB	2013-09-07 14:13	rw-r--r--	hadoop	supergroup
http www.joyme.com.txt	file	48 KB	2	64 MB	2013-09-07 14:14	rw-r--r--	hadoop	supergroup
http www.joyreactor.cc.txt	file	64 KB	2	64 MB	2013-09-07 14:14	rw-r--r--	hadoop	supergroup
http www.joystiq.com.txt	file	88 KB	2	64 MB	2013-09-07 14:14	rw-r--r--	hadoop	supergroup
http www.jp-sex.com.txt	file	88 KB	2	64 MB	2013-09-07 14:14	rw-r--r--	hadoop	supergroup

Σχήμα 7.2: *Έλεγχος του HDFS, μέσω του namenode*

Σχήμα 7.3: *Έξοδος αποτελεσμάτων της εκτέλεσης TitleFinder στο HDFS*



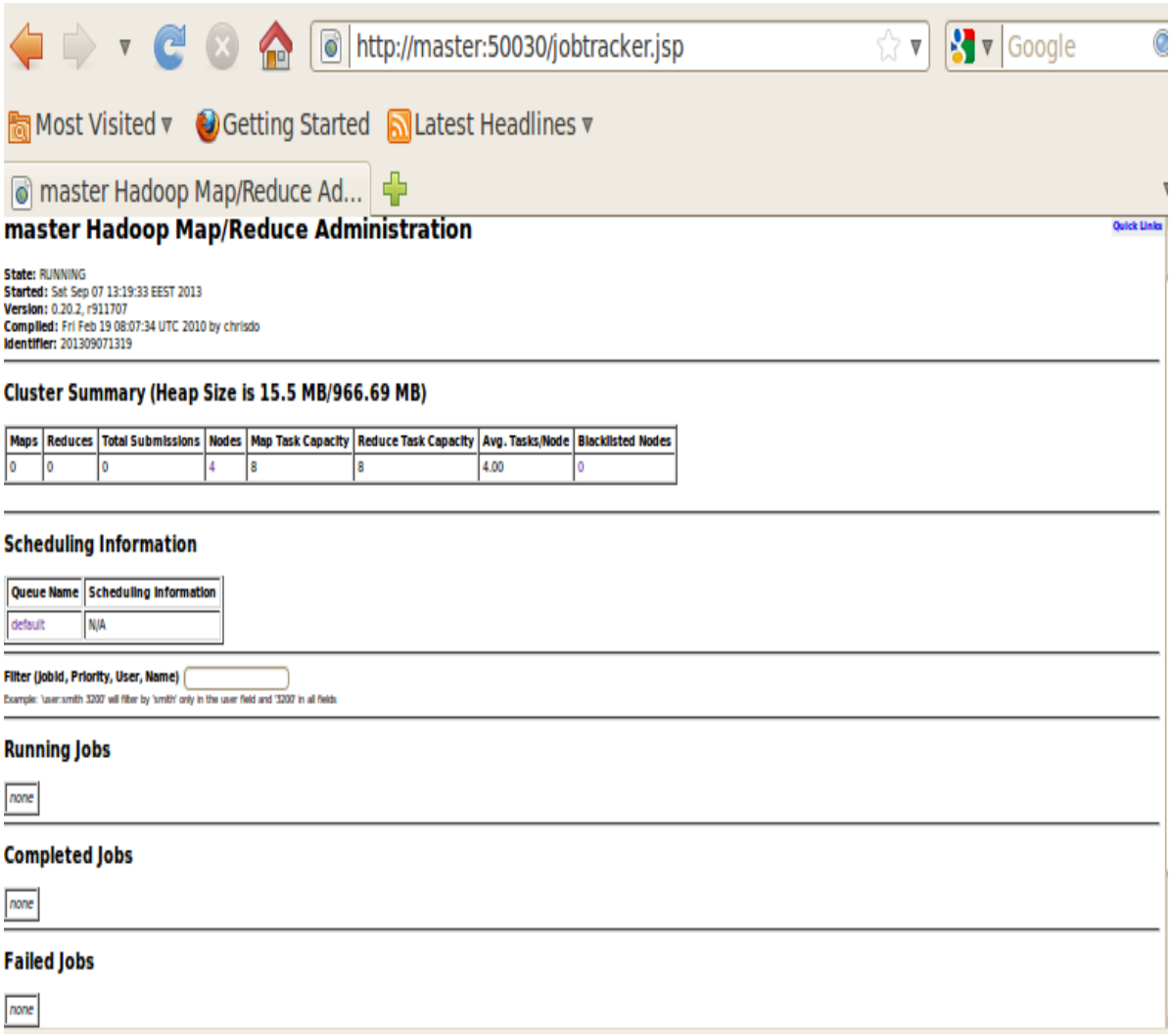
Σχήμα 7.4: *Έξοδος αποτελεσμάτων της εκτέλεσης WordFinder στο HDFS*

7.2 JobTracker Web Interface (MapReduce layer)

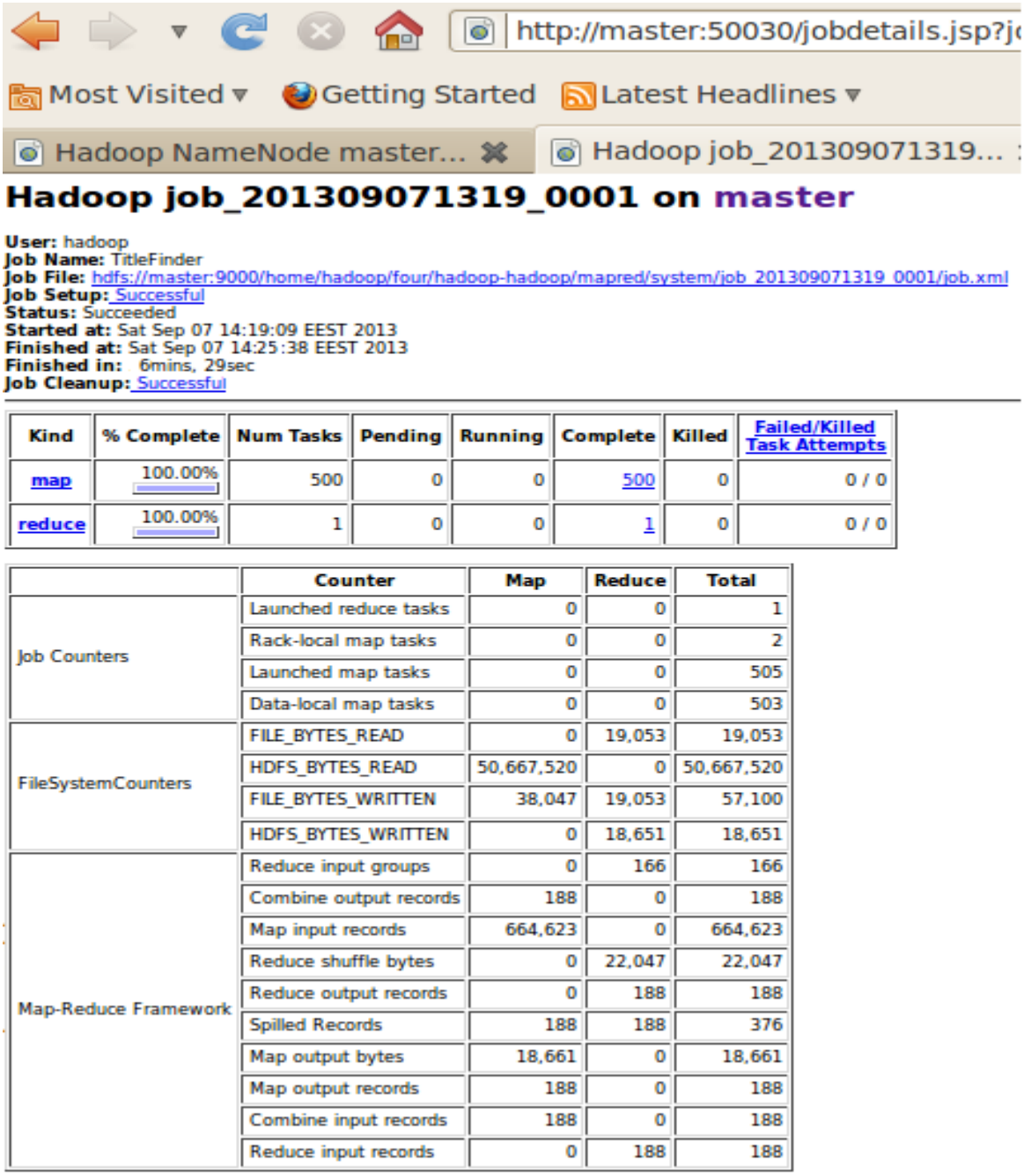
Το JobTracker web UI παρέχει πληροφορίες σχετικά με γενικά στατιστικά στοιχεία εργασίας (job) του Hadoop cluster, τους/ις τρέχοντες / ολοκληρωμένες / αποτυχημένες εργασίες (jobs) και ένα log file για το ιστορικο μιας εργασίας (Σχήμα 7.5).

Αυτό μας δίνει επίσης, πρόσβαση στα log files (αρχεία ιστορικού) του Hadoop στο τοπικό μηχάνημα (του μηχάνηματος του οποίου το web UI τρέχει επάνω).

