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

BIG DATA

Η ΑΝΑΓΚΗ - ΤΟ ΠΡΟΒΛΗΜΑ - Η ΤΕΧΝΟΛΟΓΙΑ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΤΗΣ
ΕΙΡΗΝΗΣ ΖΑΡΟΓΙΑΝΝΟΥ
ΑΜ: 05/2872

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:
ΔΗΜΗΤΡΙΟΣ ΑΧΙΛ. ΔΕΡΒΟΣ

ΘΕΣΣΑΛΟΝΙΚΗ 2013

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΡΟΛΟΓΟΣ

Η παρούσα πτυχιακή εργασία πραγματοποιήθηκε στα πλαίσια των σπουδών μου στο Τμήμα Πληροφορικής, της σχολής Τεχνολογικών Εφαρμογών του Αλεξάνδρειου Τεχνολογικού Εκπαιδευτικού Ιδρύματος Θεσσαλονίκης, υπό την επίβλεψη του κ. Δέρβου Δημήτρη, καθηγητή του ΑΤΕΙ.

Ιδιαίτερες ευχαριστίες απευθύνονται στον επιβλέποντα καθηγητή για την καθοδήγησή του σε όλη τη διάρκεια της πτυχιακής και την παροχή βιβλιογραφικών και διαδικτυακών πηγών.

ΠΕΡΙΛΗΨΗ

Καθώς ο κόσμος της πληροφορίας γίνεται κοινός τόπος για όλο και μεγαλύτερο μέρος του πληθυσμού παγκοσμίως, η ανάγκη για αποδοτική διαχείριση μεγάλου όγκου ετερογενών δεδομένων γίνεται επιτακτική. Τα δεδομένα που διακινούνται καθημερινά μέσω του διαδικτύου απαιτούν φιλτράρισμα ώστε στους τελικούς χρήστες να φθάσουν μόνο όσα είναι απαραίτητα, ενώ ταυτόχρονα κρίνεται αναγκαίος ο σχεδιασμός τεχνικών και μεθόδων που θα επιτρέψουν τη βέλτιστη αποθήκευση, διαχείριση, αναζήτηση και ανάκτηση των δεδομένων αυτών, με απώτερο σκοπό την εξόρυξη χρήσιμης πληροφορίας και γνώσης από αυτά. Η αύξηση των δεδομένων (μεγάλα δεδομένα) οδήγησε στην ανάγκη ύπαρξης συστημάτων κατάλληλων για τη διαχείρισή τους. Μία πλατφόρμα ανοιχτού κώδικα το Hadoop αναπτύχθηκε για την κάλυψη αυτών των αναγκών. Έτσι λοιπόν, τα κατανεμημένα συστήματα επέφεραν τη λύση με τη χρήση υπολογιστικών κόμβων χαμηλού κόστους, μοιράζοντας παράλληλα το φόρτο εργασίας και επιτυγχάνοντας γρήγορα αποτελέσματα.

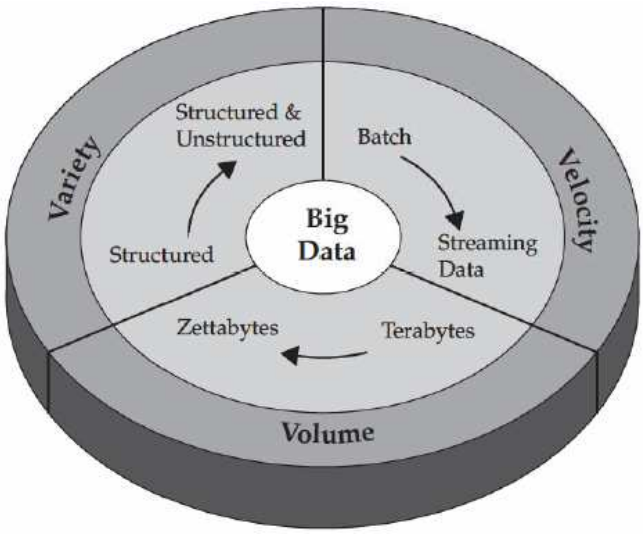
1. Τι εννοούμε με τον όρο “Big Data”

Με τον όρο Big Data (μεγάλα δεδομένα) εννοούμε εκείνα τα σύνολα δεδομένων (datasets) των οποίων το μέγεθος είναι πέρα των δυνατοτήτων των κοινώς χρησιμοποιούμενων εργαλείων λογισμικού και υλικού για τη σύλληψη, διαχείριση και επεξεργασία τους μέσα σε ένα αποδεκτό χρονικό διάστημα [4][5]. Το μέγεθος τους σήμερα κυμαίνεται από μερικές δεκάδες terabytes μέχρι πολλά petabytes. Και πηγές τέτοιων συνόλων δεδομένων αποτελούν τα: ERP data, Web logs, αισθητήρες δικτύου, συναλλαγές, κείμενα, XML, JSON, αρχεία εικόνας, βίντεο κ.α.

1.1 Τα χαρακτηριστικά των Big Data

Οι συνεχώς αυξανόμενες κερδοσκοπικές απαιτήσεις των επιχειρήσεων για την εξόρυξη έγκυρης πληροφορίας πέραν τον κλασικών μεθόδων (DataWarehouse, Business Intelligence tools), οδήγησε στην ανάγκη εκμετάλλευσης πάσης φύσεως δεδομένων. Έτσι λοιπόν, η αξία της πληροφορίας έγκειται τόσο σε δομημένα όσο και σε αδόμητα και ημιδομημένα δεδομένα. Για να κατανοήσουμε βαθύτερα τον όρο μεγάλα δεδομένα βλέποντας την Εικόνα 1.1, θα αναλύσουμε τις λέξεις κλειδιά που τα χαρακτηρίζουν:

- **Όγκος (volume)** – Είναι πλέον μεγάλος ο όγκος των δεδομένων που αποθηκεύονται σήμερα παγκοσμίως με την έξαρση της τεχνολογίας των πληροφοριών και των τηλεπικοινωνιών. Βεβαίως, πολλά από αυτά τα δεδομένα σήμερα δεν έχουν αναλυθεί καθόλου, όπου αυτό είναι ένα ακόμη πρόβλημα που πρέπει να διαχειριστούμε. Σήμερα, το Twitter δημιουργεί περισσότερο από 7 terabytes από δεδομένα κάθε μέρα, το Facebook 10 TB, και μερικές επιχειρήσεις δημιουργούν terabytes από δεδομένα κάθε ώρα στην ημέρα, στο χρόνο. Είναι γνωστό ότι, κάποιες επιχειρήσεις για να διαχειριστούν τον αυξανόμενο όγκο αποθηκευμένων δεδομένων χρησιμοποιούν συστοιχίες υπολογιστών άλλες τοπικά στο δίκτυό τους και άλλες στο “cloud”.



Εικόνα 1.1 Η IBM χαρακτηρίζει τα δεδομένα από τον όγκο, την ταχύτητα και την ποικιλία τους (V^3).

- **Ποικιλία (variety)** – Ο όγκος που σχετίζεται με το φαινόμενο Big Data μας οδηγεί σε νέες προκλήσεις για τα κέντρα δεδομένων σε μία άλλη διάσταση, την ποικιλία τους. Με τους αισθητήρες, τις έξυπνες συσκευές, καθώς και τις τεχνολογίες κοινωνικής δικτύωσης, τα δεδομένα στις επιχειρήσεις έχουν γίνει πολύπλοκα, επειδή περιλαμβάνουν όχι μόνο παραδοσιακά σχεσιακά δεδομένα, αλλά επίσης πρωτογενή δεδομένα (raw), ημιδομημένα και αδόμητα δεδομένα όπως: ιστοσελίδες, αρχεία καταγραφής ιστού (συμπεριλαμβάνοντας click-stream data), search indexes, φόρουμ κοινωνικών μέσων, έγγραφα, e-mail, δεδομένα αισθητήρων από ενεργά και παθητικά συστήματα κ.α. Τα παραδοσιακά συστήματα (RDBMS) δεν μπορούν όλα αυτά τα δεδομένα να τα διαχειριστούν, δηλαδή να τα αποθηκεύσουν και να τα επεξεργαστούν με παραδοσιακά εργαλεία ανάλυσης (πλατφόρμες λογισμικού) για την εξόρυξη πληροφορίας. Επιπλέον, θα πρέπει να έχουμε υπόψην τη σημαντικότητα των δεδομένων που δυναμικά αλλάζουν (όπως σε συστήματα πρόβλεψης καιρού) που δεν είναι αυστηρά δομημένα με βάση κάποιο σχήμα. Και με αυτή την προσέγγιση, οι επιχειρήσεις εστιάζουν στην ανάλυση, τόσο των σχεσιακών δεδομένων όσο και των μη σχεσιακών δεδομένων.

- **Ταχύτητα (velocity)** – Όπως ο μεγάλος όγκος και η ποικιλία των δεδομένων που συλλέγουμε και αποθηκεύουμε έχουν αλλάξει, τόσο και η ταχύτητα με την οποία δημιουργούνται έχει αλλάξει και είναι σημαντικό να τη διαχειριστούμε. Η ταχύτητα περιγράφει το πόσο γρήγορα τα δεδομένα καταφθάνουν και αποθηκεύονται. Αντί για τον περιορισμό του ορισμού της ταχύτητας δεδομένων σχετικά με τα αποθετήρια δεδομένων, είναι αναγκαίο να συμπεριλάβουμε και τα δεδομένα που βρίσκονται σε κίνηση – την ταχύτητα με την οποία τα δεδομένα ρέουν (streaming). Μετά από όλα αυτά, σήμερα οι επιχειρήσεις έχουν να κάνουν με petabytes αντί για terabytes, όπως με την αύξηση των αισθητήρων RFID και άλλων ροών πληροφορίας, με συνεχή ροή δεδομένων που κατέστησε αδύνατο τον χειρισμό τους από τα παραδοσιακά συστήματα. Εστιάζοντας στη ταχύτητα, αυτό μας οδηγεί σε ένα μικρό προβάδισμα απέναντι στον ανταγωνισμό, που αυτό μπορεί να σημαίνει εντοπισμό μιας τάσης, ενός προβλήματος ή μίας ευκαιρίας, λίγα δευτερόλεπτα ή χιλιοστά του δευτερολέπτου πριν από κάποιον άλλο. Επιπρόσθετα, όλο και περισσότερα δεδομένα που δημιουργούνται σήμερα έχουν μικρό χρόνο ζωής, για τα οποία η ανάλυσή τους θα πρέπει να γίνει σε πραγματικό χρόνο. Γι' αυτό, το λόγο αναδύθηκε η έννοια του "stream computing". Στην παραδοσιακή επεξεργασία, ας σκεφτούμε την εκτέλεση ερωτημάτων σχετικά με στατικά δεδομένα: για παράδειγμα, το ερώτημα "Δείξε μου όλους τους ανθρώπους που ζουν στη ζώνη πλημμύρων στο New Jersey", θα έχει ως αποτέλεσμα μία λίστα προειδοποίησης ενός καιρικού σχεδίου. Αλλά με το stream computing, μπορούμε να εκτελούμε συνεχώς το ίδιο ερώτημα, για το ποιοι άνθρωποι βρίσκονται "στη ζώνη πλημμύρων στο New Jersey", παίρνοντας έτσι συνεχώς ενημερωμένα αποτελέσματα, διότι οι πληροφορίες τοποθεσίας από GPS δεδομένα ανανεώνονται σε πραγματικό χρόνο. Η αποτελεσματική αντιμετώπιση των μεγάλων δεδομένων απαιτεί να εκτελούμε αναλύσεις σε σχέση με τον όγκο και την ποικιλία των δεδομένων, ενώ εξακολουθούν να είναι σε κίνηση (data in motion), και όχι μόνο σε κατάσταση ηρεμίας (data at rest).

1.2 Η πλατφόρμα των Big Data

Όπως με το Data Warehouse και με τις πλατφόρμες IT, η υποδομή των Big Data έχει συγκεκριμένες απαιτήσεις, όπου ο στόχος είναι η εύκολη ενσωμάτωση των μεγάλων δεδομένων με τα δεδομένα των επιχειρήσεων, ώστε να γίνουν αναλύσεις εις βάθος [17].

- **Απαιτήσεις υποδομής** – Οι απαιτήσεις για μία Big Data υποδομή ορίζονται με βάση την συλλογή, οργάνωση και ανάλυσή τους.

- **Συλλογή των Big Data** – Επειδή τα μεγάλα δεδομένα είναι κυρίως ροές δεδομένων με υψηλή ταχύτητα και μεγάλη ποικιλία, η υποδομή που απαιτείται για την απόκτηση των μεγάλων δεδομένων θα πρέπει να παρέχει μία χαμηλή προβλεπόμενη καθυστέρηση. Κατά την εισαγωγή των δεδομένων, αλλά και στην εκτέλεση απλών σύντομων ερωτημάτων πρέπει η Big Data υποδομή να είναι σε θέση να χειριστεί μεγάλου όγκου δεδομένα, σε κατανεμημένο περιβάλλον υποστηρίζοντας ευέλικτες και δυναμικές δομές δεδομένων.

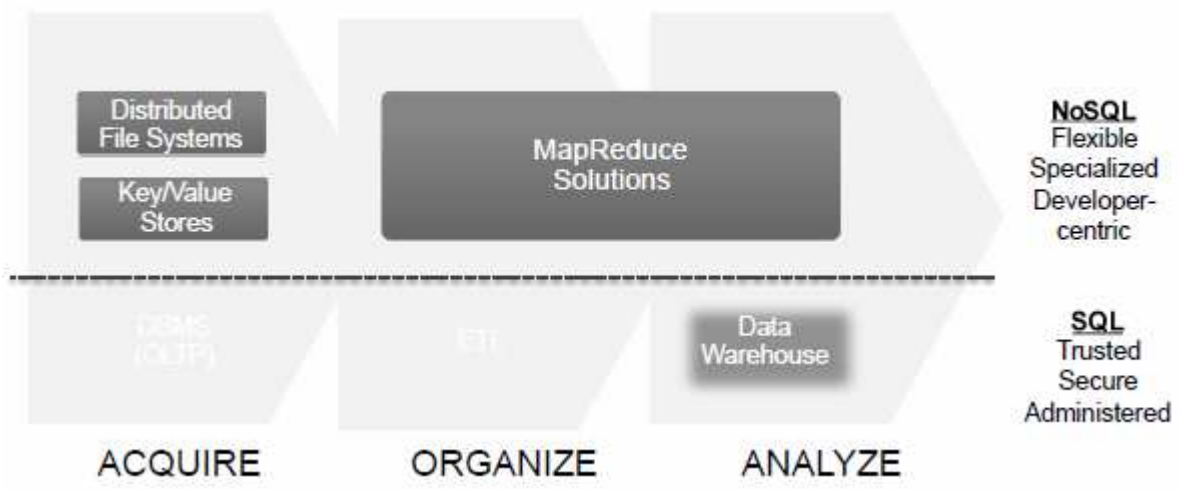
  Οι NoSQL (Not only SQL) βάσεις δεδομένων είναι εκείνες που συχνά χρησιμοποιούνται για τη απόκτηση και αποθήκευση μεγάλων δεδομένων. Ταιριάζουν για δυναμικές δομές δεδομένων που είναι εξαιρετικά επεκτάσιμες. Τα δεδομένα που αποθηκεύονται σε μια NoSQL βάση δεδομένων είναι μεγάλης ποικιλίας και αυτό λόγω της απλοποίησης της σύλληψης των δεδομένων, χωρίς κατηγοριοποίηση και έλεγχό τους με βάση κάποιο σχήμα. Για παράδειγμα, οι NoSQL βάσεις δεδομένων συχνά χρησιμοποιούνται για να συλλέγουν και να αποθηκεύουν δεδομένα των μέσων κοινωνικής δικτύωσης. Όταν οι χρήστες των μέσων αυτών κάνουν συχνά αλλαγές στις εφαρμογές τους, οι υποκείμενες δομές αποθήκευσης διατηρούνται απλές. Αντί για το σχεδιασμό ενός σχήματος με τις σχέσεις μεταξύ των οντοτήτων, αυτές οι απλές δομές περιέχουν ένα σημαντικό κλειδί για να εντοπιστεί το σημείο δεδομένων και ένα χώρο περιεχομένου όπου είναι αποθηκευμένα τα σχετικά δεδομένα. Αυτή η απλή και δυναμική δομή επιτρέπει τις αλλαγές να λάβουν χώρα χωρίς δαπανηρές αναδιοργανώσεις σε επίπεδο αποθήκευσης.

- **Η οργάνωση των Big Data** – Σε μία Data Warehouse, η οργάνωση των δεδομένων ονομάζεται ενοποίηση (data integration). Επειδή υπάρχει τόσο μεγάλος όγκος από δεδομένα, υπάρχει η τάση να οργανώνονται τα δεδομένα αυτά στην αρχική θέση αποθήκευσής τους, έτσι ώστε να εξοικονομούμε χρόνο και χρήμα, όταν δεν κινούνται γύρω από μεγάλους όγκους δεδομένων. Η υποδομή που απαιτείται για την οργάνωση των Big Data θα πρέπει να μπορεί να επεξεργαστεί και να χειριστεί τα δεδομένα στην αρχική τους θέση αποθήκευσης με υψηλή ισχύ διεκπεραίωσης (throughput). Η πλατφόρμα Apache Hadoop είναι μία από τις καινοτόμες τεχνολογίες που οργανώνουν και επεξεργάζονται τα Big Data, διατηρώντας τα δεδομένα στην αρχική θέση αποθήκευσής τους στη συστοιχία. Το κατανεμημένο σύστημα αρχείων του Hadoop (HDFS) είναι ένα σύστημα μακράς διάρκειας αποθήκευσης. Για παράδειγμα, η επεξεργασία των web logs από τα MapReduce προγράμματα που τρέχουν στη συστοιχία, παράγει συσσωματωμένα (aggregated) αποτελέσματα στην ίδια συστοιχία. Αυτά τα συσσωματωμένα αποτελέσματα φορτώνονται σε ένα σχεσιακό DBMS σύστημα.

- **Ανάλυση των Big Data** – Η υποδομή που απαιτείται για την ανάλυση των Big Data πρέπει να υποστηρίζει στατιστικές αναλύσεις και εργαλεία εξόρυξης πληροφορίας, σε μία ευρύτερη ποικιλία τύπων δεδομένων που είναι αποθηκευμένα σε διαφορετικά συστήματα. Πιο σημαντικό όμως είναι η δυνατότητα ανάλυσης των Big Data σε συνδυασμό με τα παραδοσιακά σχεσιακά δεδομένα.

Σήμερα υπάρχουν πάρα πολλές ανοιχτού κώδικα βάσεις δεδομένων για την απόκτηση και αποθήκευση των μεγάλων δεδομένων. Το Hadoop αναδεικνύεται ως το κύριο σύστημα για την οργάνωση μεγάλων δεδομένων και είναι η επέκταση της εμβέλειας των σχεσιακών βάσεων δεδομένων σε λιγότερο δομημένα δεδομένα, για την ανάλυση μεγάλων δεδομένων. Αυτά τα νέα συστήματα έχουν δημιουργήσει ένα διαιρεμένο φάσμα λύσεων που αποτελείται από NoSQL λύσεις για ευέλικτο και εξειδικευμένο προγραμματισμό και από SQL λύσεις, δηλαδή την διαχείριση, την ασφάλεια και την αξιοπιστία των σχεσιακών συστημάτων διαχείρισης βάσεων δεδομένων (RDBMS) Εικόνα 1.2.

Τα NoSQL συστήματα είναι σχεδιασμένα για να αποκτούν όλα τα δεδομένα, χωρίς την κατηγοριοποίηση και την ανάλυσή τους κατά την είσοδο στο σύστημα, και ως εκ τούτο τα δεδομένα ποικίλουν. Στα SQL συστήματα, από την άλλη πλευρά τοποθετούνται τυπικά δεδομένα σε σαφώς καθορισμένες δομές που επιβάλλουν σχετικά μεταδεδομένα για τα δεδομένα που συλλαμβάνονται ώστε να εξασφαλιστεί η συνοχή και η επικύρωση των δεδομένων.



Εικόνα 1.2 Διαιρούμενο φάσμα λύσεων.

Τα κατανεμημένα συστήματα αρχείων και οι αποθήκες συναλλαγών χρησιμοποιούνται για να συλλάβουν τα δεδομένα και είναι γενικά σύμφωνα με τις απαιτήσεις που αναφέραμε παραπάνω σε αυτό το έγγραφο. Για την διερμηνεία και την παραγωγή πληροφορίας από τα δεδομένα αυτά χρησιμοποιείται ένα πρόγραμμα το MapReduce. Τα προγράμματα MapReduce είναι προσαρμοσμένα γραπτά προγράμματα που τρέχουν παράλληλα σε κατανεμημένους κόμβους δεδομένων.

Οι αποθήκες συναλλαγών ή οι NoSQL βάσεις δεδομένων είναι εφαρμογές OLTP (On-Line Transactional Processing) στο κόσμο των μεγάλων δεδομένων, οι οποίες έχουν

βελτιστοποιηθεί για τη πολύ γρήγορη συλλογή δεδομένων και την απλότητα των ερωτημάτων. Οι NoSQL βάσεις δεδομένων παρέχουν πολύ γρήγορη απόδοση διότι τα δεδομένα που συλλαμβάνονται αποθηκεύονται γρήγορα με ένα αναγνωριστικό κλειδί αντί να διερμηνεύονται και να μπαίνουν σε ένα σχήμα. Με αυτό τον τρόπο οι NoSQL βάσεις δεδομένων μπορούν ταχέως να αποθηκεύουν μεγάλο αριθμό συναλλαγών.