Από προεπιλογή, είναι διαθέσιμο στο **http://master:50030/.**



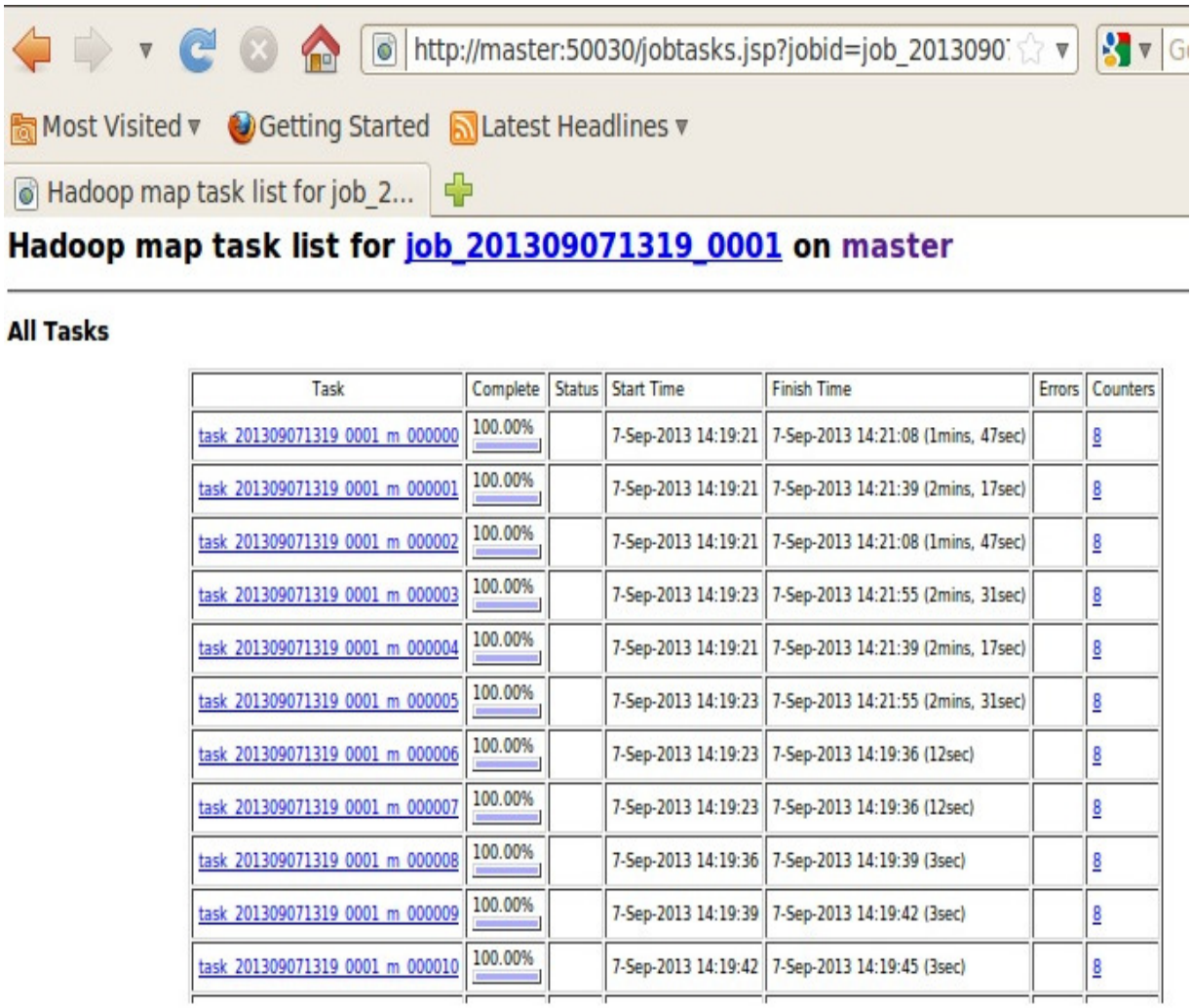
Σχήμα 7.5: *JobTracker Web UI*



Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	500	0	0	500	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

	Counter	Map	Reduce	Total
Job Counters	Launched reduce tasks	0	0	1
	Rack-local map tasks	0	0	2
	Launched map tasks	0	0	505
	Data-local map tasks	0	0	503
FileSystemCounters	FILE_BYTES_READ	0	19,053	19,053
	HDFS_BYTES_READ	50,667,520	0	50,667,520
	FILE_BYTES_WRITTEN	38,047	19,053	57,100
	HDFS_BYTES_WRITTEN	0	18,651	18,651
Map-Reduce Framework	Reduce input groups	0	166	166
	Combine output records	188	0	188
	Map input records	664,623	0	664,623
	Reduce shuffle bytes	0	22,047	22,047
	Reduce output records	0	188	188
	Spilled Records	188	188	376
	Map output bytes	18,661	0	18,661
	Map output records	188	0	188
	Combine input records	188	0	188
	Reduce input records	0	188	188

Σχήμα 7.6: *Γραφική Διασύνδεση της εκτέλεσης Job*

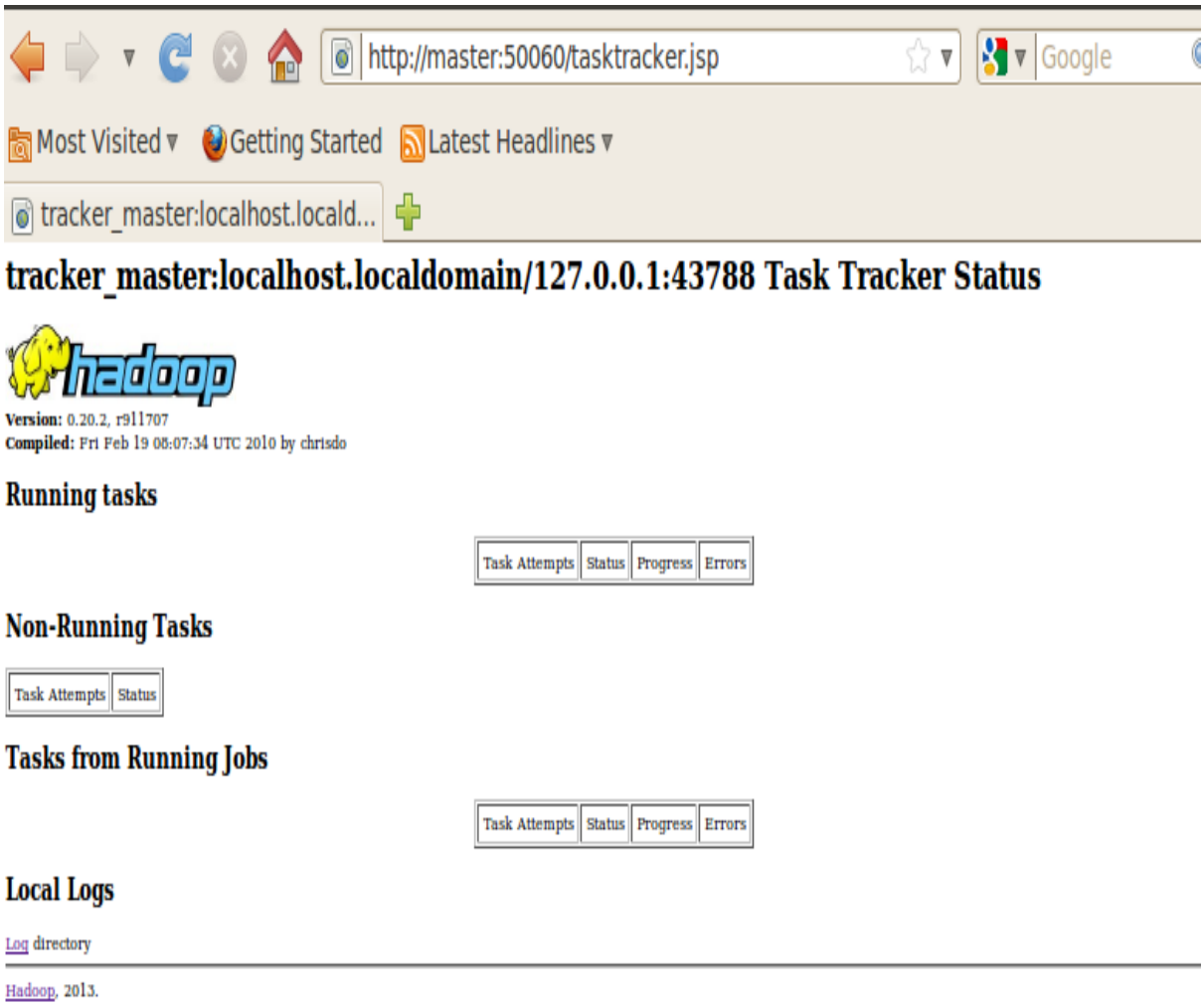


Σχήμα 7.7: *Εμφανίζονται όλες ολοκληρωμένες εργασίες (Tasks)*

7.3 TaskTracker Web Interface (MapReduce layer)

Το task tracker web UI δείχνει τις εργασίες που εκτελούνται και αυτές που δεν εκτελούνται (Σχήμα 7.8). Μας δίνει επίσης, πρόσβαση στα log files (αρχεία ιστορικού) του Hadoop στο τοπικό μηχάνημα.

Από προεπιλογή, είναι διαθέσιμο στο **http://master:50060/.**



Σχήμα 7.8: *Task Tracker web UI*

Συμπεράσματα και **Μ**ελλοντικές **Β**ελτιώσεις

Στην παρούσα εργασία παρουσιάστηκε το Hadoop, ένα open source framework του Apache Software Foundation, που δημιουργήθηκε για να υποστηρίζει τη λειτουργία διαφόρων εφαρμογών σε μεγάλα cluster από υπολογιστές γενικής χρήσης. Επίσης παρουσιάστηκε και το μοντέλο MapReduce που προσφέρει μία σχεδόν έτοιμη τεχνική επεξεργασίας και διαχείρισης ενός μεγάλου όγκου δεδομένων, και το οποίο λειτουργεί επάνω στο HDFS, και όπως υποδεικνύει το όνομά του αποτελείται από τις συναρτήσεις Map() και Reduce(). Επιπλέον, παρουσιάζονται τα αποτελέσματα μιας υλοποίησης, βασισμένης στο παραπάνω μοντέλο.

Η υλοποίηση που παρουσιάστηκε στην παρούσα εργασία, θα πρέπει να λειτουργήσει με ένα μεγάλο όγκο δεδομένων, και με έναν όσο και με περισσότερους κόμβους Hadoop, προκειμένου να γίνει η σύγκριση των αποτελεσμάτων. Απώτερος σκοπός της συγκεκριμένης υλοποίησης θα είναι η επίτευξη του καλύτερου συνολικού χρόνου επεξεργασίας, καθώς, όπως ενδεχομένως είναι προφανές, αυξάνοντας σε αριθμό τα υπολογιστικά συστήματα, αυξάνεται και η αποθηκευτική και υπολογιστική ισχύς. Έτσι το Hadoop επιτρέπει τη βέλτιστη χρήση των κατανεμημένων υπολογιστικών πόρων, ώστε να επιτύχουμε τους καλύτερους χρόνους στις εφαρμογές μας, σε σχέση με την εκτέλεση σε έναν υπολογιστή.

Δημιουργώντας με χρήση VMs (*Virtual machines*) ένα cluster με τέσσερις κόμβους στο ίδιο μηχάνημα, η υπολογιστική ισχύς του συστήματος δεν ήταν αρκετή για να ανταπεξέλθει στις στις αυξημένες προδιαγραφές τους συστήματος Hadoop, έτσι ώστε να μας δώσει τα επιθυμητά αποτελέσματα. Αυτό είχε σαν αποτέλεσμα να μη δούμε τεράστια διαφορά στους χρόνους επεξεργασίας των δεδομένων λόγω του οτι καταναλώνονται οι ίδιοι πόροι του υπολογιστικού συστήματος, και έτσι επιβαρύνεται αρκετά το σύστημα με αποτέλεσμα ολόκληρη η διεργασία να οδηγείται σε επιβράδυνση. Για το λόγο αυτό, χρησιμοποιήθηκαν τρείς κόμβοι γι'αυτή την εργασία, αντί για τέσσερις που εγκαταστάθηκαν επιτυχώς, για να αποδοθεί το επιθυμητό τελικό αποτέλεσμα.

Επίσης, η υλοποιημένη εφαρμογή σε Hadoop MapReduce με το ονομα "**Combine_ Code**" επειδή χρησιμοποιεί σχετικά πολύ μικρά αρχεία εισόδου, για την επεξεργασία των δεδομένων της, είναι ένα πρόβλημα στο Hadoop περιβάλλον. Το Hadoop λειτουργεί καλύτερα με έναν μικρό αριθμό μεγάλων αρχείων, απ' ο,τι με ένα μεγάλο αριθμό μικρών αρχείων.

Το HDFS είναι ένα κατανεμημένο σύστημα αρχείων και είναι κυρίως σχεδιασμένο για την streaming πρόσβαση & μαζική επεξεργασία (batch processing) μεγάλου όγκου δεδομένων. Το προεπιλεγμένο μέγεθος Block του HDFS είναι 64MB. Σε περίπτωση αποθήκευσης πολλών μικρών αρχείων, τα οποία είναι πάρα πολύ μικρότερα από το μέγεθος του block, δεν μπορεί να αντιμετωπίζονται αποτελεσματικά από τον HDFS, και κατά συνέπεια η απόδοση του πέφτει δραματικά.

Μια μελλοντική επέκταση αυτής της εργασίας θα αποτελούσε την εφαρμογή της παράλληλης εκτέλεσης σε περισσότερες απο δύο διεργασίες, και σε σταθερές υπολογιστικές μονάδες (π.χ., laptops, Desktops). Επιπλέον, θα ήταν προτιμότερο το λειτουργικό σύστημα να μετατραπεί από Ubuntu Linux Desktop σε Ubuntu Server για την καλύτερη απόδοση, και σαφώς λιγότερη κατανάλωση των διαθέσιμων πόρων του συστήματος (μνήμης RAM, ταχύτητας επεξεργαστή, κλπ), (π.χ. , δεν θα υπάρχει γραφικό περιβάλλον χρήστη που τώρα δεσμεύει αχρείαστα μνήμη). Έστω ο κύριος κομβος του cluster (NameNode) θα πρέπει τυπικά να τρέχει σε 64-bit hardware για να αποφευχθεί το όριο των 3 GB στο σύστημα, επειδή η διεργασία αυτή είναι από τις πιο σημαντικές ενός Hadoop Cluster και είναι αρκετά απαιτητική σε μνήμη. Εναλακτικά, θα μπορούσε κάποιος να νοικιάζει απλά ένα cluster υπολογιστών - *instances* (π.χ., Amazon EC2) μαζί με τους πόρους από το νέφος, (*cloud*) σύμφωνα με τις ανάγκες του, και να πληρώνει για την τρέχουσα χρήση του υπολογιστικού νέφους (*cloud computing*), όπου βέβαια και η υπολογιστική ισχύς θα είναι κατά πολύ μεγαλύτερη αυτής που χρησιμοποιήθηκε για τα πειράματα της παρούσης πτυχιακής εργασίας.

ΒΙΒΛΙΟΓΡΑΦΙΑ - ΑΝΑΦΟΡΕΣ

[1]. Big data - http://en.wikipedia.org/wiki/Big_data

[2]. Chuck Lam, (2010): *Hadoop in Action*, Manning Publications, (σελ.19-20)

[3]. Chuck Lam, (2010): *Hadoop in Action*, Manning Publications, (σελ.4-5)

[4]. Why Hadoop; - http://bigdata.wordpress.com/2010/05/16/why-hadoop/

[5]. Hadoop Page - http://hadoop.apache.org/

[6]. Tom White, (2010): *Hadoop: The Definitive Guide*, 2nd Edition. O'Reilly Media/Yahoo Press, (σελ.12-13)

[7]. Chuck Lam, (2010): *Hadoop in Action*, Manning Publications, (σελ.29-31)

[8]. http://casablanca.dblab.ece.ntua.gr/cloudwiki/index.php/Main_Page

[9]. Jason Venner, (2009). *Pro Hadoop*. 1st ed. : Apress, (σελ.5-6)

[10]. HDFS Architecture **-** http://hadoop.apache.org/docs/stable/hdfs_design.html#Browser+Interface

[11].http://hadoop.apache.org/docs/r0.18.3/mapred_tutorial.html#Map%2FReduce+-+User+Interfaces

[12]. MapReduce **-** http://en.wikipedia.org/wiki/MapReduce

[13]. http://wiki.apache.org/hadoop/HowManyMapsAndReduces

[14]. Hadoop Wiki - http://wiki.apache.org/hadoop/HowManyMapsAndReduces

[15]. Jeffrey Dean, Sanjay Ghemawat: *MapReduce: Simplified Data Processing on Large Clusters* Commun.ACM51(1),2008

[16]. http://developer.yahoo.com/hadoop/tutorial/module4.html

[17]. Tom White, (2010): *Hadoop: The Definitive Guide*, 2nd Edition. O'Reilly Media/Yahoo Press, (σελ. 28)

[18]. Hadoop Streaming-http://hadoop.apache.org/docs/stable/streaming.html#Hadoop+Streaming

[19]. Michael G. Noll, "*Running Hadoop on Ubuntu Linux (Single-Node Cluster)*": http://www.michael-noll.com/wiki/Running_Hadoop_On_Ubuntu_Linux_(Single-Node_Cluster).

[20]. Michael G. Noll, "*Running Hadoop on Ubuntu Linux (Multi-Node Cluster)*": http://www.michael-noll.com/wiki/Running_Hadoop_On_Ubuntu_Linux_(Multi-Node_Cluster).

[21]. http://netcins.ceid.upatras.gr/OpSys-II/Lecture11-12/hadoop_frontistirio.pdf

[22]. Chuck Lam, (2010): *Hadoop in Action*, Manning Publications, (σελ.77-80)

ΠΑΡΑΡΤΗΜΑ Α΄ - Εγκατάσταση του Hadoop και ρύθμιση του Cluster

Περιβάλλον Εργασίας

Το περιβάλλον εργασίας που επιλέξαμε για να εκτελέσουμε το πρόγραμμα που κατασκευάσαμε, ήταν το Ubuntu Linux (10.04.4), εγκατεστημενο σε εικονική μηχανή, συγκεκριμένα στο VM VirtualBox**.**

Cluster Setup

Το τρέξιμο του προγράμματος δοκιμάστηκε με τις ακόλουθες εκδόσεις λογισμικού

- **Ubuntu Linux** 10.04.4
- **Hadoop** (version **0.20.2**)

Απαιτήσεις Συστήματος

OpenJDK 6

Το hadoop έχει εγκατεστημένη απο προεπιλογή μια έκδοση της **java 1.5x**. Ωστόσο, η κατάλληλη έκδοση που χρειάζεται για να τρέξει το Hadoop είναι το **java 1.6x**.