Ωστόσο, λόγω της μεταβαλλόμενης φύσης των δεδομένων στη βάση δεδομένων NoSQL , κάθε προσπάθεια οργάνωσης των δεδομένων απαιτεί προγραμματισμό για τη διερμηνεία της λογικής αποθήκευσης που χρησιμοποιείται. Αυτό, σε συνδυασμό με την έλλειψη υποστήριξης για πολύπλοκα ερωτήματα, καθιστά δύσκολο στους χρήστες να εξάγουν αξία από τα δεδομένα σε μία βάση δεδομένων NoSQL.

Για να αξιοποιήσουμε στο έπακρο τη NoSQL λύση και να τη στρέψουμε από εξειδικευμένη, προγραμματιστική λύση σε λύση για την επιχείρηση, θα πρέπει να συνδυαστεί με SQL λύσεις σε μία ενιαία αποδεδειγμένη υποδομή, που ικανοποιεί τις απαιτήσεις διαχείρισης και την ασφάλεια των επιχειρήσεων σήμερα.

2. Hadoop

Στις μέρες μας, περιβαλλόμαστε ως επί το πλείστων από δεδομένα. Οι χρήστες του διαδικτύου μεταφορτώνουν videos, τραβούν φωτογραφίες από το κινητό τους, στέλνουν μηνύματα κειμένου σε φίλους, ανανεώνουν την κατάστασή τους στο Facebook τους, γράφουν σχόλια στο διαδίκτυο, κάνουν κλικ σε διαφημίσεις, κ.ο.κ. Οι υπολογιστές επίσης, δημιουργούν και διαχειρίζονται όλο και περισσότερα δεδομένα. Η αγορά ενός προϊόντος από το διαδίκτυο καταγράφεται ως δεδομένο αγοράς με κάποιον πωλητή.

Η εκθετική αύξηση των δεδομένων παρουσιάζεται ως πρόκληση στον τομέα των επιχειρήσεων, όπως είναι οι Google, Yahoo, Amazon και Microsoft. Χρειάζονται να διασχίσουν terabytes και petabytes από δεδομένα, ώστε να διακρίνουν ποιοι ιστότοποι είναι δημοφιλείς, ποία βιβλία έχουν ζήτηση και τι είδους διαφημίσεις προσελκύουν τους χρήστες του διαδικτύου. Τα υπάρχοντα εργαλεία έχουν αρχίσει να γίνονται ανεπαρκή για την επεξεργασία πολύ μεγάλων συνόλων δεδομένων. Η Google ήταν πρώτη που δημοσιοποίησε το MapReduce – ένα σύστημα που χρησιμοποίησαν για να κλιμακώσουν τις ανάγκες επεξεργασίας δεδομένων. Αυτό το σύστημα προκάλεσε μεγάλο ενδιαφέρον στις επιχειρήσεις που αντιμετώπιζαν το ίδιο πρόβλημα και δεν ήταν ανέφικτο για την καθεμία να εφεύρει το δικό της εργαλείο. Ο Doug Cutting είδε την ευκαιρία και έδωσε προσοχή στην ανάπτυξη μιας ανοιχτού κώδικα έκδοσης του MapReduce συστήματος, γνωστή ως Hadoop. Σήμερα, το Hadoop είναι ένα κομμάτι του πυρήνα της υπολογιστικής υποδομής για πολλές εταιρείες ιστού, όπως οι Yahoo, Facebook, LinkedIn και Twitter. Όλο και περισσότερες παραδοσιακές εταιρείες, όπως των μέσων ενημέρωσης και τηλεπικοινωνιών, αρχίζουν να υιοθετούν αυτό το σύστημα αυτό.

2.1 Τι είναι το Hadoop

Το Hadoop αναπτύχθηκε από την Apache Software Foundation (ASF) και είναι ένα ανοιχτού κώδικα επαναχρησιμοποιήσιμη πλατφόρμα λογισμικού (framework) γραμμένη σε JAVA για τη συγγραφή και την εκτέλεση κατανεμημένων εφαρμογών, οι οποίες επεξεργάζονται μεγάλου όγκου δεδομένα. Το Hadoop προήλθε από το MapReduce της Google και της Google το File System (GFS) [2][7][8][9]. Ο κατανεμημένος προγραμματισμός είναι ένα μεγάλο και ποικίλο πεδίο, αλλά οι βασικές διακρίσεις του Hadoop σχετικά με αυτό είναι οι παρακάτω:

- **Προσβάσιμο** (Accessible) – Το Hadoop τρέχει σε συστοιχίες (clusters) κόμβων (commodity machines) συνδεδεμένων στο δίκτυο ή σε υπηρεσίες διαδικτύου (cloud) , όπως το Amazon’s Elastic Compute Cloud (EC2).

- **Εύρωστο** (Robust) – Του Hadoop η αρχιτεκτονική σχεδιάστηκε με την υπόθεση ύπαρξης συχνής δυσλειτουργίας υλικού. Έτσι λοιπόν, μπορεί ομαλά να διαχειριστεί περισσότερες από αυτές τις αποτυχίες.

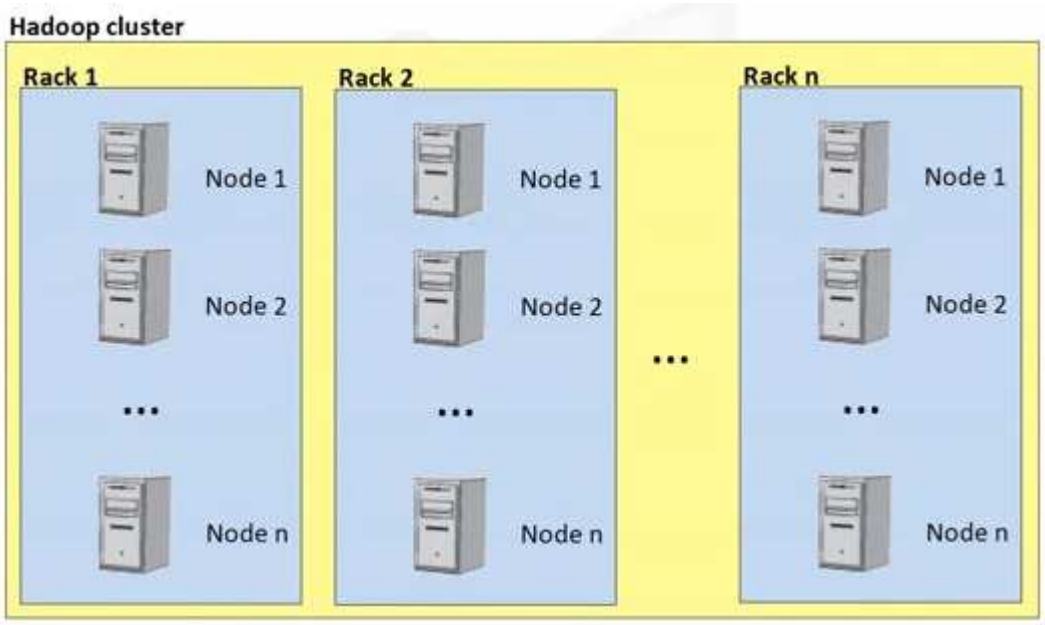
- **Επεκτάσιμο** (Scalable) – Το Hadoop κλιμακώνεται γραμμικά για την διαχείριση μεγαλύτερων όγκων δεδομένων προσθέτοντας περισσότερους κόμβους (nodes) στη συστοιχία.

- **Απλό** (Simple) – Το Hadoop επιτρέπει στους χρήστες να γράφουν γρήγορα και αποδοτικά παράλληλο κώδικα.

Η πλατφόρμα Hadoop αποτελείται από πολλά συστατικά όπως:

- **MapReduce** – Το οποίο είναι ένα περιβάλλον συγγραφής προγραμμάτων για την επεξεργασία μεγάλου όγκου δομημένων και αδόμητων δεδομένων γρήγορα και αξιόπιστα, μέσα από μεγάλες συστοιχίες από εικονικές ή φυσικές μηχανές.
- **Hadoop Distributed File System (HDFS)** – Διαχειρίζεται την αποθήκευση και τη γρήγορη πρόσβαση μεγάλου όγκου δεδομένα σε εκατοντάδες κόμβους.
- **Hive** – Για ad-hoc ερωτήματα σε σύνολα δεδομένων που είναι αποθηκευμένα στο HDFS (Hadoop's Warehouse)
- **HBase** – Ένα NoSQL περιβάλλον που δίνει τη δυνατότητα άμεσης πρόσβασης ανάγνωσης και εγγραφής στα Big Data και την παρουσίαση τους στους τελικούς χρήστες.

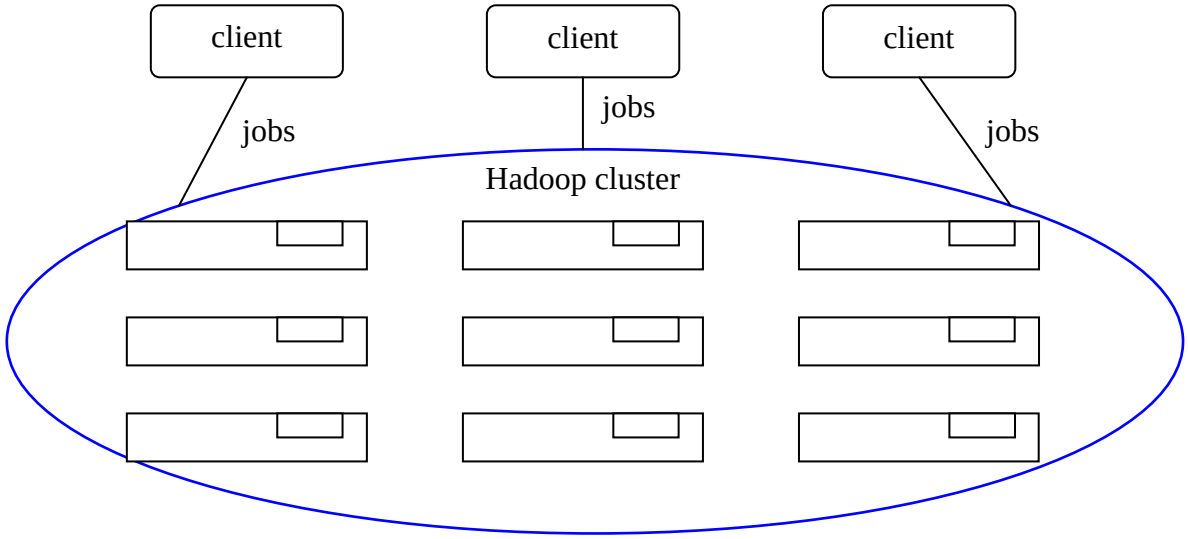
Πριν αρχίσουμε να εξηγούμε για τη φιλοσοφία των κατανεμημένων συστημάτων, θα πρέπει να διακρίνουμε τα μέρη που απαρτίζουν μία συστοιχία. Ο κόμβος είναι απλά ένας απλός υπολογιστής που περιέχει δεδομένα. Ένα πλήθος από κόμβους (περίπου 30 με 40) βρίσκονται στο ίδιο μέρος μαζί και είναι συνδεδεμένοι σε μία κοινή συσκευή δικτύωσης το "switch". Το εύρος ζώνης του δικτύου (network bandwidth) μεταξύ δύο κόμβων στο ίδιο rack είναι μεγαλύτερο από το εύρος ζώνης μεταξύ δύο κόμβων σε διαφορετικά racks. Μία συστοιχία Hadoop ή απλά συστοιχία είναι μία συλλογή από racks Εικόνα 2.1.



Εικόνα 2.1 Μία συστοιχία Hadoop.

Ένα παράδειγμα αλληλεπίδρασης κάποιου χρήστη σε μία συστοιχία Hadoop φαίνεται στην Εικόνα 2.2 όπου μία συστοιχία Hadoop απαρτίζεται από κόμβους δικτυωμένους μεταξύ τους σε μία τοποθεσία (π.χ. σε ένα data center, συχνά στο ίδιο σύνολο από racks). Η αποθήκευση και η επεξεργασία δεδομένων γίνεται στο "cloud" από υπολογιστές και διάφοροι χρήστες μπορούν να υποβάλουν "jobs" στο Hadoop από διαφορετικούς

υπολογιστές πελάτες (clients), οι οποίοι μπορούν να είναι απομακρυσμένοι από τη συστοιχία Hadoop.



Εικόνα 2.2 Μία συστοιχία Hadoop έχει πολλές παράλληλες μηχανές για αποθήκευση και επεξεργασία μεγάλων συνόλων δεδομένων. Υπολογιστές πελάτες στέλνουν εργασίες στο "cloud" υπολογιστών και παίρνουν αποτελέσματα.

2.2 Η ανάγκη που έκανε δημοφιλή την πλατφόρμα Hadoop

Η ανάγκη που οδήγησε στη χρήση κατανεμημένων συστημάτων για την επεξεργασία των μεγάλων δεδομένων οφείλεται στο ότι τα παράλληλα συστήματα είναι οικονομικά ασύμφορα. Έτσι λοιπόν, έγινε δημοφιλής η λύση της σύνδεσης πολλών υπολογιστών χαμηλού κόστους μαζί σε ένα ενιαίο λειτουργικό κατανεμημένο σύστημα.

Για να κατανοήσουμε γιατί είναι δημοφιλή τα κατανεμημένα συστήματα (scale-out) έναντι των μεγάλων μονολιθικών εξυπηρετητών (scale-up), ας συλλογιστούμε την τιμή απόδοσης της τρέχουσας τεχνολογίας I/O. Μία ισχυρή μηχανή με τέσσερα κανάλια I/O, όπου το καθένα έχει ισχύς διεκπεραίωσης (throughput) 100MB/sec θα χρειαστούν τρεις ώρες να διαβαστούν 4 TB σύνολο δεδομένων. Με το Hadoop, αυτό το ίδιο σύνολο δεδομένων διαιρείται σε μικρότερα μπλοκ (τυπικά 64 MB) τα οποία κατανέμονται σε πολλές μηχανές στη συστοιχία μέσω του HDFS (Hadoop Distributed File System). Με μέτριο λόγο αντιγραφής (replication), η συστοιχία υπολογιστών μπορεί να διαβάσει το σύνολο δεδομένων παράλληλα και έτσι να παρέχουν μία πολύ υψηλή ισχύς διεκπεραίωσης. Και με αυτή τη λογική η συστοιχία υπολογιστών, να αποτελεί την φθηνότερη λύση από τη δημιουργία ισχυρών συστημάτων, αναβαθμίζοντας τους ίδιους τους υπολογιστές.

Το Hadoop εστιάζει στη μεταφορά κώδικα-σε-δεδομένα ("code-to-data") αντί για το αντίστροφο. Αναφερόμενοι στην Εικόνα 2.2 πάλι, βλέπουμε ότι και τα δύο, τα δεδομένα και η επεξεργασία τους γίνονται μαζί στη Hadoop συστοιχία. Οι πελάτες στέλνουν μόνο το MapReduce πρόγραμμα για να εκτελεστεί, όπου αυτά τα προγράμματα είναι συνήθως μικρά (kilobytes). Και πιο σημαντικό είναι ότι η φιλοσοφία κώδικα-σε-δεδομένα εφαρμόζεται στη συστοιχία Hadoop από μόνη της. Τα δεδομένα διαιρούνται και

κατανέμονται στη συστοιχία, και όσο είναι δυνατόν, ο υπολογισμός για κάθε ένα κομμάτι δεδομένων γίνεται στο ίδιο μηχάνημα όπου βρίσκεται αυτό το κομμάτι.

Αυτή η φιλοσοφία κώδικα-σε-δεδομένα δίνει νόημα για αυτό το είδος της επεξεργασίας μεγάλου όγκου δεδομένων, για το οποίο έχει σχεδιαστεί το Hadoop. Τα προγράμματα που εκτελούνται ("code") είναι πολύ μικρότερα από τα δεδομένα και είναι ευκολότερο να διεκπεραιωθούν. Επίσης, παίρνει περισσότερο χρόνο να κινηθούν δεδομένα μέσω δικτύου, από το να εφαρμόζονται οι υπολογισμοί στο ίδιο κατανεμημένο σύστημα. Έτσι λοιπόν, αφήνουμε τα δεδομένα να παραμείνουν εκεί που είναι και μεταφέροντας τον εκτελέσιμο κώδικα στο μηχάνημα που τα φιλοξενεί.

2.3 Hadoop Distributed File System (HDFS)

Το κατανεμημένο σύστημα αρχείων της πλατφόρμας Hadoop τρέχει πάνω από το υπάρχον σύστημα αρχείων σε κάθε κόμβο της συστοιχίας. Έχει σχεδιαστεί για πολύ ειδικά πρότυπα πρόσβασης δεδομένων (sequential & streaming data access patterns) [2].

Το Hadoop λειτουργεί καλύτερα με μεγάλα αρχεία. Όσο πιο μεγάλο είναι ένα αρχείο, τόσο λιγότερο χρόνο ξοδεύει το Hadoop για την αναζήτηση (seeking) της θέσης των επόμενων δεδομένων στο δίσκο και τον περισσότερο χρόνο το Hadoop τρέχει στα όρια του εύρους ζώνης των δίσκων. Οι αναζητήσεις είναι χρονοβόρες διεργασίες και είναι χρήσιμες μόνο όταν πρόκειται για ανάλυση ενός μικρού υποσυνόλου ενός συνόλου δεδομένων. Το Hadoop σχεδιάστηκε να τρέχει σε ολόκληρο το σύνολο δεδομένων και γι'αυτό είναι καλύτερα η ελαχιστοποίηση των αναζητήσεων χρησιμοποιώντας μεγάλα αρχεία. Η πλατφόρμα αυτή σχεδιάστηκε για πρόσβαση σε δεδομένα συνεχούς ροής (streaming) ή διαδοχικά (sequential) δεδομένα, αντί για τυχαία πρόσβαση. Διαδοχικά δεδομένα σημαίνει λιγότερες διεργασίες αναζήτησης, όπου το Hadoop κάνει αναζήτηση μόνο στην αρχή του κάθε block και ξεκινά να διαβάζει διαδοχικά από εκεί. Χρησιμοποιεί τμήματα (blocks) (64MB (προεπιλογή), 128MB (συνιστάται) – σε σύγκριση με το 4KB στο UNIX) για την αποθήκευση ενός αρχείου ή μέρη του αρχείου. Ένα Hadoop block είναι ένα αρχείο πάνω από το υπάρχων σύστημα αρχείων. Όπως το υποκείμενο σύστημα αρχείων αποθηκεύει τα αρχεία σε blocks, ένα Hadoop block μπορεί να αποτελείται από πολλά blocks στο υποκείμενο σύστημα αρχείων.

Τα block έχουν πολλά πλεονεκτήματα. Πρώτων, είναι καθορισμένου μεγέθους και εύκολο να υπολογίσουμε πόσα χωράνε στο δίσκο. Δεύτερον, ένα αρχείο μπορεί να είναι μεγαλύτερο από κάθε δίσκο στο δίκτυο και τα block του να είναι σε πολλούς κόμβους. Εάν ένα αρχείο ή ένα κομμάτι του αρχείου είναι μικρότερο από το μέγεθος του block, τότε θα χρησιμοποιήσει το block που έχει το υπόλοιπο μέγεθος για την αποφυγή σπατάλης χώρου. Και τελευταίο, το HDFS είναι ανεκτικό σε σφάλματα αντιγράφοντας τα block στους κόμβους για την αποφυγή απώλειας δεδομένων.

2.4 Οι κόμβοι του Hadoop

Μέχρι εδώ, είδαμε την έννοια της κατανεμημένης αποθήκευσης και λειτουργίας. Ας δούμε πως το Hadoop την υλοποιεί σε μία πλήρως διαμορφωμένη συστοιχία, όπου "running Hadoop" σημαίνει ότι τρέχει ένα σύνολο από διεργασίες παρασκηνίου τους δαίμονες

(daemons), σε διαφορετικούς κόμβους στο δίκτυο. Αυτοί οι δαίμονες έχουν συγκεκριμένους ρόλους, μερικοί υπάρχουν μόνο σε ένα κόμβο, μερικοί υπάρχουν σε πολλούς κόμβους [1][2].