Άρα, για να γίνει η εγκατάσταση της σωστής έκδοσης ακλουθούμε τα εξής βήματα:

1. Εισάγουμε το Canonical Partner Repository στις apt αποθήκες μας

Δηλαδή μπαίνουμε στο κειμενογράφο κονσόλας **vi /etc/apt/sources.list** και γραφούμε μέσα το

```
deb http://archive.canonical.com/ lucid partner
```

έπειτα

2. Αναβαθμίζουμε το **sources.list** με την ακόλουθη εντολή

```
apt-get update
```

3. Εγκαθιστούμε το το JDK της **openjdk** με την εντολή

apt-get install openjdk-6-jdk

Για να δούμε εάν η εγκατάσταση του JDK έχει ρυθμιστεί σωστά δίνουμε την εντολή

java –version

επίσης, μπαίνουμε με την εντολή **vi conf/hadoop-env.sh** στο κειμενογράφο και ρυθμίζουμε το path της java ως εξής :

export JAVA_HOME=/usr/lib/jvm/java-1.6.0-openjdk

και μετά σώζουμε τις αλλαγές .

Προσθέτοντας έναν Χρήστη στο Hadoop

Αυτό γίνεται με τον εξής τρόπο:

addgroup hadoop **adduser --ingrup hadoop hadoop**

με τις παραπάνω εντολές προσθέτουμε ένα **hadoop** χρήστη και την ομάδα **hadoop** στο σύστημα μας (δίνουμε στον hadoop χρήστη το ίδιο όνομα με το group)

Networking

Επεξεργασία του αρχείου /etc/hosts

Με αυτή την ενημέρωση του αρχείου hosts δεν χρειάζεται να θυμόμαστε απ'έξω τα ips των μηχανημάτων του hadoop cluster. Το αρχείο hosts περιέχει αντιστοιχίσεις ονομάτων σε IPs.

Για να είναι πιο απλό, θα αναθέσουμε την IP διεύθυνση 192.168.56.1 στην master machine και 192.168.56.2, 192.168.56.3, 192.168.56.4 στα slaves machines.

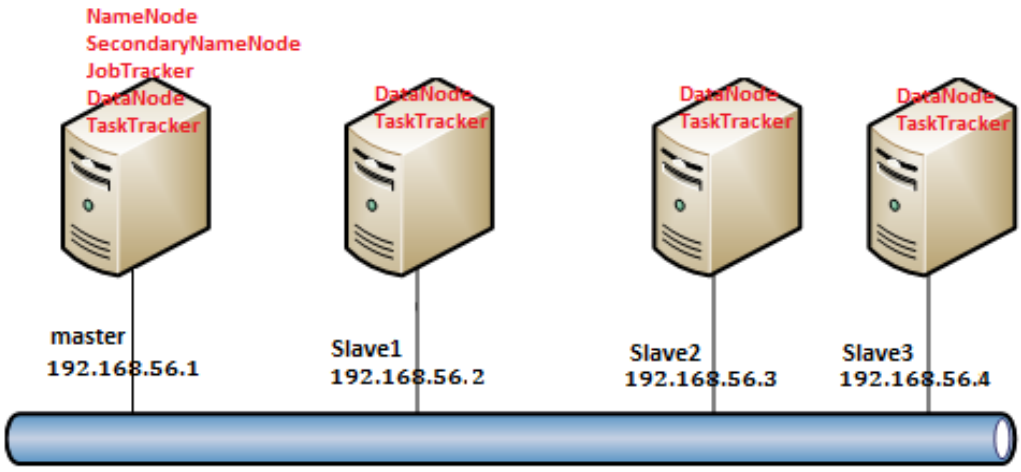
Σε όλα τα μηχανήματα προσθέτουμε στο αρχείο /etc/hosts τα παρακάτω ζεύγη:

ip_του_μηχανήματος	**hostname_του_μηχανήματος**	**ρόλος**
192.168.56.1	**master**	(NN,SN,JT,DN,TT)

192.168.56.2	**Slave1**	(DataNode and TaskTracker)
192.168.56.3	**Slave2**	(DataNode and TaskTracker)
192.168.56.4	**Slave3**	(DataNode and TaskTracker)

και **σβήνουμε** την εγγραφή

127.0.1.1 hostname του μηχανήματος



Σχήμα Α'.1: *Τοπολογία Hadoop - Small cluster* ***(Πηγή****: http://abloz.com/2012/05/23/hadoop-three-node-cluster-installation-configuration-details-an-instance-of.html)*

Ρύθμιση για Passwordless ssh (Secure Shell)

Το hadoop για να λειτουργήσει θα πρέπει οι υπολογιστές που αποτελούν το cluster να μπορούν να συνδέονται μεταξύ τους με την χρήση ssh χωρίς κωδικό. Αυτό επιτυγχάνεται με την χρήση ζεύγους ιδιωτικού/δημόσιου κλειδιού. Η βασική ιδέα της τεχνικής αυτής είναι ότι ο κάθε χρήστης μπορεί να έχει ένα προσωπικό ιδιωτικό κλειδί (αρχείο) το οποίο έρχεται ζεύγος με ένα δημόσιο κλειδί. Ο χρήστης τοποθετεί το δημόσιο κλειδί στους υπολογιστές τους οποίους θέλει να έχει πρόσβαση, και με την χρήση του ιδιωτικού μπορεί να συνδέεται σε αυτούς χωρίς κωδικό. Η πιστοποίηση του χρήστη γίνεται καθώς μόνο ο χρήστης έχει στην κατοχή του το ιδιωτικό κλειδί.

```
ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
```

η εντολή δημιουργεί ένα ιδιωτικό κλειδί id_rsa και ένα δημόσιο κλειδί id_rsa.pub. Τρέχουμε

```
cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

η παραπάνω εντολή βάζει το δημόσιο κλειδί id_rsa.pub στον κατάλογο με τα αποδεκτά δημόσια κλειδιά του χρήστη root του master μηχανήματος. Με αυτόν τον τρόπο, όποιος έχει στην κατοχή του το ιδιωτικό κλειδί id_rsa μπορεί να συνδεθεί στο root@master.

Συνδεση στο master.

ssh master (τώρα *δεν* θα πρέπει να ζητάει κωδικό)

Λοιπόν**,** σύνδεση απο το master σε master

Γενική σύνταξη: ssh **<***hostname of master***>**

```
hadoop@master:~$ ssh master
The authenticity of host 'master (192.168.56.1)' 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.
Linux master 2.6.20-16-386 #2 Thu Jun 7 20:16:13 UTC 2007 i686
...
hadoop@master:~$
```

... Και από το master σε slave1, slave2....

Γενική σύνταξη: ssh **<***hostname of* slave **>**

```
hadoop@master:~$ ssh Slave1
The authenticity of host 'slave (192.168.56.2)' 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 ... hadoop@slave1:~$

και απαντάμε "**yes**" ώστε να προστεθεί στους known hosts ο υπολογιστής στον οποίο κάνουμε ssh. Θα πρέπει να κάνουμε ssh σε **όλους** τους slaves του cluster.

Σημεία Προσοχής

1. Θα πρέπει οι διάφοροι slaves να είναι στους ssh known_hosts του master.
2. Υπάρχει χρόνος που απαιτείται για να δει το NameNode τα DataNodes. Στο χρόνο αυτό ο NameNode είναι σε *Safe mode*.

Masters vs**. S**laves

Συνήθως ένα μηχάνημα στο cluster χαρακτηρίζεται ως ο NameNode και ένα άλλο μηχάνημα ως JobTracker**,** αποκλειστικά. Αυτά είναι τα πραγματικά **"**master nodes**"**. Τα υπόλοιπα μηχανήματα του cluster δρουν τόσο ως DataNode και TaskTracker. Αυτά αποκαλούνται είτε σαν slaves είτε σαν **"**worker nodes**"**.

Διαμόρφωση

Επεξεργαζόμαστε τα αρχεία masters και slaves που βρίσκονται στο φάκελο

$ **hadoop-0.20.2/**conf/.

conf/masters (master only)

Η μηχανή στην οποία εκτελείται bin / start-dfs.sh, θα γίνει το primary NameNode.

Στο master, ενημερώνουμε conf / masters που μοιάζει με το παρακάτω:

conf/masters (on master)
master

conf/ slaves (master only)

Επισης, στο master, ενημερώνουμε conf / slaves όπως φαίνεται παρακάτω:

conf/ slaves (on master)
master slaves1 slaves2 slaves3

Εάν έχετε επιπλέον slave nodes, απλώς προσθέστε τις στο conf / slaves αρχείο, ένα hostname ανά γραμμή.

conf/ slaves (on master)
master slaves1 slaves2 slaves3 anotherslave04 anotherslave05 anotherslave06

Εδώ εμείς χρησιμοποιούμε το master μηχάνημα για να τρέξουμε DataNode και TaskTracker, έτσι έχουμε δώσει τον master στο αρχείο σκλάβων.