Οι βασικοί δαίμονες ανάλογα με τη λειτουργία τους στους κόμβους διακρίνονται σε:

HDFS – storage deamons
- NameNode
- DataNode
- Secondary NameNode

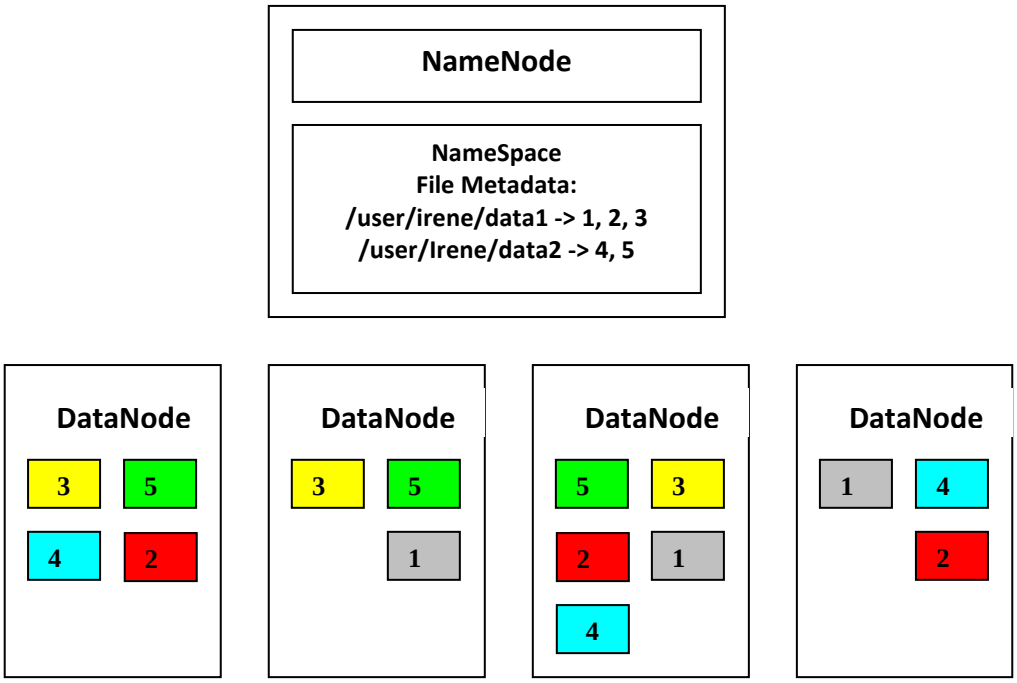
MapReduce – computing deamons
- Job Tracker
- Task Tracker

2.4.1 NameNode

Το Hadoop εφαρμόζει μία “master/slave” αρχιτεκτονική για κατανεμημένη αποθήκευση και επεξεργασία. Ο NameNode είναι ο master του HDFS, ο οποίος ορίζει τους slave DataNode δαίμονες να εκτελέσουν μία χαμηλού-επιπέδου εργασία I/O. Ο NameNode καταγράφει το πώς τα αρχεία διαιρούνται σε μπλοκ αρχεία, ποιοι κόμβοι έχουν αποθηκευμένα αυτά τα μπλοκ, και έχει τη γνώση όλου του κατανεμημένου συστήματος αρχείων. Ο NameNode λειτουργεί με χρήση της μνήμης RAM και εντατικών εργασιών I/O. Όπως γίνεται με το “server-hosting”, ο NameNode δεν αποθηκεύει τα δεδομένα του χρήστη ή εκτελεί κάποιους υπολογισμούς για ένα MapReduce πρόγραμμα για να μειώσει το φόρτο εργασίας στο υπολογιστικό μηχάνημα. Αυτό σημαίνει ότι ο NameNode κόμβος δεν λειτουργεί ως DataNode ή TaskTracker. Υπάρχει όμως ένα αρνητικό σημείο για τον NameNode – είναι ένα “μοναδικό σημείο αποτυχίας” της Hadoop συστοιχίας και για αυτό κρατά αντίγραφα της τρέχουσας κατάστασης του συστήματος σε ένα τοπικό δίσκο και στο NFS (Network File System – εξυπηρετητής δικτύου). Για να παρέχει αξιοπιστία θα πρέπει σε αυτό το κόμβο να επενδύσουμε βάζοντάς του αρκετή μνήμη RAM ώστε να κρατά εκεί τα μεταδεδομένα του συστήματος αρχείων. Για μερικούς από τους άλλους δαίμονες, αν οι υπολογιστές στους οποίους τρέχουν αποτυγχάνουν για λόγους λογισμικού ή υλικού, η Hadoop συστοιχία πιθανόν θα συνεχίσει τη λειτουργία της ομαλά ή θα μπορούσαμε να κάνουμε επανεκκίνηση του υπολογιστή.

2.4.2 DataNode

Κάθε slave υπολογιστής μέσα στη συστοιχία θα φιλοξενεί έναν DataNode δαίμονα για να εκτελεί εργασίες του κατανεμημένου συστήματος αρχείων – διάβασμα και εγγραφή των HDFS μπλοκ σε αρχεία στο τοπικό σύστημα αρχείων. Όταν θέλουμε να διαβάσουμε ή να γράψουμε σε ένα HDFS αρχείο, το αρχείο διαιρείται σε μπλοκ και ο NameNode θα ενημερώσει τον πελάτη ποιο DataNode έχει το κάθε μπλοκ. Ο πελάτης επικοινωνεί άμεσα με τους DataNode δαίμονες οι οποίοι επεξεργάζονται τα τοπικά αρχεία που τους αντιστοιχούν. Επιπλέον, ο DataNode μπορεί να επικοινωνήσει με άλλους DataNodes για να αντιγράψει τα δεδομένα του για πλεονασμό (replication). Η Εικόνα 2.3 μας δείχνει τους ρόλους του NameNode και των DataNode.



Εικόνα 2.3 Η αλληλεπίδραση του NameNode και των DataNodes στο HDFS. Ο NameNode ενημερώνεται για το πώς ένα αρχείο διαιρείται σε μπλοκ από το αρχείο μεταδεδομένων, καθώς και ποιοι DataNode κρατούν τα τρέχοντα αντίγραφα ασφαλείας των μπλοκ αυτών.

Στην εικόνα αυτή, τα αρχεία δεδομένων έχουν διαιρεθεί ως εξής: data1 σε τρία μπλοκ τα 1, 2, 3 και το data2 σε 2 μπλοκ τα 4, 5 και για κάθε μπλοκ έχουν δημιουργηθεί τρία αντίγραφα. Αυτό μας διαβεβαιώνει ότι όταν ένας DataNode κολλήσει ή είναι μη προσβάσιμος μέσω του δικτύου, θα μπορούμε ακόμη να διαβάσουμε τα αρχεία. Οι DataNode συνέχεια ενημερώνουν τον NameNode για τα τρέχων μπλοκ που είναι αποθηκευμένα, καθώς και λαμβάνουν εντολές για δημιουργία, μετακίνηση ή διαγραφή των μπλοκ στο τοπικό δίσκο.

2.4.3 Secondary NameNode

Ο Secondary NameNode (SNN) είναι ένας δαίμονας βοηθός για την παρακολούθηση της κατάστασης του HDFS στη συστοιχία. Όπως με τον NameNode, κάθε συστοιχία έχει έναν SNN, και βρίσκεται στο δικό του υπολογιστή. Ούτε ο DataNode και ούτε ο TaskTracker δαίμονας τρέχουν στον ίδιο εξυπηρετητή. Ο SNN διαφέρει από τον NameNode στο ότι δεν λαμβάνει ή καταγράφει σε πραγματικό χρόνο αλλαγές στο HDFS. Αντί για αυτό, επικοινωνεί με τον NameNode για να στιγμιότυπα (snapshots) των metadata του HDFS σε τακτά χρονικά διαστήματα, το οποίο έχει οριστεί στη συστοιχία (cluster configuration).

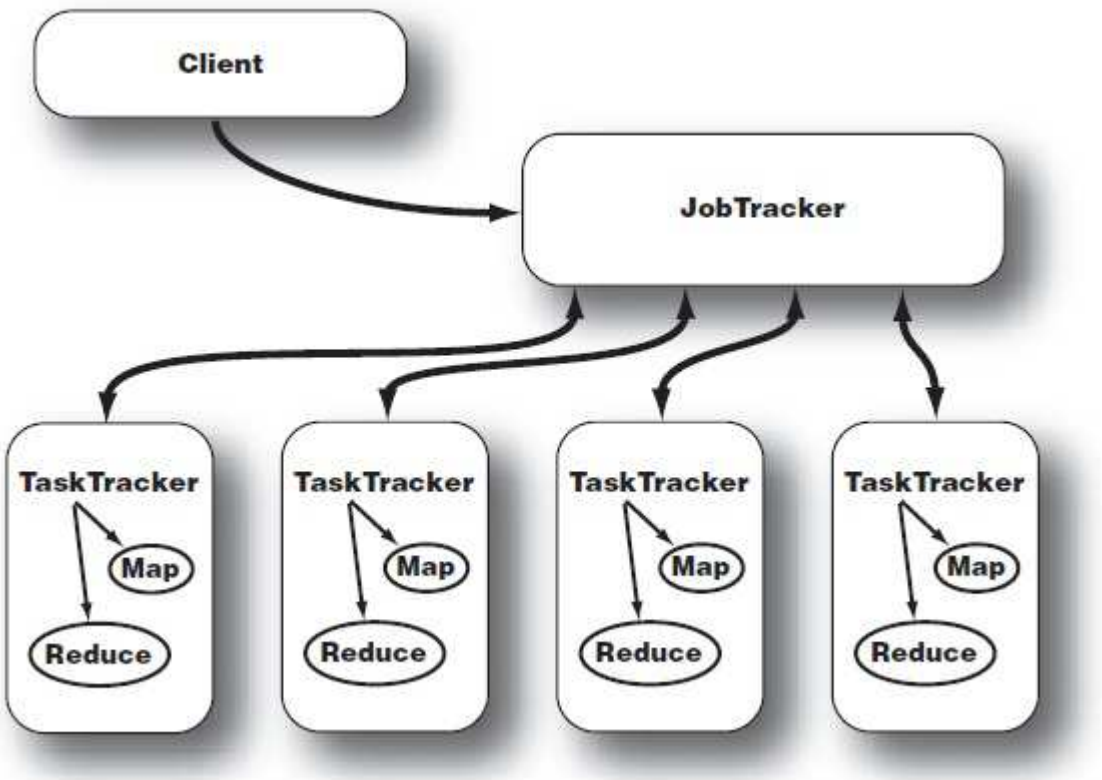
Όπως έχει αναφερθεί προηγουμένως, ο NameNode είναι ένα μοναδικό σημείο αποτυχίας της Hadoop συστοιχίας, και τα στιγμιότυπα του SNN βοηθούν στην ελαχιστοποίηση των διαστημάτων διακοπής και στην απώλεια των δεδομένων. Εντούτοις, μία αποτυχία του NameNode απαιτεί την ανθρώπινη παρέμβαση για επαναρίθμηση της συστοιχίας ώστε να χρησιμοποιεί τον SNN ως τον κύριο NameNode.

2.4.4 JobTracker

Ο JobTracker δαίμονας είναι ένα επικοινωνιακό κανάλι μεταξύ της εφαρμογής μας και του Hadoop. Όταν υποβάλουμε κώδικα στη συστοιχία, ο JobTracker αποφασίζει για το σχέδιο εκτέλεσης επιλέγοντας ποια αρχεία θα επεξεργαστεί, αναθέτει στους κόμβους διαφορετικές εργασίες και παρακολουθεί όλες τις εργασίες που εκτελούνται. Όταν μία εργασία αποτυγχάνει, ο JobTracker αυτόματα επανεκκινεί την εργασία, πιθανόν σε διαφορετικό κόμβο μέχρι ένα ανώτερο όριο προσπαθειών. Υπάρχει ένας μόνο JobTracker δαίμονας για κάθε Hadoop συστοιχία και τρέχει σε ένα εξυπηρετητή ως ένας master κόμβος της συστοιχίας.

2.4.5 TaskTracker

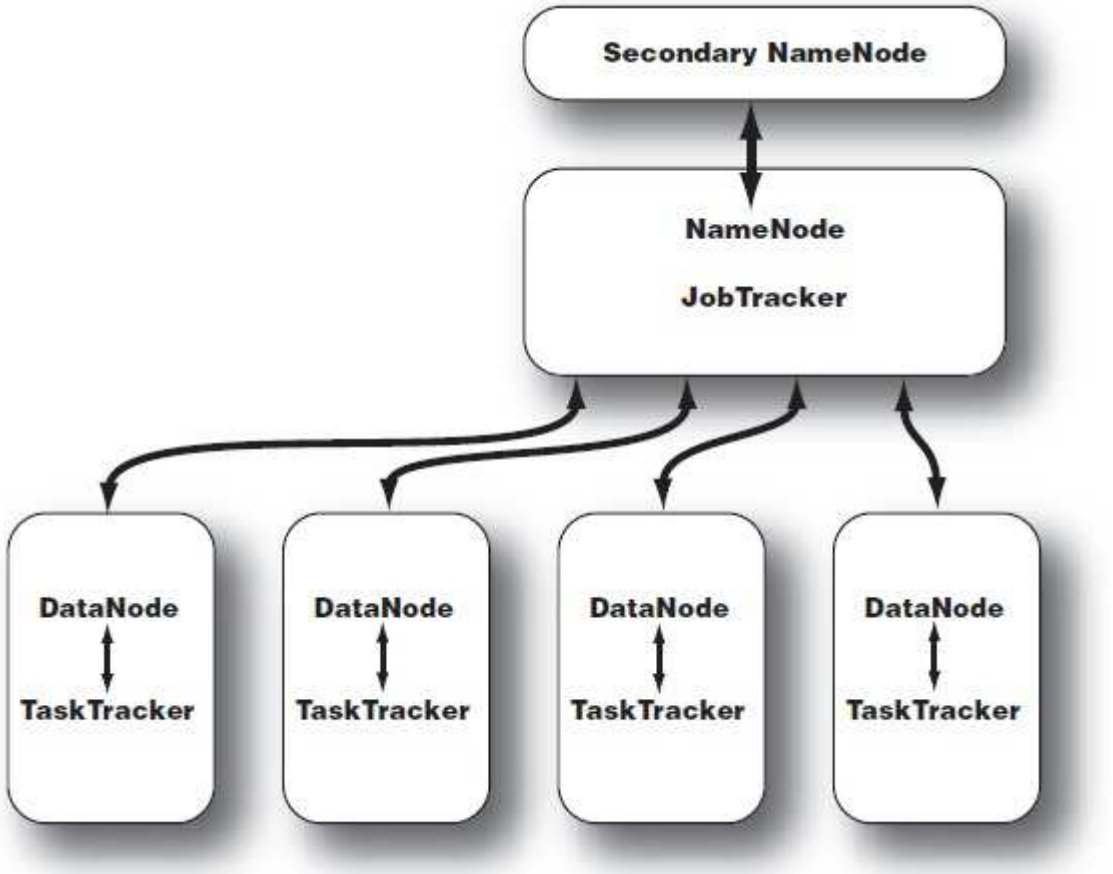
Όπως με τους δαίμονες αποθήκευσης, οι υπολογιστικοί (computing) δαίμονες επίσης ακολουθούν μία master/slave αρχιτεκτονική: ο JobTracker είναι ο master που επιβλέπει την εκτέλεση μίας MapReduce εργασίας και ο TaskTracker διαχειρίζεται την εκτέλεση αυτή σε ξεχωριστές παράλληλες εργασίες σε κάθε slave κόμβο. Η Εικόνα 2.4 δείχνει αυτή την αλληλεπίδραση.



Εικόνα 2.4 Όταν ο client καλέσει τον JobTracker να αρχίσει την εργασία επεξεργασίας των δεδομένων, ο JobTracker διαμοιράζει την εργασία και αναθέτει διαφορετικές map και reduce εργασίες σε κάθε TaskTracker στη συστοιχία.

Η Εικόνα 2.5 δείχνει μία χαρακτηριστική συστοιχία Hadoop. Αυτή η τοπολογία μας δείχνει ότι σε ένα master κόμβο τρέχουν οι NameNode και JobTracker δαίμονες και ένα μόνο κόμβο που τρέχει τον SNN δαίμονα στην περίπτωση που ο master αποτύχει. Στη

περίπτωση μεγάλων συστοιχιών, οι NameNode και JobTracker είναι σε διαφορετικούς υπολογιστές. Οι slave υπολογιστές έχουν ένα DataNode και ένα TaskTracker για να εκτελούν εργασίες στον ίδιο κόμβο όπου τα δεδομένα έχουν αποθηκευθεί.



Εικόνα 2.5 Η τοπολογία μίας συστοιχίας Hadoop είναι μίας master/slave αρχιτεκτονικής στην οποία ο NameNode και ο JobTracker είναι οι master και οι DataNodes και οι TaskTrackers είναι οι slave.

2.5 Παράδειγμα κατανόησης της κατανεμημένης επεξεργασίας

Έχοντας γνώση των μοντέλων επεξεργασίας δεδομένων όπως pipelines και message queues, αυτά τα μοντέλα παρέχουν ειδικές δυνατότητες στην ανάπτυξη εφαρμογών επεξεργασίας δεδομένων. Οι πιο δημοφιλείς pipelines (διασωληνώσεις) είναι του UNIX τα pipes. Τα pipelines βοηθούν στην επαναχρησιμοποίηση προτύπων επεξεργασίας: συνδυάζοντας αλυσοδένοντας τα υπάρχοντα πρότυπα για τη δημιουργία νέων. Οι Message Queues (ουρές μηνυμάτων) μπορούν να βοηθήσουν στο συγχρονισμό των προτύπων επεξεργασίας [1][2][3].

Παρόμοια το MapReduce είναι ένα μοντέλο επεξεργασίας δεδομένων. Το μεγαλύτερό του πλεονέκτημα είναι η ευκολία στην κλιμάκωση των δεδομένων που επεξεργάζεται όταν λειτουργεί πάνω από πολλαπλούς υπολογιστικούς κόμβους. Στο μοντέλο MapReduce, τα θεμελιώδη συστατικά επεξεργασίας ονομάζονται mappers και reducers. Αναλύοντας μια εφαρμογή επεξεργασίας δεδομένων σε mappers και reducers συχνά δεν είναι τετριμμένο. Όμως από τη στιγμή που θα γράψουμε ένα πρόγραμμα στο MapReduce, το να

κλιμακώσουμε την εφαρμογή για να τρέχει πάνω από εκατοντάδες, χιλιάδες ή δεκάδες μηχανές στη συστοιχία είναι απλά μια αλλαγή στη διαμόρφωσή της. Αυτή η απλή κλιμάκωση είναι ο λόγος που το μοντέλο MapReduce προσέλκυσε πολλούς προγραμματιστές.

Στο παράδειγμα που θα χρησιμοποιήσουμε, θα μετρήσουμε τον αριθμό των εμφανίσεων κάθε λέξης σε ένα σύνολο από έγγραφα. Στο συγκεκριμένο παράδειγμα έχουμε ένα σύνολο που αποτελείται από ένα μόνο έγγραφο το οποίο περιέχει μόνο μία πρόταση:

do as I say, not as I do.

Το αποτέλεσμα θα είναι ένας πίνακας με δύο στήλες, μια για τη λέξη και μια για τις εμφανίσεις της κάθε λέξης:

Word	Count
as	2
do	2
I	2
not	1
say	1

Το παράδειγμα το ονομάζουμε word counting και παρακάτω περιγράφεται ο ψευδοκώδικάς του:

```
define wordCount as Multiset;
for each document in documentSet {
      T= tokenize (document);
      for each token in T{
            wordCount[token]++;
      }
}
display(wordCount);
```

Αυτό το πρόγραμμα εκτελεί ένα βρόχο περνώντας από όλα τα έγγραφα. Για κάθε έγγραφο, οι λέξεις υπολογίζονται μία προς μία κάνοντας χρήση μιας συμβολικής διεργασίας (tokenization). Κάθε λέξη μπαίνει σε ένα hash table ονόματι wordCount στον οποίο σε κάθε συγκεκριμένη λέξη (key) έχουμε και το σύνολο των εμφανίσεων της (value) π.χ. `wordCount[as] = 2`.Τελικά μια συνάρτηση display() τυπώνει όλες τις καταχωρήσεις του πίνακα wordCount.

Αυτό το πρόγραμμα δουλεύει καλά όσο το σύνολο των εγγράφων που θέλουμε να επεξεργαστούμε δεν μεγαλώνει. Για παράδειγμα έστω πως θέλουμε να δημιουργήσουμε ένα φίλτρο spam μηνυμάτων έτσι ώστε να γνωρίζουμε τις πιο συχνά χρησιμοποιούμενες λέξεις στα εκατομμύρια των spam e-mails που λαμβάνονται. Με το να εκτελέσουμε την ίδια διαδικασία επαναληπτικά για όλα τα έγγραφα χρησιμοποιώντας έναν και μόνο υπολογιστή θα ήταν εξαιρετικά χρονοβόρο. Μπορούμε να επιταχύνουμε την διαδικασία αυτή ξαναγράφοντας το πρόγραμμα έτσι ώστε να κατανέμει την εργασία σε διαφορετικούς

υπολογιστές. Κάθε υπολογιστής θα επεξεργάζεται ένα διαφορετικό κομμάτι των κειμένων. Όταν όλοι οι υπολογιστές ολοκληρώσουν αυτήν την επεξεργασία, μια δεύτερη φάση επεξεργασίας θα συνδυάζει τα αποτελέσματα όλων των υπολογιστών.

Ο ψευδοκώδικας για την πρώτη φάση που θα κατανεμηθεί σε πολλού υπολογιστές είναι:

```
define wordCount as Multiset;
for each document in documentSubset {
      T= tokenize (document);
      for each token in T{
            wordCount[token]++;
      }
}
sendToSecondPhase(wordCount);
```

Και ο ψευδοκώδικας για τη δεύτερη φάση είναι:

```
define totalWordCount as Multiset;
for each wordCount received from firstPhase{
      multisetAdd (totalWordCount, wordCount);
}
```

Παρόλο που ο παραπάνω κώδικας δεν είναι ιδιαίτερα δύσκολος, ορισμένες λεπτομέρειες μπορούν να παρεμποδίσουν την λειτουργία του. Πρώτα από όλα, αγνοούμε τις απαιτήσεις εκτέλεσης ανάγνωσης των εγγράφων. Εάν όλα τα έγγραφα είναι αποθηκευμένα σε έναν κεντρικό server αποθήκευσης, τότε η συμφόρηση θα δημιουργηθεί στο εύρος ζώνης του server. Θα πρέπει να διαχωριστούν τα έγγραφα ανάμεσα στο σύνολο των μηχανών επεξεργασίας έτσι ώστε κάθε μηχάνημα να επεξεργάζεται μόνο τόσα έγγραφα όσα είναι αποθηκευμένα σε αυτό. Αυτό θα μειώσει τη συμφόρηση που προκαλείται στο κεντρικό δίσκο αποθήκευσης. Αυτό μας υπενθυμίζει ότι τόσο η επεξεργασία όσο και η αποθήκευση πρέπει να είναι στενά συνδεδεμένες στις εφαρμογές για κατανεμημένα δεδομένα.