Έτσι, ως master ορίζεται μόνο μία απο αυτές, ενω ως slaves ορίζονται όλες οι υπόλοιπες στο cluster, συμπεριλαμβανομένου και του master.

Αν δεν θέλετε να τρέξετε DataNode και TaskTracker στο κύριο κόμβο σας απλά αφαιρέστε αυτήν την καταχώρηση.

Hadoop

Εγκατάσταση hadoop 0.20.2 Cluster

Τα παρακάτω βήματα πρέπει να γίνουν σε όλους τους υπολογιστές του Cluster.

Κατεβάζουμε το installation package του Hadoop από τον *ΙΣΤΟΤΟΠΟ* https://archive.apache.org/ και στη συνέχεια αποσυμπιέζουμε τα περιεχόμενα σε κάποιο φάκελο στο χώρο του hadoop χρήστη που δημιουργήσαμε προηγουμένως.

wget https://archive.apache.org/dist/hadoop/core/hadoop-0.20.2/hadoop-0.20.2.tar.gz

tar -xvf hadoop-0.20.2.tar.gz

Πρέπει επίσης να αλλάξουμε τον κάτοχο των αρχείων Hadoop να είναι ο hadoop user και το group

chown -R hadoop:hadoop hadoop-0.20.2

ln -s hadoop-0.20.2/ hadoop

Ρύθμιση του Hadoop

Ο στόχος μας είναι η ρύθμιση του Hadoop για ένα **Multi-Node** Small Cluster. Περισσότερες πληροφορίες για το τι συμβαίνει σε αυτή την ενότητα είναι διαθέσιμες στο Hadoop Wiki

Hadoop-env.sh

Η απαιτούμενη τιμή περιβάλλοντος που πρέπει να διαχειριστούμε για το Hadoop είναι η JAVA_HOME η οποία πρέπει να δείχνει στον κατάλογο που έχω εγκατεστημένη την Java**.** Ανοίγουμε το **$** hadoop-0.20.2/conf/hadoop-env.sh με ένα συντάκτη (*editor*) και κάνουμε τα εξής βήματα**:**

Αλλάζουμε

The java implementation to use. Required.

```
# export JAVA_HOME=/usr/lib/jvm/j2sdk1.5-sun
```

σε

```
# The java implementation to use. Required.
  export JAVA_HOME=/usr/lib/jvm/java-1.6.0-openjdk
```

επίσης ,

Απενεργοποιούμε το IPv6 για το Hadoop προστέθοντας την ακόλουθη γραμμή στο /conf/hadoop-env.sh

```
export HADOOP_OPTS="-Djava.net.preferIPv4Stack=true"
```

Το hadoop ρυθμίζεται από **3** βασικά xml αρχεία τα οποία βρίσκονται στον κατάλογο **$** hadoop-0.20.2/conf (core-site.xml hdfs-site.xml και mapred-site.xml) και από τα αρχεία masters και slaves**:**

Στο αρχείο conf/core-site.xml

```
<configuration>
<property>
 <name>hadoop.tmp.dir</name>
<value>/home/hadoop/four/hadoop-${user.name}</value>
<description>A base for other temporary directories.</description>
</property>

<property>
  <name>fs.default.name</name>
<value>hdfs:// MASTER_NODE :9000</value>
</property>
```

```
</configuration>
```

Στο αρχείο conf/hdfs-site.xml

Η default τιμή για τα αντίγραφα των αρχείων που ανεβάζουμε στο HDFS είναι **3**, δηλαδή αντιγράφεται 2 φορές κάθε block που ανεβαίνει στο HDFS και τοποθετείται σε 3 διαφορετικούς κόμβους. για λόγους ανάκαμψης από την αποτυχία κάποιου κόμβου, το οποίο βοηθά στη διατήρηση της σταθερότητας του συστήματος. Επειδή το cluster που φτιάξαμε έχει μόνο **4** κόμβους θέτουμε τον αριθμό των αντιγράφων σε **2**, όπως είναι σύνηθες για μικρά clusters (2-10 nodes). Η τιμή αυτή θα πρέπει να αλλάξει ανάλογα με το μέγεθος του cluster και ύστερα από δοκιμές. Τιμές υψηλότερες από 3 συνήθως δεν είναι αναγκαίες.

Το hdfs για να λειτουργήσει χρειάζεται να δημιουργηθούν δυο ειδών κατάλογοι: Ο dfs.name.dir και οι dfs.data.dir. Ο dfs.name.dir είναι ο κατάλογος που περιέχει τις αντιστοιχήσεις αρχείων σε υπολογιστές και υπάρχει στον namenode, και οι κατάλογοι dfs.data.dir περιέχουν τα κομμάτια των αρχείων και βρίσκονται στους datanodes.

Άρα, στον master δημιουργούμε τον κατάλογο name.dir και στον slave δημιουργούμε τον κατάλογο data.dir.

```
<configuration>
<property>
 <name>dfs.replication</name>
<value>2</value>
<description> block replication</description>
</property>

<property>
  <name>dfs.name.dir</name>
  <value>${hadoop.tmp.dir}/dfs/name</value>
<description> Where the NameNode metadata should be stored</description>
</property>
```

```
<property>
  <name>dfs.data.dir</name>
  <value>${hadoop.tmp.dir}/dfs/data</value>
<description> Where DataNodes store their  blocks</description>
</property>
 </configuration>
```

Στο αρχείο conf/mapred-site.xml

Γράφουμε την ip του master ή το domain name του στη θέση του "*MASTER_NODE* ".

```
<configuration>
 <property>
 <name>mapred.job.tracker</name>
<value>MASTER_NODE :9001</value>
</property>
</configuration>
```

Διαμόρφωση του NameNode

Το πρωτο βήμα για την εκκινήσουμε την εγκατάσταση του Hadoop ειναι να γίνει format της HDFS ενός νέου κατανεμημένου συστήματος αρχείων μέσω NameNode. Όταν γίνεται format στο HDFS, *χάνονται* όλα τα δεδομένα που βρίσκονται εκείνη τη στιγμή στο HDFS. Αν χρειαστεί να ξανακάνουμε format αργότερα, τότε πρέπει να πατήσουμε κεφαλαίο "Y". Διαφορετικά, δεν γίνεται το format. Για να διαμορφώσουμε το σύστημα αρχείων, εκτελούμε την παρακάτω εντολή

```
hadoop@master:~$ /usr/local/hadoop/bin/hadoop namenode -format
```

Η έξοδος θα είναι κάπως έτσι:

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

... INFO dfs.Storage: Storage directory /app/hadoop/tmp/dfs/name has been successfully formatted.

hadoop@master:/usr/local/hadoop$
```

Eκκίνηση / **T**ερματισμός ενός **M**ulti-**N**ode **C**luster

Εκτέλεστε την εντολή:

```
hadoop@master:~$ /usr/local/hadoop/bin/start-all.sh
```

Αυτή η εντολή ξεκινά όλα τα Hadoop daemons, το NameNode, DataNodes, το JobTracker και TaskTrackers στο κυριο μηχανημα μας ενω στα Slaves μηχανηματα μας ξεκινάνε τα TaskTrackers και DataNodes.

Η έξοδος θα είναι κάπως έτσι:

```
starting namenode, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-namenode-master.out

slave2: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-datanode-slave2.out

slave1: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-datanode-slave1.out

slave3: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-datanode-slave3.out

master: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-datanode-master.out

master: starting secondarynamenode, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-secondarynamenode-master.out

starting jobtracker, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-jobtracker-master.out
```

```
master: starting tasktracker, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-tasktracker-master.out
slave2: starting tasktracker, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-tasktracker-slave2.out
slave3: starting tasktracker, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-tasktracker-slave3.out
slave1: starting tasktracker, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-tasktracker-slave1.out
hadoop@master:/usr/local/hadoop$
```

Ένα ικανό εργαλείο για τον έλεγχο του εαν τρέχουν οι αναμενόμενες Hadoop διεργασίες είναι το **jps** όπως φαίνεται παρακάτω.

```
hadoop@master:/usr/local/hadoop$ jps
16694 Jps
16341 JobTracker
15857 NameNode
16046 DataNode
16530 TaskTracker
16245 SecondaryNameNode
```

Με τον ίδιο τρόπο εκτλέστε την παρακάτω εντολή για να σταματήσετε όλα τα Hadoop daemons που τρεχουν στα μηχανηματα σας:

```
hadoop@master:~$ /usr/local/hadoop/bin/stop-all.sh
```

Η έξοδος θα είναι κάπως έτσι:

```
Stopping  jobtracker
slave2: stopping tasktracker
master: stopping tasktracker
```

```
slave3: stopping tasktracker
slave1: stopping tasktracker
stopping namenode
master: stopping datanode
slave2: stopping datanode
slave1: stopping datanode
slave3: stopping datanode
master: stopping secondarynamenode
hadoop@master:/usr/local/hadoop$
```

Μπορείτε επίσης να ελέγξετε με το **netstat** αν Hadoop 'ακούει' στις διαμορφωμένες μένες θύρες με την παρακάτω εντολή.

```
hadoop@master:~$ sudo netstat -pltenlgrep java
tcp  0   0 0.0.0.0:38197      0.0.0.0:*    LISTEN   1001
tcp  0   0 0.0.0.0:50070      0.0.0.0:*    LISTEN   1001
tcp  0   0 0.0.0.0:59418           0.0.0.0:*    LISTEN   1001
tcp  0   0 127.0.0.1:44412         0.0.0.0:*    LISTEN   1001
tcp  0   0 192.168.56.101:9000   0.0.0.0:*    LISTEN   1001
tcp  0   0 192.168.56.101:9001   0.0.0.0:*    LISTEN   1001
tcp  0   0 0.0.0.0:50090           0.0.0.0:*    LISTEN   1001
tcp  0   0 0.0.0.0:50060           0.0.0.0:*    LISTEN   1001
tcp  0   0 0.0.0.0:50030           0.0.0.0:*    LISTEN   1001
hadoop@master:~$
```

Αν υπαρχουν πιθανά λάθη/προβλήματα, εξεταστε τα log files στον καταλογο **/** log**/**.

Πρόσβαση στα Logs μέσω της γραμμής εντολών**:** hadoop/logs$ ls -ltr

HDFS Daemons

Επίσης, υπάρχουν μερικές εντολές που ξεκινούν ξεχωριστά τα daemons οπως αυτές παρακάτω **:**

Στον **Namenode** ξεκινάμε το HDFS cluster με την παρακάτω εντολή**:**

```
hadoop@master:/usr/local/hadoop$ bin/start-dfs.sh
```

Με την εντολή αυτή ξεκινάει:

- ο NameNode στο τοπικό μηχάνημα
- ο DataNode σε κάθε μηχάνημα που αναφέρεται μέσα στο αρχείο $HADOOP_HOME/conf/slaves και
- ο Secondary NameNode σε κάθε μηχάνημα που αναφέρεται μέσα στο αρχείο $HADOOP_HOME/conf/masters.

```
hadoop@master:/usr/local/hadoop$ bin/start-dfs.sh

starting namenode, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-namenode-
master.out
master: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop -datanode-
master.out
slave3: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop -datanode-
slave3.out
slave2: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop -datanode-
slave2.out
slave1: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop -datanode-
slave1.out
master: starting secondarynamenode, logging to /usr/local/hadoop/bin/../logs/hadoop-
hadoop-secondarynamenode-master.out
hadoop@master:/usr/local/hadoop$
```

Στα Slaves ξεκινάνε τα ακόλουθα με την παραπάνω εντολή:

```
DataNode
```

Αντίστοιχη εντολή που στον NameNode τερματίζουμε το HDFS cluster είναι η ακόλουθη:

```
hadoop@master:/usr/local/hadoop$ bin/stop-dfs.sh
```

Σε slave, μπορείτε να εξετάσει την επιτυχία ή την αποτυχία αυτής της εντολής εξετάζοντας το log file logs/Hadoop-hduser-datanode-slave.log.

MapReduce Daemons

```
hadoop@master:/usr/local/hadoop$ bin/start-mapred.sh
```

Με την εντολή αυτή ξεκινάει:

- ο JobTracker στο τοπικό μηχάνημα και
- ο TaskTracker σε κάθε μηχάνημα που αναφέρεται στο αρχείο $HADOOP_HOME/conf/slaves.

Στα Slaves ξεκινάνε τα ακόλουθα με την παραπάνω εντολή:

```
TaskTracker
```

Αντίστοιχη εντολή που στον JobTracker τερματίζουμε το MapReduce cluster είναι η ακόλουθη:

```
hadoop@master:/usr/local/hadoop$ bin/stop-mapred.sh
```

Μεταφορά των Αρχείων Εισόδου στο HDFS

Γενική σύνταξη: bin/hadoop dfs –put [file] [hdfs_path]

```
hadoop@master:/usr/local/hadoop/bin/hadoop dfs -put fileName fileName
```

Ο φάκελος fileName θα δημιουργηθεί στο HDFS. Σε περίπτωση που προκύψει πρόβλημα και δεν μπορεί να κάνει τη μεταφορά, τότε περιμένουμε λίγο χρόνο για να βγει ο NameNode από το safe mode. Κατά το χρόνο στον οποίο ο NameNode βρίσκεται σε safe mode οι DataNodes "δηλώνουν" την ύπαρξή τους στον namenode. Την κατάσταση του

cluster και το χρόνο που απομένει για να βγει από το safe mode μπορούμε να παρακολουθήσουμε από το web interface του namenode

Μπορούμε να δούμε τα αρχεία που μεταφέραμε στο HDFS με την εντολή:

Γενική σύνταξη: bin/hadoop dfs –ls [hdfsPath]

```
hadoop@master:/usr/local/hadoop/bin/hadoop dfs –ls /user/$hadoop_user/application/input
```

Διαγραφή των Αρχείων Εισόδου στο HDFS

```
Γενική σύνταξη: bin/hadoop dfs –rmr -skipTrash [hdfsPath]
```

Μεταγλώττιση του Προγράμματος και Δημιουργία jar (Αρχεία Java)

```
hadoop@master:~$ cd /usr/local/hadoop$

hadoop@master:/usr/local/hadoop$ mkdir application_classes

hadoop@master:/usr/local/hadoop$ javac -classpath HADOOP_VERSION-core.jar -d application_classes Application.java

hadoop@master:/usr/local/hadoop$ jar -cvf application.jar -C application_classes/ .
```

Πρόσοχή στην τελεία στο τέλος της εντολής για τη δημιουργία του jar**!**

Εκτέλεση του Προγράμματος Γραμμένη σε Map/Reduce

```
bin/hadoop jar application.jar Application.java <local_input> <local_output0> <local_output1> <keyword>
```

Τα αρχεία που θα προκύψουν μετά τον τερματισμό της φάσης reduce θα αποθηκευτούν στο HDFS σε δύο ξεχωριστούς φακέλους στο local_output0 & local_output1.

Κατά τη διάρκεια εκτέλεσης του προγράμματος μπορούμε να παρακολουθούμε τις διάφορες φάσεις, καθώς και άλλες λεπτομέρειες της εκτέλεσης στα web interfaces του namenode και του jobtracker.

Τρέχοντας Εργασίες στο MapReduce

Τώρα θα τρέξουμε την πρώτη μας Hadoop MapReduce εργασία που υλοποιήσαμε την COMBINER_FINDER.

Το πρόγραμμα θα δέχεται σαν είσοδο αρχεία κειμένου και θα εξάγει αρχεία κειμένου.

Με το προγραμμα μας θα τρεξουμε δυο MR jobs, στο **1ο** MapR job σαν έξοδο θα εξάγουμε όλους τους τίτλους (<title>.....</title>) των urls που έχουν 3 η παραπάνω "a". Όπως βλέπετε και παρακάτω:

Electronics, Cars, Fashion, Collectibles, Coupons and More Online Shopping http___www.ebay.com _.txt

www.carputermania.gr http___ www.carputermani.gr_.txt

Τα περιεχόμενα των URLs αποθηκεύονται σε ξεχωριστά αρχεία σε έναν φάκελο στο Σκληρό Δίσκο (HDD) σε έναν φάκελο με ονομα fileName.

Θα σκέφτεστε γιατί δεν έχουμε ενα αρχείο γεμάτο με διαφορετικά URLs για να το ανεβάζαμε στο HDFS και με αυτό τον τρόπο να τρέχαμε το πρόγραμμα μας έτσι ώστε τα περιεχόμενα που αντιστοιχούν σε κάθε URL να κατέβαιναν μέσω δικτύου.

Καταφύγαμε σε αυτήν τη μέθοδο για να μην εξαρτιόμαστε από την ταχύτητα του δικτύου με αποτέλεσμα να βελτιώσουμε την απόδοση του συστήματος.

Τα περιεχόμενα των URLs τα παίρνουμε με το πρόγραμμα που βρίσκεται στο Παράρτημα Β στο παράδειγμα 1. Η εξοδοι των URLs θα εμφανίζονται με τον παράκατω τρόπο:

http___ www.carputermani.gr_.txt

http___www.ebay.com _.txt

Στο **2ο** MapR job θα εξάγουμε για παράδειγμα να ψάχνει σε όλα αυτά τα περιεχόμενα των URLs μια λέξη που θα του δώσουμε εμείς (**π.χ** Cars) και να εμφανίζει τα αντίστοιχα URLs που περιέχουν τη λέξη αυτή.

Η εξοδοι των URLs θα εμφανίζονται με τον παράκατω τρόπο:

Cars http___ www.carputermani.gr_.txt

Cars http___www.ebay.com _.txt

Κατέβασμα των **Δ**εδομένων **Ε**ισόδου για το **Π**αραδείγμα μας

Θα χρησιμοποιήσουμε ενα πρόγραμμα γραμμένο σε **Java** το οποίο θα τραβάει δίνοντας του ενα .**txt** αρχείο με πολλά διαφορετικά urls το πηγαίο κώδικα των αντίστοιχων urls και θα δημιουργεί έτσι για καθε url και ενα αρχείο με τα περιεχόμενα του πηγαίου κώδικα.

Το κάθε αρχείο text που θα δημιουργείται θα είναι κωδικοποιημένη με τη μορφή **Plain Text UTF-8.**

Ο κώδικας που χρησιμοποιήθηκε για την παραπάνω διαδικασία επισυνάπτεται στο παράρτημα Β στο παράδειγμα 1.

Αντιγραφή των **Δ**εδομένων **Ε**ισόδου στο **HDFS**

Πρίν τρέξουμε την πραγματική εργασία MapReduce, πρέπει πρώτα να αντιγράψουμε τα αρχεία εισόδου απο το τοπικό σύστημα αρχείων μας στο HDFS του Hadoop.