Ένα άλλο ελάττωμα είναι ότι το wordCount (και το totalWordCount) είναι αποθηκευμένα στη μνήμη. Όταν επεξεργαζόμαστε μεγάλα σύνολα εγγράφων, το πλήθος των λέξεων που δεν επαναλαμβάνονται μπορεί να υπερβαίνουν το μέγεθος της RAM. Για παράδειγμα η Αγγλική γλώσσα έχει γύρω στο ένα εκατομμύριο λέξεις, μέγεθος που μπορεί άνετα να αποθηκευτεί σε ένα iPod αλλά το πρόγραμμά μας θα έχει να αντιμετωπίσει μοναδικά ονόματα που δεν υπάρχουν σε κανένα αγγλικό λεξικό. Για παράδειγμα, θα πρέπει να αντιμετωπίσουμε μοναδικά ονόματα όπως το Hadoop. Επίσης θα κληθούμε να αντιμετωπίσουμε ορθογραφικά λάθη όπως η λέξη exampel αντί της λέξης example και μετράμε όλους τους διαφορετικούς τύπους εμφάνισης μιας λέξης ξεχωριστά (για παράδειγμα eat, ate, eaten και eating). Αλλά ακόμη και εάν ο αριθμός τον μοναδικών λέξεων στο σύνολο των εγγράφων είναι διαχειρίσιμος από τη μνήμη, μια μικρή αλλαγή στον ορισμό του προβλήματος μπορεί να εκτινάξει την πολυπλοκότητα του προγράμματος. Για παράδειγμα, αν αντί για λέξεις σε έγγραφα μπορεί να θέλουμε να μετράμε τις IP διευθύνσεις σε ένα log file ή τη συχνότητα των bigrams (είναι ζευγάρι από συνεχόμενες λέξεις). Για παράδειγμα η πρόταση “Do as I say, not as I do” μπορει να χωριστει στα ακόλουθα bigrams: Do as, as I, I say, say not, not as, as I, not as I do. Στη τελευταία περίπτωση, θα πρέπει να δουλέψουμε με ένα πολυσύνολο αποτελούμενο από εκατομμύρια εισόδους που υπερβαίνει το μέγεθος της RAM ενός εμπορικού υπολογιστή.

Το wordCount πρόγραμμα μπορεί να μην ταιριάζει στη μνήμη οπότε πρέπει να ξαναγράψουμε το πρόγραμμα έτσι ώστε να αποθηκεύει αυτόν το hash table στο δίσκο.

Επιπλέον, στη φάση 2 έχουμε ένα μόνο μηχάνημα, το οποίο θα επεξεργάζεται το wordCount που στέλνεται από όλα τα μηχανήματα της φάσης 1. Το να επεξεργαζόμαστε ένα wordCount είναι από μόνο του αρκετά δύσκολο. Αφότου έχουμε προσθέσει αρκετά μηχανήματα για την επεξεργασία της φάσης 1, το μοναδικό μηχάνημα της φάσης 2 θα δημιουργήσει τη συμφόρηση. Η προφανής ερώτηση είναι: θα μπορούσαμε να ξαναγράψουμε τη φάση 2 με ένα κατανεμημένο τρόπο έτσι ώστε να κλιμακώνεται με την προσθήκη επιπλέον μηχανημάτων;

Η απάντηση είναι ναι! Για να κάνουμε τη φάση 2 να λειτουργεί με κατανεμημένο τρόπο θα πρέπει κάπως να διαιρέσουμε το έργο της ανάμεσα σε πολλά μηχανήματα έτσι ώστε να μπορούν να τρέξουν ανεξάρτητα το ένα από το άλλο. Θέλουμε δηλαδή να διαιρέσουμε το wordCount μετά τη φάση 1 έτσι ώστε κάθε μηχάνημα στη φάση 2 να έχει να χειρίζεται μόνο ένα κομμάτι. Ας πούμε για παράδειγμα ότι διαθέτουμε 26 μηχανήματα για τη φάση 2. Αναθέτουμε σε κάθε μηχάνημα να χειρίζεται το wordCount για λέξεις που ξεκινούν με ένα συγκεκριμένο γράμμα της αλφαβήτου. Για παράδειγμα το μηχάνημα Α στη φάση 2 θα χειρίζεται λέξεις που ξεκινούν από α. Για να κάνουμε δυνατό αυτό το διαχωρισμό στη φάση 2 χρειάζεται να προχωρήσουμε σε μια μικρή αλλαγή στη φάση 1. Αντί για ένα και μοναδικό hash table βασισμένο στο δίσκο για το wordCount θα χρειαστούμε 26 hash tables: wordCount-a, wordCount-b και ούτω καθεξής. Κάθε ένα μετράει λέξεις που ξεκινάνε με ένα συγκεκριμένο γράμμα. Μετά το πέρας της φάσης 1 το wordCount-a καθενός από τα μηχανήματα της φάσης 1 θα στείλει το αποτέλεσμά του στο μηχάνημα Α της φάσης 2, όλα τα wordCount-b θα σταλούν στο μηχάνημα Β και ούτω καθεξής. Κάθε μηχάνημα της φάσης 1 στέλνει τα αποτελέσματά του στα μηχανήματα της φάσης 2.

Γυρνώντας πίσω, αυτό το πρόγραμμα καταμέτρησης λέξεων γίνεται αρκετά πολύπλοκο. Για να το κάνουμε να δουλέψει σε μία συστοιχία από κατανεμημένα μηχανήματα βλέπουμε ότι πρέπει να προσθέσουμε τις ακόλουθες λειτουργικότητες:

- Αποθήκευση αρχείων σε πολλές μηχανές επεξεργασίας (φάση 1)
- Να γράψουμε σε ένα hash table στο δίσκο που θα μας επιτρέπει την επεξεργασία δεδομένων χωρίς να περιορίζεται από το μέγεθος της RAM.
- Διαχωρισμό (partition) των ενδιάμεσων δεδομένων (wordCount) της φάσης 1
- Πέρασμα (shuffle) των διαχωρισμένων της φάσης 1 στα κατάλληλα μηχανήματα της φάσης 2.

Όλα τα παραπάνω είναι αρκετή δουλειά για κάτι τόσο απλό όσο μια καταμέτρηση λέξεων. Επίσης, δεν έχουμε ακόμη ασχοληθεί με θέματα όπως η ανοχή σε σφάλματα. (Τι θα συμβεί αν μία μηχανή αποτύχει στο μέσον της εκτέλεσης;). Για όλα τα παραπάνω χρειαζόμαστε μία πλατφόρμα σαν το Hadoop. Γιατί όταν γράφουμε την εφαρμογή μας στο μοντέλο MapReduce το Hadoop φροντίζει για όλη αυτή τη κλιμάκωση.

Τα προγράμματα που είναι γραμμένα σε MapReduce εκτελούνται σε δύο φάσεις, τη φάση του mapping και τη φάση του reducing. Κάθε φάση ορίζεται από μια συνάρτηση επεξεργασίας, mapper και reducer αντίστοιχα. Στη φάση του mapping το MapReduce λαμβάνει τα δεδομένα εισόδου και τροφοδοτεί κάθε ξεχωριστό στοιχείο τους στη συνάρτηση mapper. Στη φάση του reducing, ο reducer επεξεργάζεται όλες τις εξόδους του mapper και μας δίνει το τελικό αποτέλεσμα.

Με απλά λόγια, ο mapper φιλτράρει και μετασχηματίζει τα δεδομένα εισόδου σε κάτι που ο reducer μπορεί να συναθροίσει. Όπως φαίνεται υπάρχει μια αξιοσημείωτη ομοιότητα των δύο φάσεων του MapReduce σε σχέση με την ανάπτυξη του μετρητή λέξεων στην κλιμάκωση. Η ομοιότητα δεν είναι τυχαία. Το πλαίσιο του MapReduce σχεδιάστηκε έπειτα από χρόνια εμπειρίας σε εφαρμογές κλιμακούμενες και κατανεμημένες. Αυτό το πρότυπο των δύο φάσεων εμφανίζεται σε πολλά κλιμακωτά προγράμματα και αποτελεί βάση του πλαισίου του MapReduce.

Για να κλιμακώσουμε την κατανεμημένη εφαρμογή του μετρητή λέξεων στην προηγούμενη παράγραφο έπρεπε επίσης να γράψουμε τις συναρτήσεις partitioning και shuffling. Αυτές οι δύο συναρτήσεις είναι κοινά σχεδιαστικά πρότυπα που συνεργάζονται με τις συναρτήσεις mapping και reducing. Σε αντίθεση με τις mapping και reducing, οι partitioning και shuffling είναι γενικές λειτουργικότητες που δεν εξαρτώνται και πολύ από την εφαρμογή επεξεργασίας δεδομένων. Το πλαίσιο του MapReduce περιέχει μια εξορισμού υλοποίηση που δουλεύει στις περισσότερες περιπτώσεις.

Για να λειτουργήσουν σωστά οι mapping, reducing, partitioning και shuffling, θα πρέπει να συμφωνήσουμε σε μια κοινή δομή για τα δεδομένα προς επεξεργασία. Πρέπει να είναι αρκετά ευέλικτο και ισχυρό για να χειριστεί τις περισσότερες εφαρμογές επεξεργασίας δεδομένων. Το MapReduce χρησιμοποιεί λίστες (lists) και ζευγάρια (key/value) σαν στοιχειώδη δεδομένα. Τα keys και values είναι συνήθως ακέραιοι ή αλφαριθμητικά καθώς επίσης μπορεί να είναι τεχνητές τιμές για να αγνοούνται ή πολύπλοκοι τύποι αντικειμένων. Οι συναρτήσεις map και reduce πρέπει να υπακούουν στους ακόλουθους περιορισμούς σχετικά με τους τύπους των keys και values.

	Input	Output
map	<k1, v1>	list<k2, v2>
reduce	<k2, list(v2)>	list<k3, v3>

Στο πλαίσιο MapReduce γράφουμε εφαρμογές ορίζοντας τους mapper και reducer. Ας δούμε ολοκληρωμένη τη ροή δεδομένων:

1. Η είσοδος της εφαρμογής μας πρέπει να δομείται σαν μια λίστα από ζευγάρια (key/value), list(<k1, v1>). Αυτή η μορφή εισόδου μπορεί να μοιάζει αόριστη αλλά στην πράξη είναι αρκετά απλή. Η μορφή της εισόδου για την επεξεργασία πολλαπλών αρχείων είναι συνήθως της μορφής list(<String filename, String file content>). Η μορφή της εισόδου για την επεξεργασία ενός μεγάλου αρχείου, όπως ένα log file είναι list(<Integer line_number, String log_event>).

2. Η λίστα των ζευγαριών (key/value) κατακερματίζεται και κάθε ξεχωριστό ζευγάρι (key/value), <k1, v1>, επεξεργάζεται καλώντας τη συνάρτηση map του mapper. Στην πράξη το key k1 συνήθως αγνοείται από τον mapper. Ο mapper μετατρέπει κάθε ζευγάρι <k1, v1> σε μια λίστα από ζευγάρια <k2, v2>. Οι λεπτομέρειες της μετατροπής σε μεγάλο βαθμό καθορίζουν το τι κάνει το πρόγραμμα MapReduce. Να σημειώσουμε ότι τα ζευγάρια (key/value) επεξεργάζονται με τυχαία σειρά. Η μετατροπή πρέπει να είναι αυτόνομη και λόγω αυτού η έξοδος εξαρτάται μόνο από το ζεύγος (key/value).

   Για το word counting, ο mapper παίρνει <String filename, String file_content> και γρήγορα αγνοεί το filename. Μπορεί να έχει σαν έξοδο μια λίστα από <String word, Integer count> αλλά μπορεί να είναι ακόμη πιο απλή. Όπως γνωρίζουμε το counts αθροιζεται σε ένα μεταγενέστερο στάδιο, οπότε μπορούμε να εξάγουμε μια λίστα με <String word, Integer 1> με επαναλαμβανόμενες εισόδους και να αφήσουμε το συνολικό άθροισμα για αργότερα. Έτσι στη λίστα εξόδου μπορούμε να έχουμε το ζεύγος (key/value) <”foo”, 3> μια φορά ή μπορούμε να έχουμε το ζεύγος <”foo”, 1> τρεις φορές. Όπως θα δούμε, η τελευταία προσέγγιση μπορεί να έχει ορισμένα οφέλη στην απόδοση, αλλά ας αφήσουμε αυτή τη βελτιστοποίηση προς το παρόν έως ότου να κατανοήσουμε πλήρως το πλαίσιο MapReduce.

3. Η έξοδος από όλους τους mapper είναι το άθροισμα μιας τεράστιας λίστας από ζεύγη <k2, v2>. Όλα τα ζεύγη που μοιράζονται το ίδιο k2 ομαδοποιούνται μαζί σε ένα καινούργιο ζεύγος (key/value) <k2, list<v2)>. Το πλαίσιο ζητάει από το reducer να επεξεργαστεί κάθε ένα από αυτά τα σύνολα ζευγών (key/value) ξεχωριστά. Ακολουθώντας το παράδειγμα word counting, η έξοδος της συνάρτησης map για ένα έγγραφο με ένα ζεύγος <”foo”, 1> τρεις φορές, και η έξοδος map για άλλο έγγραφο θα είναι μία λίστα με το ζευγάρι <”foo”, 1> δύο φορές. Το σύνολο των ζευγών που θα παραλάβει ο reducer είναι το <”foo”, list(1,1,1,1,1)>. Στον μετρητή λέξεων, η έξοδος του reducer θα είναι <”foo”, 5>, το οποίο είναι ο συνολικός αριθμός των φορών που η foo εμφανίστηκε στο σύνολο των εγγράφων. Κάθε reducer δουλεύει με μια διαφορετική λέξη. Το πλαίσιο MapReduce αυτόματα συλλέγει όλα τα ζευγάρια <k3, v3> και τα καταγράφει σε αρχεία. Να σημειώσουμε ότι για το παράδειγμα του μετρητή λέξεων οι τύποι δεδομένων k2 και k3 είναι ίδιοι όπως και οι v2 και v3. Αυτό βέβαια δεν είναι δεδομένο για άλλες εφαρμογές επεξεργασίας δεδομένων.

Ας ξαναγράψουμε το πρόγραμμα του μετρητή λέξεων στο MapReduce για να δούμε πώς όλα τα προηγούμενα συνδυάζονται μεταξύ τους.

```
map(String filename, String document){
      List<String> T = tokenize(document);
      for each token in T {
            emit((String)token, (Integer)1);
      }
}
reduce(String token, List<Integer> value){
      Integer sum=0;
      for each value in values{
            sum= sum + value;
      }
```

```
            emit((String)token, (Integer)sum);
        }
```

Όπως αναφέραμε στα προηγούμενα η έξοδος για τις συναρτήσεις map και reduce είναι λίστες. Όπως μπορούμε να δούμε από τον ψευδοκώδικα στην πράξη χρησιμοποιούμε μια ειδική συνάρτηση που ονομάζεται emit() για να δημιουργούνται στοιχεία στη λίστα κάθε φορά. Αυτή η συνάρτηση emit() επιπλέον απαλλάσσει τον προγραμματιστεί από τη διαχείριση μιας μεγάλης λίστας.

2.6 Τρόποι λειτουργίας του Hadoop

Τρεις είναι οι τρόποι λειτουργίας του Hadoop και τα βασικά αρχεία διαμόρφωσης σε κάθε περίπτωση είναι τα core-site.xml, mapred-site.xml και hdfs-site.xml [2].

- Local (standalone) mode
- Pseudo-distributed mode
- Fully distributed mode

Local (standalone) mode – Είναι η προεπιλεγμένη λειτουργία του Hadoop. Με την εγκατάσταση της εφαρμογής Hadoop, αγνοείται η εγκατάσταση του υλικού. Το Hadoop συντηρητικά επιλέγει την ελάχιστη διαμόρφωση και τα τρία XML αρχεία είναι άδεια στο πλαίσιο της παρούσας προεπιλεγμένης λειτουργίας, όπως φαίνεται παρακάτω:

```
<?xml version=”1.0”?>
<?xml-stylesheet type=”text/xsl” href=”configuration.xsl”?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
</configuration>
```

Το Hadoop εκτελείται στο τοπικό μηχάνημα, γιατί δεν υπάρχει ανάγκη να επικοινωνήσει με άλλους κόμβους. Αυτή η λειτουργία δεν χρησιμοποιεί το HDFS,ούτε και τους δαίμονες του Hadoop. Κυρίως χρησιμοποιείται για ανάπτυξη και αποσφαλμάτωση των προγραμμάτων MapReduce.

Pseudo-distributed mode – Στη προκειμένη περίπτωση, το Hadoop τρέχει σε “συστοιχία ενός κόμβου” με όλους τους δαίμονες να τρέχουν σε ένα μηχάνημα. Χρησιμοποιείται για λόγους όπως στη προηγούμενη λειτουργία, αλλά και για έλεγχο χρησιμοποιημένης μνήμης, θέματα εισόδου και εξόδου στο HDFS, και για την αλληλεπίδραση των δαιμόνων. Στο ΠΑΡΑΡΤΗΜΑ Α περιγράφεται αναλυτικά αυτή η λειτουργία.

Fully distributed mode – Με τα πλεονεκτήματα που μας δίνει η κατανεμημένη επεξεργασία και αποθήκευση, αυτή είναι η λειτουργία που τα εκμεταλλεύεται όλα. Εδώ διακρίνουμε τους servers σε τρεις κατηγορίες:

- master – Ο master κόμβος της συστοιχίας στον οποίο τρέχουν οι NameNode και JobTracker δαίμονες

- backup – Ο server στον οποίο τρέχει ο Secondary NameNode δαίμονας.
- hadoop1, hadoop2, hadoop3, ... – Είναι οι slave κόμβοι της συστοιχίας στους οποίους τρέχουν οι DataNode και TaskTracker δαίμονες.

Αναλυτικά περιγράφεται η διαδικασία αυτή στο Παράρτημα Β.

2.7 Συστατικά του Hadoop

Τα συστατικά του Hadoop είναι ένα σύνολο από βιβλιοθήκες που υποστηρίζουν τα επιμέρους έργα του Hadoop. Είναι σημαντικό να αναφέρουμε ότι το HDFS δεν είναι ένα POSIX-συμβατό σύστημα αρχείων, το οποίο σημαίνει ότι δεν μπορούμε να έχουμε πρόσβαση σε αυτό, όπως θα μπορούσαμε με του Linux ή του Unix το σύστημα αρχείων. Για να έχουμε πρόσβαση στο HDFS, χρειάζεται να χρησιμοποιήσουμε εντολές της μορφής /bin/hdfs dfs <args>, όπου args θα είναι το όρισμα εντολής που θέλουμε να χρησιμοποιήσουμε στα αρχεία στο σύστημα αρχείων [4].

Μερικές HDFS εντολές κελύφους:

cat	Αντιγράφει τα αρχεία στην καθιερωμένη έξοδο (stdout).
chmod	Αλλάζει τις άδειες για εγγραφή και ανάγνωση σε ένα αρχείο ή σε ένα σύνολο αρχείων.
chown	Αλλάζει τον ιδιοκτήτη του αρχείου ή του συνόλου αρχείων.
copyFromLocal	Αντιγράφει ένα αρχείο από το τοπικό σύστημα αρχείων στο HDFS.
copyToLocal	Αντιγράφει ένα αρχείο από το HDFS στο τοπικό σύστημα αρχείων.
cp	Αντιγράφει HDFS αρχεία από τον ένα κατάλογο στον άλλο.
ls	Εμφανίζει τη λίστα των αρχείων σε ένα κατάλογο.
mkdir	Δημιουργεί ένα κατάλογο στο HDFS
mv	Μετακινεί αρχεία από τον ένα κατάλογο στον άλλο.
rm	Διαγράφει ένα αρχείο και το στέλνει στο κάδο ανακύκλωσης (trash)

2.8 Γλώσσες ανάπτυξης εφαρμογών στο Hadoop

Για την απόκρυψη της πολυπλοκότητας των προγραμματιστικών μοντέλων στο Hadoop, αναπτύχθηκαν γλώσσες προγραμματισμού τρίτης και τέταρτης γενιάς που μπορούν να τρέχουν πάνω στη στοίβα του Hadoop [4].

2 .8.1 Pig και PigLatin

Η γλώσσα προγραμματισμού Pig αρχικά αναπτύχθηκε από την Yahoo! Και επιτρέπει στους χρήστες του Hadoop να εστιάζουν στην ανάλυση μεγάλων συνόλων δεδομένων, μη ξοδεύοντας χρόνο στη συγγραφή mapper και reducer προγραμμάτων. Η γλώσσα αυτή είναι για κάθε είδος δεδομένων και αποτελείται από δύο συστατικά. Το πρώτο είναι η ίδια η γλώσσα, η οποία ονομάζεται PigLatin και το δεύτερο είναι ένα εκτελέσιμο περιβάλλον εκεί που το PigLatin πρόγραμμα εκτελείται. Εμείς από δω και στο εξής θα αναφερόμαστε στη γλώσσα με το όνομα Pig.

Με την Pig είναι εύκολο να γράψουμε mapper και reducer προγράμματα. Το πρώτο βήμα σε ένα πρόγραμμα Pig είναι να φορτώσουμε (LOAD) δεδομένα, τα οποία θα διαχειριστούμε από το HDFS. Μετά μπορούμε να τρέξουμε τα δεδομένα μέσω ενός συνόλου επεξεργασιών (TRANSFORM) (μετατρέπονται σε mapper και reducer task). Τέλος, εξάγουμε (DUMP) τα δεδομένα στην οθόνη ή τα αποθηκεύουμε (SAVE) σε ένα αρχείο κάπου.

LOAD

Ένα πρόγραμμα Pig, για να έχει πρόσβαση στα δεδομένα που βρίσκονται στο HDFS χρησιμοποιούμε την εντολή LOAD ‘data_file’ (όπου data_file ορίζουμε ένα HDFS αρχείο ή κατάλογο). Εάν έχουμε ορίσει ένα κατάλογο, όλα τα αρχεία του καταλόγου θα φορτωθούν στο πρόγραμμα. Εάν έχουμε ορίσει ένα αρχείο, το οποίο δεν μπορεί άμεσα να είναι προσβάσιμο από το πρόγραμμα, μπορούμε να προσθέσουμε με την εντολή USING μία μέθοδο ορισμένη από τον χρήστη, για να διαβάσει και να διερμηνεύσει τα δεδομένα.

TRANSFORM

Ο μετασχηματισμός είναι μία διαδικασία επεξεργασίας των δεδομώνων. Μπορούμε να φιλτράρουμε (FILTER) τις εγγραφές που δεν μας ενδιαφέρουν, να συνενώσουμε (JOIN) δύο σύνολα από αρχεία δεδομένων, να ομαδοποιήσουμε (GROUP) τα δεδομένα, να ταξινομήσουμε (ORDER) τα αποτελέσματα και πολλά άλλα. Το παρακάτω παράδειγμα παίρνει σαν είσοδο ένα αρχείο από Twitter feeds, επιλέγει μόνο εκείνα τα tweets που χρησιμοποιούν την αγγλική γλώσσα (en – iso_language_code), μετά τα ομαδοποιεί με βάση τον χρήστη που τιτιβίζει και εμφανίζει το άθροισμα των επαναλαμβανόμενων tweets του χρήστη.

```
L  = LOAD ‘hdfs://node/tweet_data’;
FL = FILTER L BY iso_language_code EQ ‘en’;
G  = GROUP FL BY from_user;
RT = FOREACH G GENERATE group, SUM(retweets);
```

DUMP και STORE

Εάν δεν έχουμε ορίσει τις εντολές DUMP και STORE, τα αποτελέσματα του προγράμματος Pig δεν δημιουργούνται. Τυπικά χρησιμοποιούμε την εντολή DUMP, η οποία στέλνει τα αποτελέσματα στην οθόνη, για να κάνουμε αποσφαλμάτωση στο πρόγραμμα. Όταν θέλουμε να αναλύσουμε τα αποτελέσματα, χρησιμοποιούμε την εντολή STORE.

Υπάρχουν τρεις τρόποι να τρέξουμε ένα Pig πρόγραμμα στο Hadoop: ενσωματωμένο σε ένα script, ενσωματωμένο σε ένα Java πρόγραμμα ή την γραμμή εντολών του Pig (Grunt shell).

2.8.2 HIVE

Η γλώσσα HIVE δημιουργήθηκε από το Facebook. Επιτρέπει σε SQL προγραμματιστές να γράφουν Hive Query Language (HQL) δηλώσεις όμοιες με τις SQL. Αν και δεν καταλαβαίνει όλες τις εντολές, είναι ακόμη χρήσιμη. Οι HQL δηλώσεις διασπόνται από την υπηρεσία HIVE σε MapReduce jobs και εκτελούνται στη Hadoop συστοιχία.

Μπορούμε να τρέξουμε HQL ερωτήματα με τρεις τρόπους: με γραμμή εντολών (Hive Shell), από μία JDBC και ODBC εφαρμογής με τη χρήση των Hive JDBC ODBC drivers ή με ένα Hive Thrift Client. Ο Hive Thrift Client είναι όπως κάθε άλλη client βάση δεδομένων που είναι εγκατεστημένη στον client υπολογιστή του χρήστη (ή σε ένα μεσαίου επιπέδου ενός τριών επιπέδων αρχιτεκτονική) και επικοινωνεί με τις υπηρεσίες του Hive που τρέχουν στον server. Μπορούμε να χρησιμοποιήσουμε τον Hive Thrift Client σε εφαρμογές γραμμένες σε C++, Java, PHP, Python & Ruby (όπως και η χρησιμοποίηση client-side γλωσσών με ενσωματωμένη SQL για πρόσβαση σε βάση δεδομένων όπως η DB2 ή η Netezza). Το παρακάτω παράδειγμα μας δείχνει τη δημιουργία ενός πίνακα, τη φόρτωση στον πίνακα των δεδομένων και ένα ερώτημα HQL.

```
CREATE TABLE Tweets(from_user STRING, userid BIGINT, tweettxt STRING, retweets
INT)
COMMENT ‘This is the Twitter feed table’
STORED AS SEQUENCEFILE;
LOAD DATA INPATH ‘hdfs://node/tweetidata’ INTO TABLE TWEETS;
SELECT from_user, SUM(retweets)
FROM TWEETS
GROUP BY from_user;
```

Όπως βλέπουμε η Hive μοιάζει πάρα πολύ με τον κώδικα παραδοσιακών βάσεων δεδομένων SQL. Ωστόσο, επειδή η Hive βασίζεται στο Hadoop και σε MapReduce λειτουργίες, υπάρχουν κάποιες διαφορές. Πρώτων, το Hadoop έχει σχεδιαστεί για μακρύ συνεχές σκανάρισμα, και επειδή η Hive βασίζεται στο Hadoop, μπορεί να έχουμε ερωτήματα με υψηλή καθυστέρηση (μερικών λεπτών). Αυτό σημαίνει ότι η Hive δεν είναι κατάλληλη για εφαρμογές που θέλουν γρήγορη ανταπόκριση, όπως θα περιμέναμε με μία βάση δεδομένων όπως η DB2. Τέλος, η Hive είναι read-based και ως εκ τούτου δεν είναι κατάλληλη για επεξεργασία συναλλαγών, που τυπικά συνεπάγεται σε ένα υψηλό ποσοστό λειτουργιών εγγραφής.

2.8.3 Jaql

Η Jaql είναι μία γλώσσα υποβολής ερωτημάτων σε Javascript Object Notation (JSON), αλλά υποστηρίζει πολύ περισσότερες γλώσσες ερωτημάτων από την JSON. Μας επιτρέπει να επεξεργαζόμαστε μαζί δομημένα και μη παραδοσιακά δεδομένα και παραχωρήθηκε από την IBM στη κοινότητα ανοιχτού κώδικα. Ιδιαίτερα, με την Jaql μπορούμε να χρησιμοποιήσουμε τις εντολές select, join, group, και να φιλτράρουμε τα δεδομένα που είναι αποθηκευμένα στο HDFS, μοιάζει πολύ με ένα μείγμα από Pig και Hive. Η Jaql γλώσσα έχει πάρει από πολλές προγραμματιστικές γλώσσες, συμπεριλαμβανομένου των Lisp, SQL, XQuery και Pig. Η Jaql είναι μία λειτουργική, δηλωτική γλώσσα ερωτημάτων που σχεδιάστηκε για επεξεργασία μεγάλων συνόλων δεδομένων. Για παραλληλισμό, η Jaql ξαναγράφει high-level ερωτήματα αποτελούμενα από MapReduce jobs.

Οι περισσότεροι προγραμματιστές επιλέγουν JSON format για ανταλλαγή δεδομένων, διότι είναι εύκολο στην ανάγνωσή του από τον άνθρωπο και για τη δομή του, που είναι εύκολο από τις εφαρμογές να το αναλύσουν ή να το επεξεργαστούν. Το JSON έχει χτιστεί πάνω από δύο τύπους δομών. Η πρώτη είναι μία συλλογή από name/value ζευγάρια, κάνοντάς το ιδανικό για το Hadoop. Αυτά τα ζευγάρια name/value μπορεί να είναι οτιδήποτε, αφού είναι απλό κείμενο (ένα άλλο κοινό σημείο με το Hadoop), τα οποία μπορεί να αποτελούν μία εγγραφή σε μία βάση δεδομένων, ένα αντικείμενο, ένα σχεσιακό πίνακα, και πολλά άλλα. Το δεύτερο είναι η δομή του JSON που είναι ικανή να δημιουργήσει μία ταξινομημένη λίστα από τιμές όπως ο πίνακας, η λίστα ή μία ακολουθία που μπορεί να έχουμε στα προγράμματά μας. Ένα αντικείμενο JSON αναπαρίσταται ως εξής: `{string:value}`, και ένας πίνακας ως: `[value, value, ...]`, όπου value μπορεί να είναι αλφαριθμητικό, αριθμός, JSON αντικείμενο ή άλλος JSON πίνακας. Παρακάτω βλέπουμε ένα κομμάτι ενός Twitter feed σε μορφή JSON:

```
Results: [
{
created_at: “Thurs, 14 Jul 2011 09:47:45 +0000”
from_user: “eatonchris”
geo: {
     coordinates: [
          43.334543,
          78.933333
          ]
          type: “Point”
     }
iso_language_code: “en”
text: “Reliance Life Insurance migrates ......”
retweet: 3
to_user_id: null
to_user_id_str: null
}
```

Και η Jaql και η JSON είναι record-oriented μοντέλα, και έτσι ταιριάζουν μαζί τέλεια. Σημειώνουμε ότι το JSON format δεν είναι το μόνο που υποστηρίζει η Jaql, η οποία

μπορεί να υποστηρίξει και άλλες ημιδομημένες πηγές δεδομένων όπως XML, CSV, flat αρχεία, relational δεδομένα και άλλα. Ωστόσο, θα χρησιμοποιήσουμε το παραπάνω JSON παράδειγμα με Jaql ερωτήματα. Όπως θα δούμε η Jaql μοιάζει πολύ με την Pig, αλλά επίσης έχει μερικές ομοιότητες με την SQL.

Jaql τελεστές

Η Jaql έχει χτιστεί πάνω σε βασικούς τελεστές. Θα δούμε μερικούς από τους πιο δημοφιλείς, και θα περάσουμε σε κάποια απλά παραδείγματα που θα χρησιμοποιήσουμε για αναζήτηση στο Twitter feed που είδαμε προηγουμένως.

FILTER - Ο τελεστής FILTER παίρνει για είσοδο έναν πίνακα και φιλτράρει τα στοιχεία του που ενδιαφέρουν ως προς κάποιο χαρακτηριστικό. Είναι η αντίστοιχη WHERE της SQL. Για παράδειγμα, εάν θέλουμε να ψάξουμε όλες τις εγγραφές του Twitter feed που δημιουργήθηκαν από τον χρήστη `eatonchris`, τότε το ερώτημα θα είναι ως εξής:

```
filter $.from_user == “eatonchris”
```

Εάν θέλουμε να δούμε μόνο τα tweets που έχουν retweets περισσότερα από δύο, τότε:

```
filter $.retweet > 2
```

TRANSFORM - Ο τελεστής TRANSFORM παίρνει έναν πίνακα για είσοδο και εξάγει ένα άλλο πίνακα όπου τα στοιχεία του πρώτου πίνακα έχουν μετατραπεί με κάποιο τρόπο. Είναι το αντίστοιχο SELECT. Για παράδειγμα, εάν ένα πίνακας εισόδου έχει δύο νούμερα N1 και N2, ο TRANSFORM τελεστής θα παράγει ένα άθροισμα αυτών:

```
transform {sum: $N1 + $N2}
```

GROUP - Ο GROUP τελεστής δουλεύει όπως η GROUP BY δήλωση, όπου ένα σύνολο από δεδομενα ομαδοποιούνται στην έξοδο. Για παράδειγμα, εάν θέλουμε να μετρήσουμε το συνολικό αριθμό των tweets τότε:

```
group into count ($)
```

Ομοίως, εάν θέλουμε το άθροισμα όλων των retweets ανά χρήστη τότε:

```
group by u = $.from_user into {total: sum($.retweet) };
```

JOIN – Ο JOIN τελεστής παίρνει δύο εισόδους και παράγει στην έξοδο ένα πίνακα βασισμένο στη σύζευξη που ορίζεται στη WHERE δήλωση. Ας υποθέσουμε ότι έχουμε ένα πίνακα από tweets και επίσης έχουμε ένα σύνολο από ενδιαφέροντα δεδομένα που έρχονται από μία ομάδα ανθρώπων τους οποίους ακολουθούμε στο Twitter. Ο πίνακας μπορεί να είναι ως εξής:

```
Following = {from_user: “eatonchris”},
               (from_user: “paulzikopoulos”}
```

Σε αυτό το παράδειγμα, θα χρησιμοποιήσουμε τον JOIN τελεστή για να συζεύξουμε τα Tweeter feed δεδομένα με τα Twitter following δεδομένα για να παραχθούν αποτελέσματα μόνο για τα tweets των ανθρώπων που ακολουθούν, έτσι έχουμε:

```
join feed, follow
where feed.from_user = following.from_user
into {feed.*}
```

EXPAND - Ο EXPAND τελεστής παίρνει έναν εμφωλευμένο πίνακα ως είσοδο και παράγει ένα μόνο πίνακα ως έξοδο. Ας υποθέσουμε ότι έχουμε ένα εμφωλευμένο πίνακα με γεωγραφικές συντεταγμένες όπως:

```
geolocation = [[93.564, 123.222],[21.768, 90.321]]
```

Σε αυτή τη περίπτωση η εντολή `geolocation -> expand;` θα επιστρέψει τον πίνακα ως εξής:

```
[93.564, 123.222, 21.768, 90.321]
```

SORT - Ο τελεστής SORT παίρνει ένα πίνακα για είσοδο και παράγει ένα πίνακα για έξοδο, όπου τα στοιχεία είναι σε ταξινομημένη σειρά. Η προεπιλεγμένη ταξινόμηση εδώ είναι η αύξουσα. Μπορούμε να έχουμε φθίνουσα σειρά με την δήλωση sort by desc.

TOP - Ο TOP τελεστής επιστρέφει τα πρώτα n στοιχεία του πίνακα εισόδου, όπου n είναι ένας `<integer>` που ακολουθεί την λέξη TOP.

Εκτός από τους βασικούς τελεστές επεξεργασίας, η Jaql έχει επίσης ένα μεγάλο σύνολο από built-in μεθόδους για να διαβάζουμε, για να διαχειριζόμαστε και για να γράφουμε δεδομένα, όπως και για τη κλήση εξωτερικών μεθόδων όπως HDFS κλήσεις και άλλα. Μπορούμε να προσθέσουμε τις δικές μας μεθόδους, οι οποίες να καλούν άλλες μεθόδους.

Όπως μία MapReduce job είναι ροή δεδομένων, η Jaql μπορεί να θεωρηθεί ως μία διασωλήνωση των δεδομένων που ρέουν από μία πηγή (source), μέσω ενός συνόλου από διάφορους τελεστές, και εξάγονται σε ένα προορισμό (sink). Ο τελεστής που χρησιμοποιείται για να δηλώσει τη ροή από τον ένα τελεστή στον άλλο είναι ένα τόξο: ->. Σε αντίθεση με την SQL, όπου η έξοδος έρχεται πρώτη (για παράδειγμα, η SELECT λίστα), στην Jaql, οι λειτουργίες είναι σε μία φυσική σειρά, εκεί που ορίζουμε τη πηγή, ακολουθούμενη από διάφορους τελεστές που θέλουμε να διαχειριστούμε τα δεδομένα, και τελικά ο προορισμός. Ας συνοψίσουμε όσα περιγράψαμε με ένα παράδειγμα:

```
$tweets = read(hdfs(“tweet_log”));
$tweets
-> filter $.iso_language_code = ‘en’
-> group by u = $.from_user
     into { user: $from_user, total: sum($.retweet)};
```

Η πρώτη γραμμή ανοίγει ένα αρχείο με δεδομένα από το HDFS, και του δίνει ένα όνομα `$tweets`. Μετά, διαβάζει τα `$tweets` και φιλτράρει τα δεδομένα που έχουν τη αγγλική γλώσσα. Έπειτα, αυτές οι εγγραφές ομαδοποιούνται περνώντας για κάθε χρήστη τη τιμή του αθροίσματος των retweets. Εσωτερικά, ο μηχανισμός της Jaql μετατρέπει το ερώτημα σε map και reduce tasks που μπορεί να επηρεάσει σημαντικά τη μείωση του χρόνου ανάπτυξης της εφαρμογής σε σχέση με την ανάλυση μεγάλων δεδομένων στο Hadoop. Η Jaql είναι επεκτάσιμη και ευέλικτη, επιτρέπει το πέρασμα δεδομένων μεταξύ της διεπαφής ερωτημάτων και μιας γλώσσα προγραμματισμού της επιλογής μας (για παράδειγμα: Java, JavaScript, Python, Perl, Ruby).

2.9 Hadoop Streaming

Πέρα από την Java, μπορούμε να γράψουμε map και reduce προγράμματα σε άλλες γλώσσες που χρησιμοποιούν το Hadoop Streaming API (εν συντομία streaming). Το streaming είναι βασισμένο στο UNIX Streaming, όπου η είσοδος είναι το stdin και η έξοδος είναι το stdout. Αυτές οι ροές δεδομένων αναπαριστούν την διεπαφή μεταξύ του Hadoop και των προγραμμάτων μας [4].

Η διεπαφή Streaming μας οδηγεί στο να δημιουργήσουμε σύντομα και απλά προγράμματα χρησιμοποιώντας διάφορες γλώσσες σεναρίων (π.χ. Python ή Ruby). Και αυτό επειδή είναι text-based φύσης τα δεδομένα που ρέουν, όπου κάθε γραμμή του κειμένου είναι μία εγγραφή. Το παρακάτω παράδειγμα δείχνει την εκτέλεση map και reduce προγραμμάτων γραμμένα σε Python χρησιμοποιώντας το Hadoop Streaming API.

```
hadoop jar contrib/streaming/Hadoop-streaming.jar \
         -input input/dataset.txt \
         -output output \
         -mapper text_processor_map.py \
         -reducer text_processor_reduce.py
```

Ένα παράδειγμα στη περίπτωση αυτή περιγράφεται αναλυτικά στο ΠΑΡΑΡΤΗΜΑ Γ.

2.10 Άλλα συστατικά του Hadoop

Υπάρχουν επιπλέον πολλά άλλα προγράμματα ανοιχτού κώδικα που είναι υποπρογράμματα του Hadoop ή υψηλού επιπέδου προγράμματα Apache [4].

2.10.1 ZooKeeper

Το ZooKeeper είναι ένα ανοιχτού κώδικα Apache project που παρέχει μία συγκεντρωτική υποδομή και υπηρεσίες που ενεργοποιούν τον συγχρονισμό σε μία συστοιχία. Το ZooKeeper διατηρεί τα κοινά αντικείμενα που χρειάζονται σε ένα περιβάλλον συστοιχίας. Παραδείγματα αυτών των αντικειμένων περιλαμβάνουν την πληροφορία διαμόρφωσης, ιεραρχικό namespace, και ούτω καθεξής. Οι εφαρμογές μπορούν να ενεργοποιούν αυτές τις υπηρεσίες για να συντονίσουν την κατανεμημένη επεξεργασία σε μεγάλες συστοιχίες.

Ας φανταστούμε μία συστοιχία Hadoop αποτελούμενη από 500 ή περισσότερους εξυπηρετητές. Εάν έπρεπε να διαχειριστούμε μία βάση δεδομένων σε μία συστοιχία με 10 εξυπηρετητές, ξέρουμε ότι υπάρχει ανάγκη για συγκεντρωτική διαχείριση ολόκληρης της συστοιχίας από άποψη υπηρεσιών ανάλυσης ονομάτων, υπηρεσίες ομάδας, υπηρεσίες συγχρονισμού και άλλα. Επιπρόσθετα, πολλά άλλα προγράμματα που διαχειρίζονται τις συστοιχίες Hadoop προαπαιτούν αυτές τις υπηρεσίες, οι οποίες είναι διαθέσιμες στο Zookeeper. Η αλληλεπίδραση με το Zookeeper αυτή τη στιγμή μπορεί να γίνει μέσω της Java ή της C.

Το Zookeeper παρέχει μία υποδομή για συγχρονισμό μεταξύ των κόμβων και μπορεί να χρησιμοποιηθεί από εφαρμογές για επιβεβαίωση, ότι οι εργασίες στη συστοιχία είναι serialized ή synchronized. Αυτό το κάνει διατηρώντας τη κατάσταση ως πληροφορία στη μνήμη των Zookeeper εξυπηρετητών. Ένας Zookeeper εξυπηρετητής είναι ένα μηχάνημα που κρατά ένα αντίγραφο της κατάστασης ολόκληρου του συστήματος και η πληροφορία αυτή παραμένει σε τοπικά αρχεία log. Μία μεγάλη συστοιχία Hadoop μπορεί να υποστηρίξει πολλούς Zookeeper εξυπηρετητές (σε αυτή τη περίπτωση, ένας master εξυπηρετητής συγχρονίζει τους top-level εξυπηρετητές). Κάθε υπολογιστής πελάτης επικοινωνεί με ένα από τους Zookeeper εξυπηρετητές για να λάβει και να ενημερώσει τη δική του πληροφορία συγχρονισμού.

Στη μνήμη του Zookeeper εξυπηρετητή υπάρχει ένα αρχείο που ονομάζεται znode. Το znode μπορεί να ενημερώνεται από κάθε κόμβο στη συστοιχία, και κάθε κόμβος στη συστοιχία μπορεί να καταγράφει τις αλλαγές που γίνονται στο znode. Χρησιμοποιώντας το znode, οι εφαρμογές μπορούν να συγχρονίσουν τις εργασίες τους στη κατανεμημένη συστοιχία, ενημερώνοντας την κατάστασή τους στον Zookeeper znode, ο οποίος θα ενημερώσει τους υπόλοιπους στη συστοιχία για την τρέχων κατάσταση του κόμβου. Αυτή η υπηρεσία είναι ουσιώδης για την διαχείριση και την σειριοποιήση εργασιών σε ένα μεγάλο κατανεμημένο σύνολο από εξυπηρετητές.