```
hadoop@master:/usr/local/hadoop$ bin/hadoop bin/hadoop dfs –put fileName fileName
hadoop@master:/usr/local/hadoop$ bin/hadoop dfs -ls
Found 2 items
drwxr-xr-x - hadoop supergroup 0 2013-06-13 00:31 /usr/hadoop / fileName
drwxr-xr-x - hadoop supergroup 0 2013-06-13 00:31 /usr/hadoop / titlefinder_out
drwxr-xr-x - hadoop supergroup 0 2013-06-13 00:31 /usr/hadoop / wordfinder_out
hadoop@ master:/usr/local/hadoop$
```

Εκτελέσει **M**ap**R**educe **Ε**ργασίας

Τώρα, τρέχουμε το παραδειγμα της εργασίας του CombineFinder ως εξής:

```
hadoop@master:/usr/local/hadoop$ bin/hadoop jar combinefinder.jar CombineFinder <local_input> <local_output0> <local_output1> <keyword>
```

Όπου ο κατάλογος <local_input> ανήκει στο HDFS και είναι ο κατάλογος στον οποίο ανεβάσαμε τα δεδομένα εισοδου, τα οποία θα τα διαβάσει, θα τα επεξεργαστεί και θα αποθηκεύσει το αποτέλεσμα μετά το τέλος της εκτέλεσης του προγράμματος στον κατάλογο HDFS το οποίο θα περιέχει και τα δεδομένα εξόδου.

hadoop@master:/usr/local/hadoop$ **bin/hadoop jar combinefinder.jar CombineFinder fileName titlefinder_out wordfinder_out Cars**

13/08/18 01:00:21 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications should implement Tool for the same.

13/08/18 01:00:21 INFO input.FileInputFormat: Total input paths to process : 11

13/08/18 01:00:22 INFO mapred.JobClient: Running job: job_201308172336_000**5**

13/08/18 01:00:23 INFO mapred.JobClient: map 0% reduce 0%

13/08/18 01:00:32 INFO mapred.JobClient: map 9% reduce 0%

13/08/18 01:00:35 INFO mapred.JobClient: map 18% reduce 0%

13/08/18 01:00:37 INFO mapred.JobClient: map 27% reduce 0%

13/08/18 01:00:40 INFO mapred.JobClient: map 45% reduce 0%

13/08/18 01:00:43 INFO mapred.JobClient: map 63% reduce 0%

13/08/18 01:00:44 INFO mapred.JobClient: map 81% reduce 0%

13/08/18 01:00:45 INFO mapred.JobClient: map 100% reduce 0%

13/08/18 01:00:54 INFO mapred.JobClient: map 100% reduce 27%

13/08/18 01:01:00 INFO mapred.JobClient: map 100% reduce 100%

13/08/18 01:01:0**2** INFO mapred.JobClient: Job complete: job_201308172336_0005

13/08/18 01:01:02 INFO mapred.JobClient: Counters: 17

13/08/18 01:01:02 INFO mapred.JobClient: Job Counters

13/08/18 01:01:02 INFO mapred.JobClient: *Launched reduce tasks=1*

13/08/18 01:01:02 INFO mapred.JobClient: *Launched map tasks=11*

13/08/18 01:01:02 INFO mapred.JobClient: Data-**local** map tasks=11

13/08/18 01:01:02 INFO mapred.JobClient: FileSystemCounters

13/08/18 01:01:02 INFO mapred.JobClient: FILE_BYTES_READ=248

13/08/18 01:01:02 INFO mapred.JobClient: HDFS_BYTES_READ=353510

13/08/18 01:01:02 INFO mapred.JobClient: FILE_BYTES_WRITTEN=908

13/08/18 01:01:02 INFO mapred.JobClient: HDFS_BYTES_WRITTEN=236

13/08/18 01:01:02 INFO mapred.JobClient: Map-Reduce Framework

13/08/18 01:01:02 INFO mapred.JobClient: *Reduce input* ***groups****=3*

13/08/18 01:01:02 INFO mapred.JobClient: Combine output records=3

13/08/18 01:01:02 INFO mapred.JobClient: Map input records=6134

13/08/18 01:01:02 INFO mapred.JobClient: Reduce shuffle bytes=302

13/08/18 01:01:02 INFO mapred.JobClient: Reduce output records=3

13/08/18 01:01:02 INFO mapred.JobClient: Spilled Records=6

13/08/18 01:01:02 INFO mapred.JobClient: Map output bytes=236

13/08/18 01:01:02 INFO mapred.JobClient: Combine input records=3

13/08/18 01:01:02 INFO mapred.JobClient: Map output records=3 *(key value pairs that Mapper outputs)*

13/08/18 01:01:02 INFO mapred.JobClient: Reduce input records=3

Total execution time: 0

13/08/18 01:01:03 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications should implement Tool for the same.

13/08/18 01:01:03 INFO input.FileInputFormat: Total input paths to process : 11

13/08/18 01:01:03 INFO mapred.JobClient: Running job: job_201308172336_000**6**

13/08/18 01:01:04 INFO mapred.JobClient: map 0% reduce 0%

13/08/18 01:01:20 INFO mapred.JobClient: map 18% reduce 0%

13/08/18 01:01:22 INFO mapred.JobClient: map 27% reduce 0%

13/08/18 01:01:23 INFO mapred.JobClient: map 45% reduce 0%

13/08/18 01:01:27 INFO mapred.JobClient: map 54% reduce 0%

13/08/18 01:01:30 INFO mapred.JobClient: map 63% reduce 0%

13/08/18 01:01:32 INFO mapred.JobClient: map 81% reduce 0%

13/08/18 01:01:33 INFO mapred.JobClient: map 96% reduce 0%

```
13/08/18 01:01:36 INFO mapred.JobClient:  map 100% reduce 0%
13/08/18 01:01:38 INFO mapred.JobClient:  map 100% reduce 21%
13/08/18 01:01:47 INFO mapred.JobClient:  map 100% reduce 100%
13/08/18 01:01:49 INFO mapred.JobClient: Job complete: job_201308172336_0006
13/08/18 01:01:49 INFO mapred.JobClient: Counters: 17
13/08/18 01:01:49 INFO mapred.JobClient:   Job Counters
13/08/18 01:01:49 INFO mapred.JobClient:     Launched reduce tasks=1
13/08/18 01:01:49 INFO mapred.JobClient:     Launched map tasks=11
13/08/18 01:01:49 INFO mapred.JobClient:     Data-local map tasks=11
13/08/18 01:01:49 INFO mapred.JobClient:   FileSystemCounters
13/08/18 01:01:49 INFO mapred.JobClient:     FILE_BYTES_READ=74
13/08/18 01:01:49 INFO mapred.JobClient:     HDFS_BYTES_READ=353510
13/08/18 01:01:49 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=560
13/08/18 01:01:49 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=64
13/08/18 01:01:49 INFO mapred.JobClient:   Map-Reduce Framework
13/08/18 01:01:49 INFO mapred.JobClient:     Reduce input groups=2
13/08/18 01:01:49 INFO mapred.JobClient:     Combine output records=2
13/08/18 01:01:49 INFO mapred.JobClient:     Map input records=6134
13/08/18 01:01:49 INFO mapred.JobClient:     Reduce shuffle bytes=134
13/08/18 01:01:49 INFO mapred.JobClient:     Reduce output records=2
13/08/18 01:01:49 INFO mapred.JobClient:     Spilled Records=4
13/08/18 01:01:49 INFO mapred.JobClient:     Map output bytes=64
13/08/18 01:01:49 INFO mapred.JobClient:     Combine input records=2
13/08/18 01:01:49 INFO mapred.JobClient:     Map output records=2
13/08/18 01:01:49 INFO mapred.JobClient:     Reduce input records=2
Total execution time: 0
```

Στην παραπάνω εκτέλεση αποτυπώνονται δύο εργασίες (*jobs*) που εκτελούνται μαζί με κάποια στατιστικά δεδομένα για αυτές, όπως για παράδειγμα ο αριθμός των δεδομένων εισόδου/εξόδου στις φάσεις Map & Reduce, καθώς και τον αριθμό των tasks που ξεκίνησαν. Οι πληροφορίες αυτές φαίνονται και στο web interface του JobTracker (*http://master:50030/jobtracker.jsp*), όπως βλέπουμε στο **Σχήμα (7.5) .**

Ελέγξτε αν το αποτέλεσμα έχει αποθηκευτή επιτυχώς στον κατάλογο HDFS με τον παρακάτω τρόπο.