2.10.2 HBase

Η HBase είναι ένα column-oriented σύστημα διαχείρισης βάσης δεδομένων που τρέχει πάνω στο HDFS. Σε αντίθεση με τα συστήματα σχεσιακών βάσεων δεδομένων, η HBase δεν υποστηρίζει μία δομημένη γλώσσα ερωτημάτων όπως η SQL και δεν είναι μία αποθήκη σχεσιακών δεδομένων καθόλου. Οι HBase εφαρμογές είναι γραμμένες σε Java που μοιάζουν με μία τυπική εφαρμογή MapReduce. Η HBase υποστηρίζει εφαρμογές γραμμένες σε Avro, REST, και Thrift interfaces.

Ένα HBase σύστημα περιλαμβάνει ένα σύνολο από πίνακες. Κάθε πίνακας περιέχει γραμμές και στήλες, όπως και στις παραδοσιακές βάσεις δεδομένων. Κάθε πίνακας πρέπει να έχει ένα ορισμένο στοιχείο για πρωτεύων κλειδί. Μία HBase στήλη παριστά ένα χαρακτηριστικό ενός αντικειμένου; για παράδειγμα, εάν ο πίνακας αποθηκεύει διαγνωστικά logs από εξυπηρετητές στο περιβάλλον μας, όπου κάθε γραμμή είναι μία log εγγραφή και μία τυπική στήλη στο πίνακα να είναι μία χρονοσφραγίδα (timestamp) του χρόνου εγγραφής της log εγγραφής ή ίσως το όνομα του εξυπηρετητή (servername) από όπου η εγγραφή προήλθε. Η HBase επιτρέπει για πολλά χαρακτηριστικά να ομαδοποιούνται μαζί σε οικογένειες στηλών (column families), όπου τα στοιχεία μίας οικογένειας στηλών είναι όλα αποθηκευμένα μαζί. Αυτό είναι διαφορετικό από μία row-oriented σχεσιακή βάση

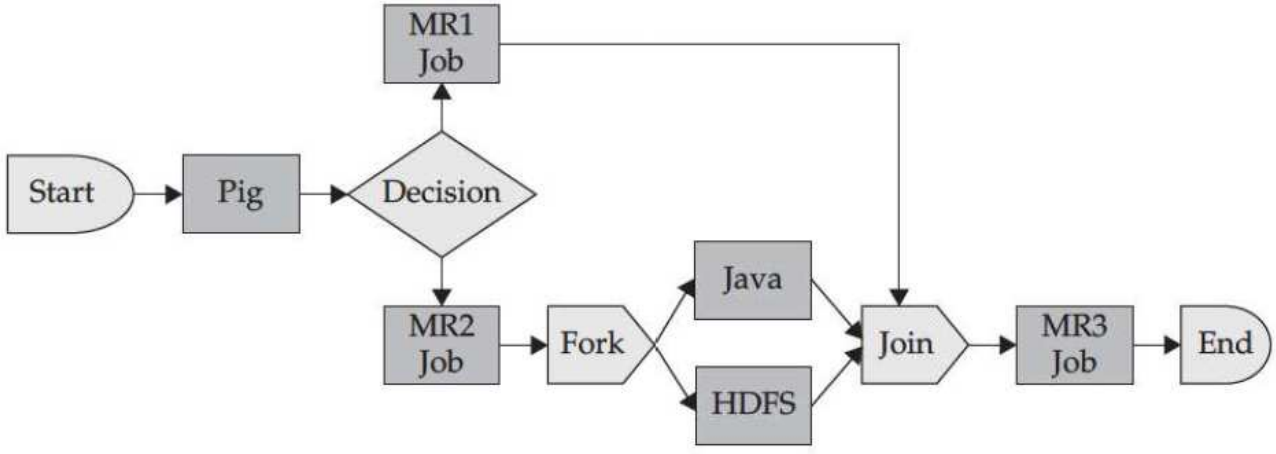
δεδομένων, όπου όλες οι στήλες μίας γραμμής είναι αποθηκευμένες μαζί. Με την HBase πρέπει να ορίζουμε εκ τον προτέρων το σχήμα του πίνακα και τις οικογένειες στηλών. Ωστόσο, υπάρχει ευελιξία με το να προσθέτουμε νέες στήλες στις οικογένειες κάθε στιγμή, κάνοντας το σχήμα ευέλικτο και ως εκ τούτου ικανό να προσαρμόζεται στις απαιτούμενες αλλαγές της εφαρμογής.

Όπως το HDFS έχει ένα NameNode και Datanodes slaves κόμβους, και το MapReduce έχει ένα JobTracker και TaskTrackers slaves, η HBase είναι χτισμένη με την ίδια λογική. Στην HBase ένας master κόμβος διαχειρίζεται τη συστοιχία και τους εξυπηρετητές της περιοχής (region servers) στους οποίους είναι αποθηκευμένα κομμάτια των πινάκων και εκτελούν τις εργασίες σχετικά με τα δεδομένα. Η HBase επίσης είναι ευάλωτη στην απώλεια του master κόμβου.

2.10.3 Oozie

Όπως έχουμε προφανώς παρατηρήσει στο MapReduce, πολλές jobs μπορεί να χρειαστεί να γίνουν αλυσιδωτά μαζί ώστε να ικανοποιήσουν πολύπλοκες εφαρμογές. Το Oozie είναι ανοιχτού κώδικα Apache project που απλοποιεί τη ροή εργασιών και τον συντονισμό μεταξύ των jobs. Παρέχει στους χρήστες τη δυνατότητα να ορίζουν ενέργειες και εξαρτήσεις μεταξύ ενεργειών. Το Oozie θα αναλάβει την εκτέλεση των ενεργειών, όταν θα προκύψουν οι απαιτούμενες εξαρτήσεις.

Μία ροή εργασιών στο Oozie ορίζειται στο Directed Acyclical Graph (DAG). Acyclical σημαίνει ότι δεν υπάρχουν επαναλήψεις στο γράφο (με άλλα λόγια, υπάρχει ένα σημείο αρχής και ένα σημείο τερματισμού στο γράφο), και όλες οι εργασίες και οι εξαρτήσεις δείχνουν από την αρχή ως το τέλος χωρίς να επιστρέφουν προς τα πίσω. Ένα DAG είναι φτιαγμένο από ενέργειες κόμβων και εξαρτημένους κόμβους. Μία ενέργεια κόμβου μπορεί να είναι μία MapReduce job, μία Pig εφαρμογή, μία εργασία του συστήματος αρχείων, ή μία Java εφαρμογή. Ο έλεγχος ροής στο γράφο αναπαρίσταται από στοιχεία τους κόμβους που ακολουθούν την λογική, ότι η είσοδος τους είναι η προηγούμενη εργασία στο γράφο.



Εικόνα 2.6 Μία Oozie ροή εργασιών που περιλαμβάνει πολλά σημεία απόφασης ως μέρος μίας απ'άκρου εις άκρον εκτέλεση.

Ο έλεγχος ροής μπορεί να προγραμματιστεί για την έναρξή του σε ένα δεδομένο χρόνο ή με την άφιξη ειδικών δεδομένων στο σύστημα αρχείων. Μετά την έναρξη, περαιτέρω ενέργειες στη ροή εργασιών εκτελούνται μετά την ολοκλήρωση των προηγούμενων ενεργειών στο γράφο. Στην Εικόνα 2.6 βλέπουμε ένα παράδειγμα μίας ροή εργασιών στο Oozie, όπου οι κόμβοι αναπαριστούν τις ενέργειες.

2.10.4 Lucene

Το Lucene είναι ένα ευρέως διάσημο ανοιχτού κώδικα Apache project για αναζήτηση κειμένου και περιλαμβάνεται σε πολλά ανοιχτού κώδικα projects. Το Lucene προηγείται του Hadoop και έχει γίνει ένα top-level Apache project από το 2005. Παρέχει πλήρη ευρετηρίαση κειμένου και αποτελείται από πολλές βιβλιοθήκες αναζήτησης που χρησιμοποιούνται σε Java εφαρμογές. (Αλλά και σε άλλες: C++, Python, Perl κ.τ.λ.).

Η διαδικασία είναι αρκετά απλή, ας πούμε ότι χρειαζόμαστε να ψάξουμε μέσα σε μία συλλογή από κείμενα. Το Lucene διασπάει αυτά τα κείμενα σε πεδία κειμένου και βάζει ένα δείκτη σε αυτά τα πεδία. Ο δείκτης είναι το κλειδί συστατικό του Lucene, καθώς χρησιμοποιείται για γρήγορες δυνατότητες αναζήτησης κειμένου. Μετά χρησιμοποιούμε τις μεθόδους αναζήτησης από του Lucene από τις βιβλιοθήκες για να βρούμε τα συστατικά του κειμένου.

2.10.5 Avro

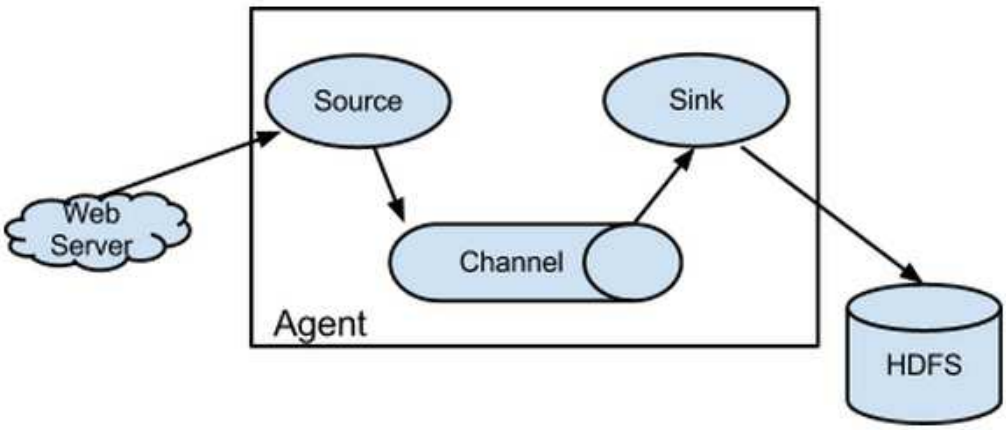
Το Avro είναι ένα Apache project που παρέχει υπηρεσίες σειριοποίησης (serialization) δεδομένων. Όταν γράφουμε Avro δεδομένα σε ένα αρχείο, το σχήμα που ορίζει τα δεδομένα είναι γραμμένο μέσα στο αρχείο. Αυτό διευκολύνει κάθε εφαρμογή να διαβάσει τα δεδομένα σε μεταγενέστερο χρόνο, επειδή το σχήμα που ορίζει τα δεδομένα αποθηκεύεται μαζί με το αρχείο. Ένα ακόμη πλεονέκτημα στη διαδικασία του Avro είναι ότι τα δεδομένα μπορούν να εκδοθούν με βάσει το γεγονός της αλλαγής του σχήματός τους σε μία εφαρμογή που μπορεί εύκολα να το χειριστεί επειδή το σχήμα για τα παλαιότερα δεδομένα παραμένει αποθηκευμένο μέσα στο αρχείο δεδομένων. Το σχήμα στο Avro ορίζεται με JSON.

Το σχήμα πιο συγκεκριμένα ορίζει τους τύπους δεδομένων που περιέχονται μέσα σε ένα αρχείο και είναι είναι έγκυρο καθώς τα δεδομένα είναι γραμμένα στο αρχείο χρησιμοποιώντας τα Avro APIs. Ομοίως, τα δεδομένα αφού διαμορφωθούν με βάση το ορισμένο σχήμα, διαβάζονται και επιστρέφουν πίσω στο αρχείο. Το σχήμα μας επιτρέπει να ορίσουμε δύο τύπους δεδομένων. Οι πρώτοι είναι οι primitive τύποι δεδομένων, όπως οι STRING, INT[eger], LONG, FLOAT, DOUBLE, BYTE, NULL και BOOLEAN. Οι δεύτεροι είναι οι σύνθετοι τύποι όπως είναι μία εγγραφή, ένας πίνακας, ένας απαριθμητής, μία map, μία ένωση ή ένας fixed τύπος.

Τα APIs του Avro είναι διαθέσιμα σε C, C++, C#, Java, Python, Ruby και PHP, κάνοντάς το διαθέσιμο στα περισσότερα περιβάλλοντα ανάπτυξης εφαρμογών που είναι κοινά στο Hadoop.

2.10.6 Flume

Το Flume είναι μία κατανεμημένη, αξιόπιστη και διαθέσιμη υπηρεσία για την αποτελεσματική συλλογή, την συνάθροιση, και τη μετακίνηση μεγάλων ποσοτήτων δεδομένων καταγραφής (logs). Έχει μία απλή και ευέλικτη αρχιτεκτονική που βασίζεται σε ροές δεδομένων, καθώς είναι ισχυρή και ανεκτική σε σφάλματα με ρυθμιζόμενους μηχανισμούς αξιοπιστίας και πολλούς μηχανισμούς ανάκτησης και εφεδρικής λειτουργίας (failover). Χρησιμοποιεί ένα απλό μοντέλο επέκτασης δεδομένων που επιτρέπει την απευθείας σύνδεσης ανάλυσης εφαρμογής. Στην Εικόνα 2.7 διακρίνουμε τα βασικά συστατικά μέρη μιας εφαρμογής Flume.



Εικόνα 2.7 Τα συστατικά μέρη μιας εφαρμογής Flume [21].

Το Flume θα λέγαμε ότι είναι ένα κανάλι που οδηγεί το νερό από τη πηγή σε ένα άλλο μέρος, όπου το νερό χρειάζεται. Το Flume δημιουργήθηκε για να επιτρέπει τη ροή δεδομένων από τη πηγή στο Hadoop περιβάλλον. Στο Flume, οι οντότητες με τις οποίες δουλεύουμε είναι οι sources, decorators και sinks. Μία source μπορεί να είναι οποιαδήποτε πηγή δεδομένων, και το Flume έχει ήδη ορισμένους source adapters. Ένας sink είναι ο στόχος της συγκεκριμένης διεργασίας (και στο Flume, μεταξύ άλλων παραδειγμάτων που χρησιμοποιούν αυτόν τον όρο, ο sink μίας διεργασίας μπορεί να είναι η πηγή για την επόμενη διεργασία παραπέρα). Ο decorator είναι μία διεργασία ροής δεδομένων που μπορεί να μεταφέρει το stream με κάποιο τρόπο, ο οποίος θα μπορούσε να είναι η συμπίεση ή αποσυμπίεση δεδομένων, τροποποίηση δεδομένων από πρόσθεση ή αφαίρεση κομματιών της πληροφορίας, και πολλά άλλα.

Ένας αριθμός από ήδη ορισμένους source adapters περιλαμβάνονται στο Flume. Για παράδειγμα, μερικοί adapters επιτρέπουν τη ροή του οτιδήποτε έρχεται από TCP θύρα να εισέλθει στη ροή, ή οτιδήποτε έρχεται από την stdin. Με ένα αριθμό από αρχεία κειμένου οι source adapters κάνουν λεπτομερέστερο έλεγχο για να πάρουν ένα ειδικό αρχείο και να το τροφοδοτήσουν στη ροή δεδομένων ή ακόμη να πάρουν την tail του αρχείου και συνεχώς να τροφοδοτούν τη ροή με οποιαδήποτε καινούργια δεδομένα βρίσκονται στο αρχείο αυτό. Το τελευταίο είναι πολύ χρήσιμο για τη διαγνωστική τροφοδότηση ή για τις καταγραφές ιστού σε δεδομένα ροής, που σταθερά επισυνάπτονται , και ο TAIL operator συνεχώς θα τραβά τις τελευταίες εισαγωγές από το αρχείο και θα τις βάζει στη ροή δεδομένων. Ένας αριθμός από άλλους εκ των προτέρων ορισμένους source adapters,

όπως η εντολή εξόδου, επιτρέπουν να χρησιμοποιήσουμε οποιαδήποτε εκτελέσιμη εντολή για την τροφοδότηση στη ροή των δεδομένων.

Υπάρχουν τρεις τύποι sinks στο Flume. Ο ένας sink είναι βασικά η τελική ροή προορισμού και είναι γνωστή ως Collector Tier_Event sink. Αυτό είναι εκεί που θα καταλήξει η ροή (ή πιθανόν πολλές ροές συζευγμένες μαζί) στο HDFS το σύστημα αρχείων. Ο άλλος sink τύπος που χρησιμοποιείται στο Flume ονομάζεται Agent Tier Event. Ο sink αυτός χρησιμοποιείται όταν θέλουμε να είναι ο ίδιος η πηγή εισόδου για άλλη εργασία. Όταν χρησιμοποιούμε αυτούς τους sinks, το Flume θα επιβεβαιώσει επίσης την ακεραιότητα της ροής στέλνοντας πίσω επιβεβαιώσεις ότι τα δεδομένα έχουν φτάσει στο προορισμό. Ο τελικός sink τύπος γνωστός ως Basic sink, οποίος μπορεί να είναι ένα αρχείο κειμένου, η κονσόλα οθόνης, ένα απλό HDFS μονοπάτι ή ένας άδειος κάδος (trush) όπου τα δεδομένα είναι διαγραμμένα.

2.11 SQL και Hadoop

Γνωρίζοντας ότι το Hadoop είναι ένα ανοιχτού κώδικα επαναχρησιμοποιήσιμη πλατφόρμα λογισμικού, αυτό που το κάνει καλύτερο από της κλασικές σχεσιακές βάσεις δεδομένων, είναι ότι η κινητήριος δύναμη της επεξεργασίας δεδομένων σήμερα είναι οι εφαρμογές. Ένας λόγος είναι ότι, η SQL (structured query language) έχει σχεδιαστεί για δομημένα δεδομένα. Πολλές αρχικές Hadoop εφαρμογές είχαν να κάνουν με μη-δομημένα δεδομένα όπως το κείμενο. Από αυτή την άποψη, το Hadoop παρέχει ένα πιο γενικό πρότυπο από την SQL [2].

Δουλεύοντας μόνο με δομημένα δεδομένα, η σύγκριση είναι πιο λεπτή. Καταρχήν, η SQL και το Hadoop μπορεί να είναι συμπληρωματικά, καθώς η SQL είναι μία γλώσσα ερωτημάτων, η οποία μπορεί να υλοποιηθεί πάνω στο Hadoop, ως η μηχανή εκτέλεσης. Αλλά στην πράξη, οι SQL βάσεις δεδομένων τείνουν να αναφέρονται σε ένα μεγάλο σύνολο από παλαιές τεχνολογίες, με αρκετούς κυρίαρχους προμηθευτές, από βελτιστοποιημένες χρονολογικά εφαρμογές. Πολλές όμως από τις υπάρχουσες εμπορικές βάσεις δεδομένων δεν συμφωνούν με τις προδιαγραφές, που το Hadoop στοχεύει.

Έχοντας αυτό υπόψη, ας κάνουμε μία πιο λεπτομερή σύγκριση του Hadoop με τις τυπικά SQL βάσεις δεδομένων σε συγκεκριμένες διαστάσεις

- **Επεκτασιμότητα (scale-out) αντί για αναβάθμιση (scale-up) –** Μιλώντας για επεκτασιμότητα των εμπορικών σχεσιακών βάσεων δεδομένων είναι μία σχετικά ακριβή επιλογή. Ο σχεδιασμός τους είναι περισσότερο φιλικός για αναβάθμιση. Για να τρέξει μία μεγαλύτερη βάση δεδομένων χρειάζεται να αγοράσουμε ένα μεγαλύτερο μηχάνημα (mainframe). Αυτού του είδους υπολογιστές δεν συμφέρουν οικονομικά. Για παράδειγμα, ένα μηχάνημα τέσσερις φορές πιο ισχυρό από τους επιτραπέζιους υπολογιστές (PC) κοστίζει περισσότερο από την τοποθέτηση επιτραπέζιων υπολογιστών σε μία συστοιχία. Το Hadoop έχει σχεδιαστεί για “scale-out” αρχιτεκτονική που λειτουργεί πάνω σε μία συστοιχία από επιτραπέζιους υπολογιστές. Προσθέτοντας πόρους στο σύστημα σημαίνει ότι προστίθενται επιτραπέζιοι υπολογιστές στη Hadoop συστοιχία. Συνήθως οι συστοιχίες αποτελούνται από δέκα έως εκατοντάδες τέτοιους υπολογιστές.

- **Ζεύγος κλειδί/τιμή (key/value) αντί για σχεσιακού πίνακες -** Βασική αρχή των σχεσιακών βάσεων δεδομένων είναι ότι, τα δεδομένα βρίσκονται σε πίνακες που έχουν σχεσιακή δομή οριζόμενη από ένα σχήμα (scheme). Ωστόσο, το σχεσιακό μοντέλο έχει συγκεκριμένες ιδιότητες και πολλές σύγχρονες εφαρμογές έχουν να κάνουν με τύπους δεδομένων που δεν ταιριάζουν στο μοντέλο αυτό. Έγγραφα κειμένου, εικόνες και XML αρχεία είναι πολύ γνωστά παραδείγματα. Επίσης, τα μεγάλα σύνολα δεδομένων είναι συνήθως μη-δομημένα ή ημι-δομημένα. Το Hadoop χρησιμοποιεί “key/value” ζεύγη ως τη βασική μονάδα δεδομένων, η οποία είναι ευέλικτη αρκετά για να δουλεύει με λιγότερο-δομημένα δεδομένα. Στο Hadoop, τα δεδομένα που μπορεί να προκύψουν σε οποιαδήποτε μορφή, τελικά μετατρέπονται σε “key/value” ζεύγη για λειτουργίες επεξεργασίας.

- **Λειτουργικό προγραμματισμό (functional programming - MapReduce) αντί για δηλωτικά ερωτήματα (SQL)** - Η SQL είναι ουσιαστικά μία υψηλού επιπέδου δηλωτική γλώσσα. Η αναζήτηση δεδομένων γίνεται με ορισμένο το αποτέλεσμα που θέλουμε και αφήνοντας την μηχανή βάσεως δεδομένων να τα αντλήσει. Σύμφωνα με το MapReduce καθορίζουμε τα βήματα για την επεξεργασία των δεδομένων, το οποίο είναι ανάλογο του μηχανισμό εκτέλεσης της SQL βάσης δεδομένων(SQL engine). Με την SQL έχουμε δηλώσεις ερωτημάτων, με το MapReduce έχουμε σενάρια και κώδικες. Το MapReduce μας επιτρέπει να επεξεργαζόμαστε κώδικα με ένα πολύ γενικό τρόπο από τα SQL ερωτήματα.

- **Εκτός σύνδεσης επεξεργασία δέσμης εντολών (batch processing) αντί για ηλεκτρονικές συναλλαγές (online transactions)** - Το Hadoop σχεδιάστηκε για εκτός σύνδεση επεξεργασία και ανάλυση μεγάλης κλίμακας δεδομένων. Δεν δουλεύει με τυχαίο διάβασμα ή εγγραφή σε μερικές εγγραφές, οι οποίες είναι του τύπου “On-line Transaction Processing” (OLTP). Το Hadoop χρησιμοποιείται ως “write-once, read-many-times” για τα αποθηκευμένα δεδομένα. Αυτό το σημείο είναι παρόμοιο με τις αποθήκες δεδομένων (Data Warehouses) στο κόσμο της SQL.

2.12 Η ιστορία του Hadoop

Το Hadoop ξεκίνησε ως ένα μέρος του project Nutch, το οποίο με τη σειρά του ήταν ένα υπο project του Apache Lucene. Ο Doug Cutting δημιούργησε όλα αυτά τα τρία projects και καθένα από αυτά ήταν μια λογική ακολουθία των προηγούμενων [2].

Το Lucene είναι μια βιβλιοθήκη για ευρετηρίαση πλήρους κειμένου και για αναζήτηση. Αν μας δοθεί μία συλλογή κειμένων ο προγραμματιστής μπορεί πολύ εύκολα να προσθέσει τη δυνατότητα αναζήτησης στο έγγραφο χρησιμοποιώντας τη μηχανή Lucene. Desktop search, enterprise search και πολλές domain-specific μηχανές αναζήτησης έχουν δημιουργηθεί χρησιμοποιώντας το Lucene. Το Nutch είναι η πιο φιλόδοξη επέκταση του Lucene. Αυτό που κάνει είναι να προσπαθεί να χτίσει μια ολοκληρωμένη μηχανή αναζήτησης στο διαδίκτυο χρησιμοποιώντας το Lucene ως βασικό συστατικό. Το Nutch περιέχει συντακτικούς αναλυτές για HTML, ένα web crawler, μία βάση δεδομένων συνδεδεμένου γράφου και διάφορα επιπλέον συστατικά απαραίτητα σε μια μηχανή

αναζήτησης στο διαδίκτυο. Ο Doug Cutting οραματίστηκε το Nutch σαν ένα εναλλακτικό μέσο σε σχέση με τις εμπορικές τεχνολογίες όπως της Google.

Ακόμη, έχοντας προσθέσει συστατικά όπως έναν crawler και έναν parser, μια μηχανή αναζήτησης στο διαδίκτυο διαφέρει από μια βασική μηχανή αναζήτησης εγγράφων στο θέμα της κλιμάκωσης. Ενώ το Lucene στοχεύει στο να ευρετηριάσει εκατομμύρια έγγραφα, το Nutch πρέπει να είναι ικανό να χειριστεί δισεκατομμύρια από ιστοσελίδες χωρίς να γίνεται υπερβολικά ακριβό στη λειτουργία. Το Nutch χρειάζεται ένα επιπλέον επίπεδο για το χειρισμό διαφόρων συνηθισμένων προκλήσεων, των κατανεμημένων διεργασιών, του πλεονασμού, των αποτυχιών και της εξισορρόπησης φόρτου.

Γύρω στο 2004, η Google εξέδωσε δύο έγγραφα που περιέγραφαν το δικό της σύστημα αρχείων (Google File System) και το πλαίσιο MapReduce. Η Google ισχυρίστηκε ότι χρησιμοποίησε αυτές τις δύο τεχνολογίες για να κλιμακώσει το δικό της σύστημα αναζήτησης. Ο Doug Cutting αμέσως αντιλήφθηκε ότι μπορούσε να εφαρμόσει ιδανικά αυτές τις τεχνολογίες στο Nutch, και έτσι η ομάδα του ανέπτυξε το νέο πλαίσιο και εισήγαγε το Nutch σε αυτό. Η νέα υλοποίηση ενίσχυσε άμεσα την κλιμάκωση του Nutch. Ξεκίνησε να χειρίζεται μια σειρά από εκατοντάδες εκατομμύρια ιστοσελίδες και μπορούσε να τρέξει σε συστοιχίες από δεκάδες κόμβους. Ο Doug Cutting συνειδητοποίησε ότι ένα τέτοιο project ειδικά για να πάρει σάρκα και οστά με τις δύο αυτές τεχνολογίες, ήταν επιβεβλημένο να έχουμε κλιμάκωση στο διαδίκτυο, και έτσι γεννήθηκε το Hadoop. Η Yahoo προσέλαβε τον Doug Cutting το 2006 για να εργαστεί με μια εξειδικευμένη ομάδα στην ανάπτυξη του Hadoop ως ένα ανοιχτού κώδικα project. Δύο χρόνια μετά, το Hadoop που έτρεχε σε δεκάδες χιλιάδες συστοιχίες του πυρήνα Linux ήταν το σύστημα παραγωγής για ευρετηρίαση στον ιστό.

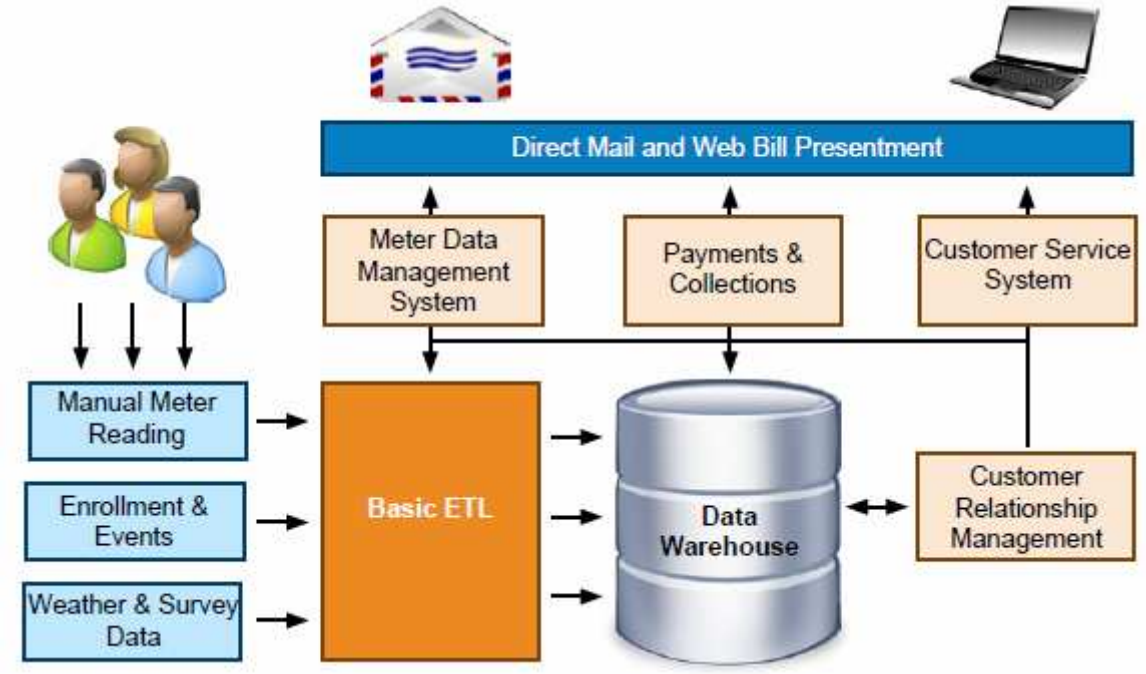
3. Hadoop και Data Warehouse

Τα τελευταία χρόνια, η αγορά πληροφορικής βιώνει μια σημαντική καινοτομία που κλονίζει ολόκληρη την υποδομή του κέντρου δεδομένων (data center). Η νέα Hadoop τεχνολογία έχει ξεκινήσει από ένα ταπεινό ξεκίνημα στην παγκόσμια υιοθέτηση κέντρων δεδομένων με νέες ιδέες υποδομών και τη δημιουργία νέων επιχειρηματικών ευκαιριών, τοποθετώντας παράλληλη επεξεργασία στα χέρια του μέσου προγραμματιστή.

Όπως συμβαίνει με όλες τις καινοτομίες της τεχνολογίας, η σύγχυση και τα επιπόλαια συμπεράσματα οδηγούν σε λανθασμένες απόψεις, όπως το ότι μερικοί υποστηρίζουν για παράδειγμα, ότι το Hadoop αντικαθιστά τις σχεσιακές βάσεις δεδομένων και γίνεται η νέα αποθήκη δεδομένων (Data Warehouse). Φυσικά, και δεν είναι έτσι, οι διαφορές τους ξεπερνούν τις ομοιότητες.

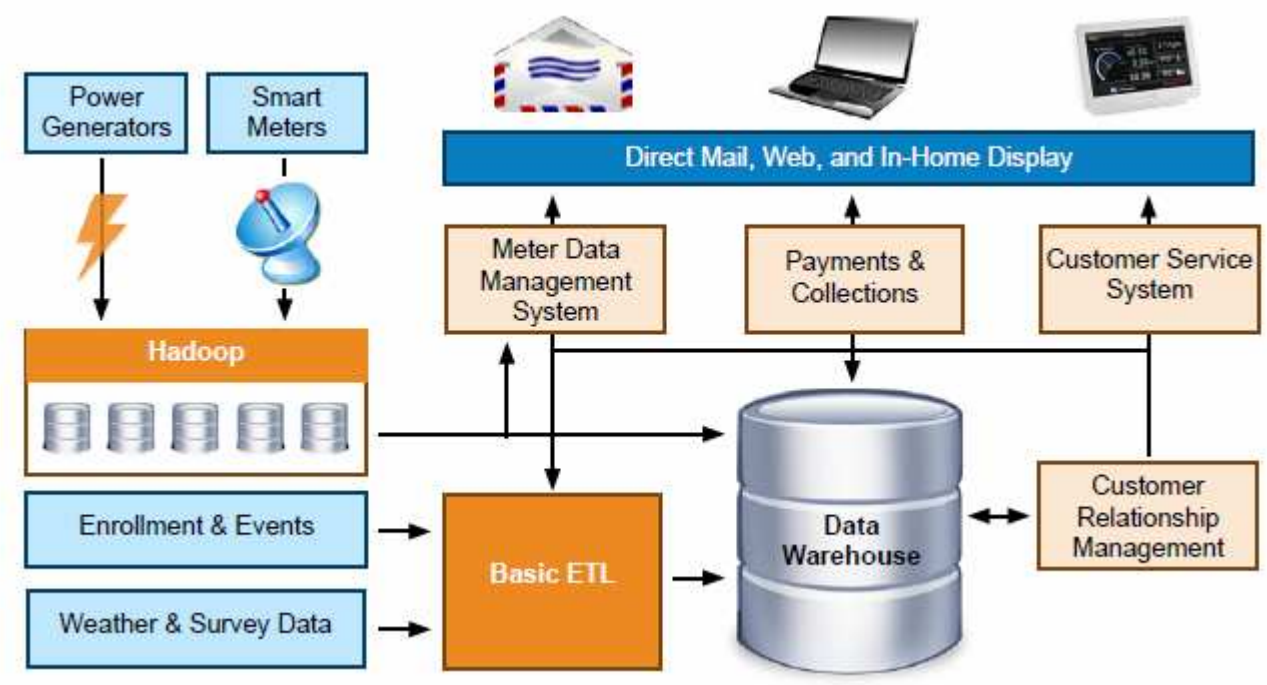
Προκειμένου, να αποσαφηνιστούν οι διαφορές μεταξύ του Hadoop και του DataWarehouse τεχνολογιών, θα χρησιμοποιήσουμε ως παράδειγμα τους έξυπνους μετρητές (Smart Meters), που αποτελούν πηγή μεγάλων δεδομένων [17].

Οι έξυπνοι μετρητές υπάρχουν σε σπίτια προκειμένου να βοηθήσουν τους καταναλωτές και να ωφελήσουν τις εταιρείες διαχείρισης κατανάλωσης νερού, ηλεκτρικού ρεύματος και φυσικού αερίου. Έως και σήμερα, υπάλληλοι της εκάστοτε εταιρείας πήγαιναν από σπίτι σε σπίτι, για να καταγράψουν τις μετρήσεις και να ενημερώσουν την εταιρεία προκειμένου να προβεί σε χρέωση κατανάλωσης του κάθε πελάτη της, βλέπε Εικόνα 3.1. Λόγω του κόστους εργασίας, πολλές επιχειρήσεις κοινής ωφέλειας αποφάσισαν να αλλάξουν τους μηνιαίους ελέγχους σε τετραμηνιαίους. Το γεγονός αυτό προξένησε καθυστερήσεις εσόδων και την ανάγκη ανάλυσης της οικιακή χρήσης, σε κάθε λεπτομέρεια.



Εικόνα 3.1 Πριν: Ροή Δεδομένων μίας καταγραφής μέτρησης χειροκίνητα.

Έστω ότι μία εταιρεία ηλεκτρικού ρεύματος εξυπηρετεί 10 εκατομμύρια νοικοκυριά. Μόνο το ένα τέταρτο από τα 10 εκατομμύρια μετρήσεις πραγματοποιήθηκαν για την έκδοση λογαριασμών κοινής ωφέλειας. Με νόμους της κυβέρνησης και της τιμής του πετρελαίου στα ύψη, η εταιρεία προχώρησε στην εγκατάσταση έξυπνων μετρητών, έτσι ώστε να παίρνει ωριαίες μετρήσεις κατανάλωσης ηλεκτρικού ρεύματος. Συλλέγουν τώρα 21,6 δισεκατομμύρια αναγνώσεις αισθητήρων ανά τετράμηνο από έξυπνους μετρητές. Η ανάλυση των δεδομένων των μετρητών επί μήνες και χρόνια, μπορεί να συνεχιστεί σε εκστρατείες εξοικονόμησης ενέργειας, με βάση τις καιρικές συνθήκες και άλλους παράγοντες, προς όφελος τόσο του καταναλωτή, όσο και της εταιρείας. Όταν όμως οι καταναλωτές έχουν επιλέξει ένα πρόγραμμα φθηνότερης ηλεκτρικής ενέργειας από της 8 μ.μ. έως τις 5 το πρωί, απαιτούνται διαστήματα αναφορών των πέντε λεπτών από τους έξυπνους μετρητές, ώστε να εντοπίσουν πότε γίνεται μεγάλη κατανάλωση ενέργειας στα σπίτια τους. Κατά διαστήματα πέντε λεπτών, οι έξυπνοι μετρητές συλλέγουν πάνω από 100 δις. ενδείξεις του μετρητή κάθε 90 ημέρες, και η εταιρεία έχει πλέον ένα Big Data πρόβλημα. Ο όγκος των δεδομένων υπερβαίνει την ικανότητα επεξεργασίας τους από το υπάρχον λογισμικό και υλικό. Έτσι η εταιρεία στρέφεται προς το Hadoop για να χειριστεί τις εισερχόμενες ενδείξεις μετρητή.



Εικόνα 3.2 Μετά: Ο μετρητής διαβάζει τις ενδείξεις κάθε 5 ή 60 λεπτά μέσω έξυπνων μετρητών

Το Hadoop σήμερα έχει ρόλο κλειδί στη σύλληψη, στη μεταφορά, και στη διαχείριση των δεδομένων. Χρησιμοποιώντας εργαλεία όπως το Apache Pig, τέτοιες προχωρημένες μεταφορές μπορούν να γίνουν στο Hadoop καταβάλλοντας τη λιγότερη χειροκίνητη προγραμματιστική προσπάθεια; μιας και το Hadoop είναι ένα χαμηλού κόστους αποθήκευσης αποθετήριο, τα δεδομένα μπορούν να βρίσκονται εκεί για μήνες ή ακόμη και χρόνια. Από τότε το Hadoop χρησιμοποιείται για τη μεταφορά των δεδομένων, τα οποία έπειτα οδηγούνται σε Data Warehouses και σε συστήματα MDMS (Meter Data Management Systems). Η εταιρεία κάνει πρόσθετες προσφορές στους πελάτες για την εξοικονόμηση χρημάτων χρησιμοποιώντας αναλύσεις για τις τάσεις στο νοικοκυριό, τη

γειτονιά, την ώρα της ημέρας και τις τοπικές εκδηλώσεις. Έτσιμ ο έξυπνος μετρητής μπορεί να δώσει λεπτομερή γνώση για την κατανάλωση του ρεύματος, βλέπε Εικόνα 3.2.

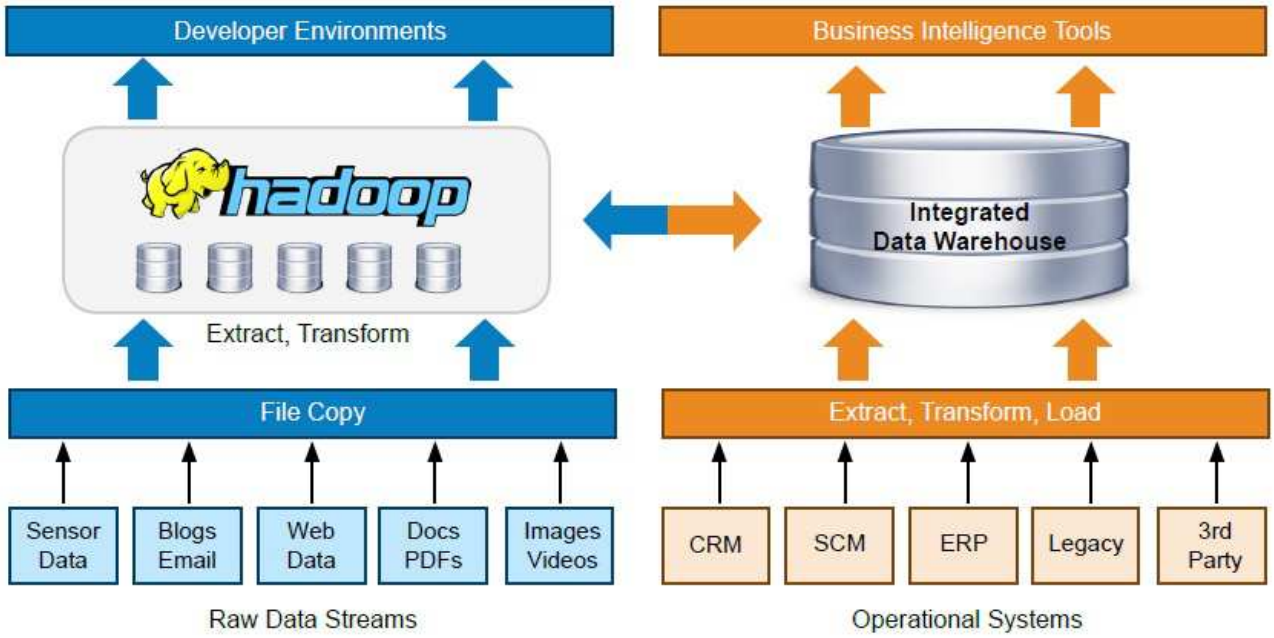
Σημειώνουμε ότι το Hadoop δεν είναι ένα Extract-Transform-Load (ETL) εργαλείο. Είναι μία πλατφόρμα που υποστηρίζει την εκτέλεση ETL διεργασιών παράλληλα. Το Hadoop απλά είναι ένα κανάλι για τη μεταφορά ομάδων δεδομένων.