```
hadoop@master:/usr/local/hadoop$ bin/hadoop dfs -ls titlefinder_out/* wordfinder_out/*
Found 2 items
drwxr-xr-x - hadoop supergroup 0 2013-06-13 00:31 /usr/hadoop/titlefinder_out/_logs/history
drwxr-xr-x - hadoop supergroup 0 2013-06-13 00:40 /usr/hadoop/titlefinder_out /part-r-00000
drwxr-xr-x - hadoop supergroup 0 2013-06-13 00:31 /usr/hadoop/wordfinder_out/_logs/history
drwxr-xr-x - hadoop supergroup 0 2013-06-13 00:40 /usr/hadoop/wordfinder_out /part-r-00000
hadoop@master:/usr/local/hadoop$
```

<u>**A**νάκτηση των **A**ποτελεσμάτων της **E**ργασίας από το **HDFS**</u>

Θα μπορουσαμε να δουμε τα αποτελεσματα της MapReduce εργασιας απευθειας απο το HDFS και με αλλο τροπο:

```
hadoop@master:/usr/local/hadoop$ bin/hadoop dfs –cat titlefinder_out/part-r-00000

Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & more
        www.amazon.com.txt
Electronics, Cars, Fashion, Collectibles, Coupons and More Online Shopping | eBay
        http___www.ebay.com _.txt
```

www.carputermania.gr **www.carputermania.gr.txt**

hadoop@master:/usr/local/hadoop$ bin/hadoop dfs –cat wordfinder_out/part-r-00000

Cars http___ www.carputermani.gr_.txt

hadoop@master:/usr/local/hadoop$ bin/hadoop dfs –cat "titlefinder_out/part-*" "wordfinder_out/part-*"

Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & more **www.amazon.com.txt**

Electronics, Cars, Fashion, Collectibles, Coupons and More Online Shopping | eBay **http___www.ebay.com _.txt**

www.carputermania.gr **www.carputermania.gr.txt**

Cars http___ www.carputermani.gr_.txt

ΠΑΡΑΡΤΗΜΑ Β΄ – ΚΩΔΙΚΑΣ ΠΑΡΑΔΕΙΓΜΑΤΩΝ

ΚΩΔΙΚΑΣ ΠΑΡΑΔΕΙΓΜΑΤΟΣ 1 – SOURCE CODE

```
import java.io.IOException;
import java.io.PrintWriter;
import java.net.URL;
import java.util.Scanner;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.InputStream;

public class SourceCode{

        public static void main(String args[]) throws IOException
        {
          // Read in input file
          InputStream input = new FileInputStream ( ("C:/InputURLs/urls.txt") );
                    try {
      Scanner freader = new Scanner(input);
      while ( freader.hasNextLine() ) {
        String url = freader.nextLine();
        System.out.println(url);

String out = new Scanner(new URL(url).openStream(), "UTF-8").useDelimiter("\\A").next();
```

```
                System.out.println(out);
PrintWriter pwout = new PrintWriter( new FileWriter (url.replaceAll("[^-\\w.]",
"_")+".txt",true) );
                pwout.println(out);
            }
            freader.close();
            } catch (IOException ex) {
                ex.printStackTrace();
            }     System.out.println("Done");
}
}
```

ΚΩΔΙΚΑΣ ΠΑΡΑΔΕΙΓΜΑΤΟΣ 2 – COMBINE FINDER ΣΤΗΝ ΠΡΑΞΗ ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ HADOOP 0.20.2 API

```
import java.io.IOException;

import java.util.regex.Matcher;

import java.util.regex.Pattern;


import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.FileSplit;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;


public class CombineFinder

{

    // Όταν επεκτείνουμε το MapReduce mapper class ορίζουμε τις key/value τύπους για εισόδους και εξόδους μας.

    //Map Class

     public static class WordFinderMap

          extends Mapper<LongWritable, Text, Text, Text>
```

```
/ / υποστηριζόμενοι τύποι δεδομένων στο Hadoop
private static Text word = new Text();
  private static Text location = new Text();


 //Map function
public void map(LongWritable key, Text val, Context context)
        throws IOException, InterruptedException
{
          // Get the name of the file from the inputsplit in the context
          String fileName = ((FileSplit)
          context.getInputSplit()).getPath().getName();
          location.set(fileName);
          // assuming this is the file content
          String line = val.toString();
          // isolate keyword
          String keyword = context.getConfiguration().get("KEYWORD");
          System.out.println("----------" + keyword);
          Pattern pattern = Pattern.compile(String.format(".*%s.*", keyword));
          Matcher matcher = pattern.matcher(line.toString());
          boolean result = matcher.find();
          if (result)
          {
                word.set(keyword);
        // add keyword to output along with filename
        context.write(location, word);
          }
```

```
        }
    } // WordFinderMapper end

    //Reduce Class
    public static class WordFinderCombiner
            extends Reducer<Text, Text, Text, Text>
    {

    // Reduce function
    public void reduce(Text location, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {
System.out.println("------------------before COMBINER>>>>>>>>>>>>>>>>>>>>>>>>");
                Text keyword = null;
                for (Text val : values)
                {
                    keyword = val;
System.out.println(String.format("combine: %s - %s", location, keyword));
                    break;
                }
                context.write(location, keyword );
System.out.println("<<<<<<<<<<<<<<<<<<<< after COMBINER----------------------");

        }
    }
```

```
/**
 * The actual main() method for our program; this is the "driver" for the
 * MapReduce job.
 *
 * @throws ClassNotFoundException
 * @throws InterruptedException
 */
private static boolean excuteWordFinderJob(String[] args)
        throws IOException,InterruptedException, ClassNotFoundException
{
         // Start counting time
        long startTime = System.currentTimeMillis();

        //Create configuration
        Configuration conf = new Configuration();
        conf.set("KEYWORD", args[3]);
        // Create job
        Job job = new Job(conf, "WordFinder");
        job.setJarByClass(CombineFinder.class);
        // setting the class names
        job.setMapperClass(WordFinderMap.class);
        job.setCombinerClass(WordFinderCombiner.class);
        // Set the output Key type for the Mapper
        job.setMapOutputKeyClass(Text.class);
       // Set the output Value type for the Mapper
        job.setMapOutputValueClass(Text.class);
       // Set the output Key type for the Reducer
```

```
        job.setOutputKeyClass(Text.class);
        // Set the output Value type for the Reducer
        job.setOutputValueClass(Text.class);
        //File Input/Output argument passed as a command line argument
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[2]));
        boolean result = job.waitForCompletion(true);
      // Stop counting time
        long endTime = System.currentTimeMillis();
        int minu = (int) (((endTime - startTime) / (1000*60)) % 60);
        System.out.println("Total execution time: " + minu);
        return result;

  } // main end

     //Map Class
    public static class TitleFinderMap
      extends Mapper<LongWritable, Text, Text, Text>
{
        private final static Text word = new Text();
        private final static Text location = new Text();

        private boolean checkTitleIfMatched(String title)
        {
                Pattern pattern = Pattern.compile(".*[Aa].*[Aa].*[Aa].*");
                Matcher matcher = pattern.matcher(title.toString());
```

DD

```
            boolean result = matcher.find();


            return result;
    }

    // map function
      public void map(LongWritable key, Text val, Context context)
            throws IOException, InterruptedException
    {
            String fileName = ((FileSplit) context.getInputSplit()).getPath()
                              .getName();
            System.out.println("file name: " + fileName);
            location.set(fileName);
            // assuming this is the file content
            String line = val.toString();
            System.out.println("file content: " + line);

            // isolate page title
            String titleTag = "<title>";
            String endTitleTag = "</title>";
            int titleStart = line.indexOf(titleTag);
            int titleEnd = line.indexOf(endTitleTag);

            if (titleStart != -1 && titleEnd != -1)
            {
            String title = line.substring(titleStart + titleTag.length(),titleEnd);
```

```
                if (checkTitleIfMatched(title))
                  {
                  System.out.println(String.format("%s - %s", title, location));
                  // add matched title to output along with filename
                  word.set(title);
                // The mechanism through which we output the key/value pair that we
                want to pass to the reducer
                context.write(word, location);
                        }
                }
        }
} // TitleFinderMapper end

      // Reduce class
public static class TitleFinderReduce
      extends Reducer<Text, Text, Text, Text>
{

       // Reduce function
       public void reduce(Text key, Iterable<Text> values, Context context)
              throws IOException, InterruptedException
       {
              for (Text val : values)
              {
                     context.write(key, val);
              }
```

```
        }
    } // TitleFinderReducer end

    /**
     * The actual main() method for our program; this is the "driver" for the
     * MapReduce job.
     *
     * @throws ClassNotFoundException
     * @throws InterruptedException
     */
    private static boolean executeTitleFinderJob(String[] args)
            throws IOException,InterruptedException, ClassNotFoundException
    {

            long startTime = System.currentTimeMillis();
            Configuration conf = new Configuration();
            Job job = new Job(conf, "TitleFinder");
            job.setJarByClass(CombineFinder.class);
            job.setMapperClass(TitleFinderMap.class);
            job.setReducerClass(TitleFinderReduce.class);
            job.setCombinerClass(TitleFinderReduce.class);
            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(Text.class);
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(Text.class);
            FileInputFormat.addInputPath(job, new Path(args[0]));
```

```
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        boolean result = job.waitForCompletion(true);
        long endTime = System.currentTimeMillis();
        int minu = (int) (((endTime - startTime) / (1000*60)) % 60);
        System.out.println("Total execution time: " + minu);
        return result;

    }
    /**
     * @param args
     * @throws ClassNotFoundException
     * @throws InterruptedException
     * @throws IOException
     */
      // "Driver" for the MapReduce job.
    public static void main(String[] args) throws IOException, InterruptedException,
    ClassNotFoundException
    {
        if(args.length < 4)
        {
    System.err.println("Usage : hadoop jar combinefinder.jar CombineFinder
    <local_input>  <local_output0> " + "<local_output1> <keyword>");
    System.exit(0);
        }
    executeTitleFinderJob(args);
    excuteWordFinderJob(args);      } }
```