Καθώς οι εταιρείες άρχισαν να χρησιμοποιούν μεγάλες ποσότητες δεδομένων, τα μετέφεραν μέσω δικτύου για μετασχηματισμό (μετατρέπει ένα σύνολο τιμών δεδομένων από τη μορφή δεδομένων ενός συστήματος δεδομένων της πηγής στην μορφή των δεδομένων ενός συστήματος δεδομένων του προορισμού) ή για ανάλυση το οποίο καθίσταται μη ρεαλιστικό. Μεταφέροντας TeraBytes από το ένα σύστημα στο άλλο καθημερινά δεν είναι και η πάγια λύση αυξάνοντας έτσι το φόρτο του δίκτυου. Για αυτό το λόγο είναι σημαντική η λύση της μεταφοράς του κώδικα σε δεδομένα (“code-to-data”, Κεφάλαιο 2). Επιπλέον έχοντας βάλει τα μεγάλα δεδομένα σε ένα χώρο αποθήκευσης του δικτύου SAN (Storage Area Network) ή σε ένα ETL server αρχίζει το σύστημα αυτό να λειτουργεί για αυτό τον όγκο δεδομένων. Να έχουμε όμως υπόψην ότι, όταν κινούμε τα δεδομένα με αργή επεξεργασία, περιοριζόμαστε από το εύρος ζώνης αποθήκευσης του SAN, που συχνά αποτυγχάνει να αντιμετωπίσει όλη αυτή τη μαζική επεξεργασία. Με το Hadoop, ακατέργαστα (raw) δεδομένα φορτώνονται άμεσα σε χαμηλού κόστους εξυπηρετητές μία φορά, και μόνο η πληροφορία με την υψηλότερη αξία περνά σε άλλα συστήματα. Οι ETL διεργασίες τρέχουν παράλληλα σε όλη τη συστοιχία, με αποτέλεσμα οι πολύ γρήγορες επεξεργασίες μπορούν να πετύχουν την προώθηση των δεδομένων από ένα SAN σε μία συλλογή από ETL servers. Χρησιμοποιώντας το Hadoop πρέπει να επισημάνουμε ότι, τα δεδομένα δεν φορτώνονται σε ένα SAN μόνο για να προωθηθούν έξω από αυτό σε όλο το δίκτυο πολλές φορές για κάθε μετασχηματισμό. Δεν πρέπει να μας εκπλήσσει ότι πολλά συστήματα Hadoop βρίσκονται δίπλα σε Data Warehouses. Αυτά τα συστήματα εξυπηρετούν διάφορους σκοπούς και συμπληρώνουν το ένα το άλλο.

Μία μεγάλη μεσιτική εταιρεία χρησιμοποιεί το Hadoop για την επεξεργασία raw click streams που δημιουργούνται από τους πελάτες στο website της εταιρείας. Με την επεξεργασία αυτών των click streams , παρέχεται γνώση σημαντική για τις προτιμήσεις του πελάτη, η οποία περνάει σε μία DataWarehouse. Η DataWarehouse συνδυάζει αυτές τις προτιμήσεις των πελατών με τις εκστρατείες marketing και τους μηχανισμούς συστάσεων για να προσφέρει επενδυτικές προτάσεις και ανάλυσεις για τους καταναλωτές.

Μία eCommerce υπηρεσία χρησιμοποιεί το Hadoop για μηχανική μάθηση με σκοπό την ανίχνευση απάτης σε website προμηθευτών. Αυτά τα websites εκθέτουν πρότυπα που το Hadoop χρησιμοποιεί για να παράγει ένα προβλέψιμο μοντέλο. Το μοντέλο αυτό αντιγράφεται σε μία Data Warehouse όπου χρησιμοποιείται για εύρεση δραστηριοτήτων πωλήσεων που ταιριάζουν στο πρότυπο του website. Όταν βρεθεί ο προμηθευτής θα διερευνηθεί και ενδεχομένως θα διακοπεί η χρήση του website.

Τελικά, είναι αυξανόμενη η χρήση λύσεων με το Hadoop και το MapReduce σε συνδυασμό με τις Data Warehouses στις επιχειρήσεις. Στην Εικόνα 3.3 διακρίνουμε τις πηγές δεδομένων, την ανταλλαγή δεδομένων και την ποικιλομορφία των δεδομένων, στα οποία μπορούν να έχουν πρόσβαση και τα δύο συστήματα.



Εικόνα 3.3 Η βασική αρχιτεκτονική συστημάτων δεδομένων των επιχειρήσεων σήμερα.

Πολύπλοκες Hadoop εργασίες μπορούν να χρησιμοποιήσουν τη Data Warehouse ως πηγή δεδομένων, ταυτόχρονα, αξιοποιώντας τις δυνατότητες επεξεργασίας μεγάλων δεδομένων στα δύο συστήματα. Κάθε MapReduce πρόγραμμα μπορεί να υποβάλει SQL ερωτήματα στη Data Warehouse. Βάζοντάς τα σε μία σειρά, ένα MapReduce είναι ένα ακόμη πρόγραμμα και μία Data Warehouse είναι μία ακόμη βάση δεδομένων. Ας φανταστούμε ότι, 100 MapReduce προγράμματα αποκτούν πρόσβαση ταυτόχρονα σε 100 Data Warehouse κόμβους παράλληλα. Μαζί τα ακατέργαστα δεδομένα αλλά και τα δεδομένα από τις Data Warehouse, αποτελούν πρόκληση για την επεξεργασία μεγάλων δεδομένων. Αυτό εκμεταλλεύονται σήμερα οι επιχειρήσεις για να επιτύχουν ανταγωνιστικά πλεονεκτήματα.

3.1 Data Warehouse

Έχοντας μιλήσει για το Hadoop, πρέπει να αναφερθούμε ειδικότερα και στη τεχνολογία του Data Warehouse για να κατανοήσουμε εις βάθος την αρχιτεκτονική των συστημάτων που ακολουθούν οι επιχειρήσεις σήμερα.

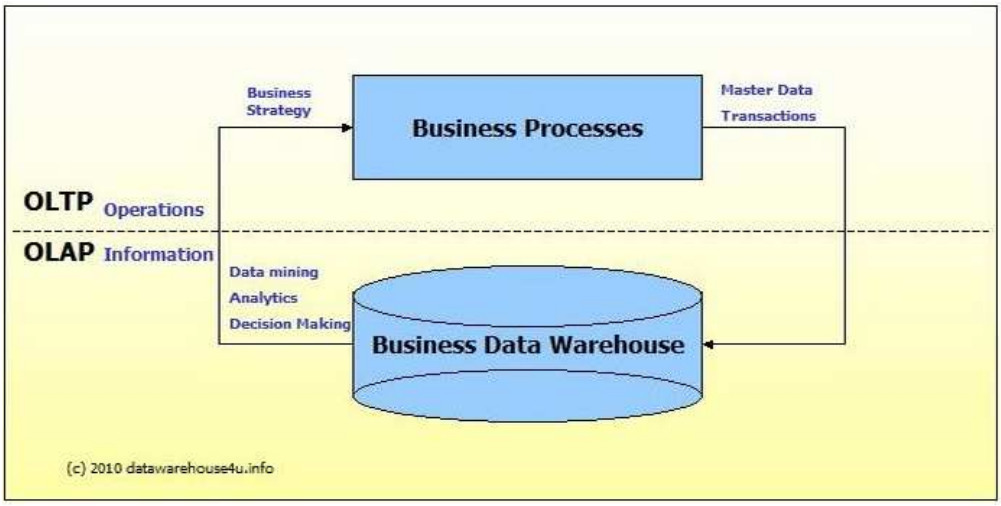
Οι οργανισμοί τη δεκαετία του ‘80 παρατήρησαν τη σημαντικότητα χρήσης των δεδομένων όχι μόνο για επεξεργασία, αλλά επίσης για την παροχή πληροφοριών από αυτά (Business Intelligence). Αυτή η παροχή πληροφορίας δεν θα δικαιολογούσε μόνο προηγούμενες αποφάσεις, αλλά επίσης θα βοηθούσε στη λήψη αποφάσεων για το μέλλον. Καθώς οι BI εφαρμογές αναδύθηκαν, γρήγορα προέκυψε το ζήτημα του ότι τα δεδομένα από τη βάση δεδομένων συναλλαγών έπρεπε πρώτα να μετασχηματιστούν (transform) και να αποθηκευθούν σε άλλη βάση δεδομένων με ένα ειδικό σχήμα κατάλληλο για την παραγωγή πληροφορίας. Αυτή η βάση δεδομένων θα μπορούσε να χρησιμοποιηθεί για αρχειοθέτηση, θα ήταν μεγαλύτερη από τη βάση δεδομένων συναλλαγών και θα έτρεχαν εκθέσεις που θα βοηθούσαν τους μεγάλες εταιρείες στη λήψη προληπτικών αποφάσεων. Αυτή η ξεχωριστή βάση, συνήθως αποθηκεύει το παρελθόν και το παρόν της εταιρείας.

Όμοια με την πραγματική αποθήκη, μία Data Warehouse συγκεντρώνει τα δεδομένα από μία κεντρική πηγή, τυπικά από μία βάση δεδομένων συναλλαγών και έπειτα αποθηκεύει και κατανέμει αυτά τα δεδομένα με τέτοιο τρόπο ώστε να γίνονται εύκολα αναλύσεις και να δημιουργούνται εκθέσεις. Η διαφορά μεταξύ μίας τυπικής βάσης δεδομένων και μίας Data Warehouse δεν βασίζεται μόνο στον όγκο των δεδομένων που μπορούν να αποθηκευθούν αλλά επίσης στον τρόπο που αποθηκεύονται (indexes, cubes κ.α.).

Από το να έχουμε πολλά συστήματα ανεξάρτητα για τη λήψη αποφάσεων, τα οποία οδηγούν στη σύγκρουση της πληροφορίας, μία Data Warehouse ενώνει όλες αυτές τις πηγές. Τα δεδομένα αποθηκεύονται με τέτοιο τρόπο ώστε η ακεραιότητα και η ποιότητά τους να είναι εγγυημένη. Αυτό επιτυγχάνεται με τη χρήση μίας Extract, Transform, Load διαδικασίας, γνωστή ως ETL (συνοπτικά, ο βασικός στόχος του ETL είναι να φιλτράρει τα πλεονάζοντα δεδομένα που δεν απαιτούνται στη ανάλυση και να δημιουργηθούν έτσι γρήγορα εκθέσεις). Μαζί με τα BI εργαλεία, τα οποία συλλέγουν και παρουσιάζουν τα δεδομένα στις κατάλληλες μορφές, αυτός ο συνδυασμός παρέχει μία ισχυρή λύση για την απόκτηση γνώσης.

3.1.1 Διαφορά: Data Warehouse και συστήτατα OLTP

Τα IT (Information Technology) συστήματα διακρίνονται σε OLTP (OnLine Transactional Processing) και σε OLAP (OnLine Analytical Processing) [19]. Γενικά, μπορούμε να πούμε ότι τα OLTP συστήματα είναι πηγές που παρέχουν δεδομένα σε Data Warehouses, όπου εκεί τα OLAP συστήματα βοηθούν στην ανάλυσή τους. Βλέπε Εικόνα 3.5.



Εικόνα 3.5 Συστήματα OLTP και OLAP

OLTP (On-line Transaction Processing) - Το σύστημα αυτό αναφέρεται σε εργασίες που έχουν πρόσβαση στα δεδομένα τυχαία. Τυπικά τα OLTP συστήματα εκτελούν αποδοτικά τα κλασικά ερωτήματα select, insert, update και delete, διατηρώντας την ακεραιότητα των δεδομένων σε πολλαπλής πρόσβασης περιβάλλοντα και η αποτελεσματικότητά τους μετριέται με βάση των αριθμό συναλλαγών το δευτερόλεπτο. Τα δεδομένα εδώ είναι κανονικοποιημένα (συνήθως σε 3NF), δηλαδή μειώνεται ή εξαλείφεται ο πλεονασμός τους, ενώ ταυτόχρονα εξασφαλίζεται η συνοχή τους.

OLAP (On-line Analytical Processing) – Το σύστημα αυτό χαρακτηρίζεται από σχετικά χαμηλού όγκου συναλλαγών. Τα ερωτήματα που συχνά τρέχουν σε αυτό το σύστημα είναι πολύπλοκα και περιλαμβάνουν συσσωματώσεις (aggregations). Για τα OLAP συστήματα ο χρόνος απόκρισης είναι ένα μέτρο αποτελεσματικότητας. Οι OLAP εφαρμογές χρησιμοποιούνται ευρέως με Data Mining τεχνικές. Σε μία OLAP βάση δεδομένων υπάρχουν ιστορικά δεδομένα συναλλαγών όπως και σε μία OLTP βάση δεδομένων, όπου αυτά τα δεδομένα έχουν μεταφερθεί μέσω της ETL διαδικασίας και είναι αποθηκευμένα σε πολυδιάστατο σχήμα. Με αυτό τον τρόπο βελτιστοποιείται η απόδοση για την δημιουργία αναφορών και περαιτέρω αναλύσεων. Τα OLAP συστήματα είναι ρυθμισμένα για γρήγορη ανάγνωση μόνο. Σε σύγκριση μετα OLTP συστήματα, τα δεδομένα σε μία OLAP Data Warehouse είναι λιγότερο κανονικοποιημένα από ένα σύστημα OLTP. Το μεγάλο πλεονέκτημα αυτής της προσέγγισης είναι ότι κάνει τη βάση δεδομένων αναγνώσιμη και γρήγορη στην αναζήτηση δεδομένων.

3.2 Επιλέγοντας μεταξύ Hadoop και Data Warehouse

Από τότε που εκτελούνται εργασίες για δεδομένα στο Hadoop και στη Data Warehouse, είναι καλύτερα να δούμε ποιες είναι οι απαιτήσεις που κάνουν κάθε μία από αυτές τις πλατφόρμες να αποδώσουν γρήγορα, εύκολα και με το λιγότερο κόστος κάθε φορά. Τα κέντρα δεδομένων που χρησιμοποιούν και τις δύο τεχνολογίες αναπτύσσουν την ικανότητα να γνωρίζουν πότε να χρησιμοποιηθεί ποια.

Το Data Warehouse και το Hadoop δεν διαφέρουν πολύ. Κάθε εργαλείο θα μπορούσε να είναι η σωστή λύση. Η επιλογή του καλύτερου εργαλείου, εξαρτάται από τις απαιτήσεις της κάθε εταιρείας. Σε πολλές περιπτώσεις, το Hadoop και το Data Warehouse συνεργάζονται ως μία αλυσίδα εφοδιασμού πληροφορίας για ένα συγκεκριμένο φόρτο εργασίας.

Υπάρχουν διάφορα προσωρινά σύνολα δεδομένων που είναι ακατάλληλα για τη Data Warehouse. Τα προσωρινά σύνολα δεδομένων αναλύονται ή χρησιμοποιούνται μεμονωμένα. Αντιπροσωπεύουν μία χρονική στιγμή όπως ορίζεται από τους επιστήμονες δεδομένων και δεν αντλούν την πρωταρχική τους αξία μέσω της ενοποίησης με άλλα σύνολα δεδομένων. Παραδείγματα των προσωρινών δεδομένων είναι η επιστημονική ανάλυση των δεδομένων στην αστρονομία και στη φυσική. Τυπικά ο επιστήμονας δεδομένων έχει ένα τεράστιο σύνολο δεδομένων για τη μελέτη και από τη στιγμή που βρει τα αποτελέσματα, τα δεδομένα μπορούν να διατηρούνται για ένα μήνα περίπου. Αυτό το είδος των δεδομένων που δεν ταιριάζει στη Data Warehouse, ενώ το Hadoop ωφελεί τέτοιου είδους ερευνητικά έργα.

Άλλα είδη προσωρινών δεδομένων περιλαμβάνουν διευθύνσεις διαδικτύου URL και ιστοσελίδες. Το πρόβλημα της ευρετηριακής αναζήτησης (index search) οδήγησε την Google στη δημιουργία της MapReduce λύσης και στη συνέχεια στην ανάπτυξη ανοιχτού κώδικα με το Hadoop. Εταιρείες μηχανών αναζήτησης ανιχνεύουν τα δεδομένα στο διαδίκτυο, συμπεριλαμβανομένου URLs, tags, metadata και external links. Χρησιμοποιώντας αυτά τα δεδομένα, οι μηχανές αναζήτησης εκτελούν μαζικούς MapReduce υπολογισμούς για την κατάταξη και ευρετηριάση των URL, έτσι ώστε οι

επισκέπτες της μηχανής αναζήτησης να πάρουν την πιο σχετική απάντηση σε κάθε αίτημα. Η πρόκληση είναι ότι οι ιστοσελίδες και οι σύνδεσμοι URL αλλάζουν κάθε μέρα. Λαμβάνοντας υποψήν το τεράστιο όγκο των δεδομένων που συλλέγονται, είναι ανέφικτο να σωθεί ακόμη και το μισό από αυτό για περισσότερο από μία μέρα. Ως εκ τούτο, Hadoop είναι η μόνη βιώσιμη λύση για αυτά τα ήδη των προβλημάτων.

Αντίθετα, θεωρώντας μία τράπεζα που αναπτύσσεται δια απορροφήσεως μικρών περιφερειακών τραπεζών. Πριν από τη συγχώνευση, παίρνουν μαγνητικές ταινίες των λογαριασμών των καταναλωτών από τις περιφερειακές τράπεζες. Χρησιμοποιεί τα δεδομένα για να βρει την μακροπρόθεσμη αφοσίωση, την κερδοφορία κ.α. Χρησιμοποιώντας αυτές τις πληροφορίες, η αποδέκτρια τράπεζα θα διαπραγματευτεί για μια δίκαιη τιμή αγοράς – έχοντας μόνο τέσσερις εβδομάδες για ανάλυση. Αυτό καθιστά δύσκολη τη μοντελοποίηση των δεδομένων, την αναδόμησή τους καθώς και άλλων εργασιών απαραίτητων για την εισαγωγή των δεδομένων στη Data Warehouse. Το Hadoop έχει τα πλεονέκτημα της ευελιξίας και της εκτίμησης του χρόνου. Η Data Warehouse έχει το πλεονέκτημα ότι επιτρέπει στους λογαριασμούς της περιφερειακής τράπεζας να ενωθούν με τους υπάρχοντες λογαριασμούς, παρέχοντας γρήγορη αναγνώριση πελατών και επικαλύψεων και σύγκριση της ποιότητας του κάθε λογαριασμού με τους υπάρχοντες λογαριασμούς. Ποια είναι η καλύτερη πλατφόρμα δεδομένου ότι τόσο το Hadoop όσο και η Data Warehouse κάνουν την ίδια δουλειά; Εξαρτάται από το ποιες είναι οι ανάγκες της αποκτηθείσας τράπεζας και το προς τα πού θα δώσουν προτεραιότητα. Μία λύση είναι το Hadoop για την αποθήκευση και την βελτιστοποίηση των δεδομένων και στη συνέχεια απομακρύνοντας τα μη ενδιαφέροντα δεδομένα να εισέρχονται σε μία Data Warehouse για περαιτέρω ανάλυση και συγκρίσεις λογαριασμών.

Η αξία της ανάλυσης βάσεων δεδομένων είναι διπλή: Πρώτον, ο αλγόριθμος εξόρυξης πληροφορίας τρέχει παράλληλα, παρέχοντας γρήγορα αποτελέσματα και επιτρέποντας πολλές εξερευνήσεις την ημέρα. Δεύτερον, ο επιστήμονας δεδομένων δεν περιορίζεται για ανάλυση σε μικρά δείγματα δεδομένων. Μπορούν την κρίσιμη στιγμή να αποκτήσουν γρήγορα κα ακριβή αποτελέσματα. Τις δυνατότητες αυτές παρέχει το MapReduce που υλοποιείται στο Hadoop. Μία φανερή διαφορά είναι ότι στη Data Warehouse έχουμε καθαρά ενοποιημένα δεδομένα, εκεί που το Hadoop συχνά έχει raw δεδομένα σε ποσότητα. Κατά συνέπεια, ένας τρόπος για να επιλέξουμε μεταξύ του Hadoop και της Data Warehouse για την εξόρυξη πληροφορίας βασίζεται στα ίδια τα δεδομένα.

Ένας άλλος απλός τρόπος για να αποφασιστεί πότε να χρησιμοποιηθεί ποιο εργαλείο είναι το κατάλληλο, είναι να εξετάσουμε το είδος του φόρτου εργασίας και τον χρήστη. Οι περισσότεροι χρήστες των επιχειρήσεων μπορούν να αξιοποιήσουν διαδραστικά εργαλεία BI όπως reports, dashboards, ad-hoc και επαναληπτική ανάλυση. Πουθενά δεν είναι αυτό πιο φανερό από ότι στη χρήση του OLAP. Εάν η απαίτηση των επιχειρήσεων είναι το διαδραστικό περιβάλλον ανάλυσης, η Data Warehouse είναι η καλύτερη επιλογή.

Τόσο το Hadoop όσο και η Data Warehouse εξαρτώνται από την εκτέλεση σύνθετων μαζικών εργασιών που επεξεργάζονται τεράστιο όγκο δεδομένων. Όταν ένα πρόγραμμα πρέπει να τρέξει παράλληλα για την επίτευξη της επεκτασιμότητας, το πρόγραμμα είναι εξαιρετικά περίπλοκο, και το Hadoop έχει πολλά πλεονεκτήματα. Ωστόσο, για μια μεγάλης κλίμακας επεξεργασία απαιτείται ένας προγραμματιστής MapReduce. Σε αντίθεση, με πολλές σύνθετες εφαρμογές που τρέχουν το βράδυ μαζικά χρησιμοποιώντας μία Data

Warehouse. Αν και αυτές οι εφαρμογές μπορεί να είναι τόσο περίπλοκες όπως κάθε MapReduce πρόγραμμα, δεν εκτελούνται παράλληλα. Έτσι, η απαίτηση για να τρέξει οποιαδήποτε γλώσσα και οποιοδήποτε επίπεδο της πολυπλοκότητας του προγράμματος παράλληλα τάσσεται υπέρ Hadoop. Εάν το πρόγραμμα δεν χρειάζεται να τρέξει παράλληλα, η ευκολία προγραμματισμού SQL σε συνδυασμό με την Data Warehouse είναι ίσως η καλύτερη επιλογή. Τρέχοντας παράλληλα MapReduce προγράμματα και χρησιμοποιώντας SQL στην αποθήκη δεδομένων, αξιοποιεί τα πλεονεκτήματα και των δύο υποσυστημάτων.

Ο παρακάτω πίνακας αποτελεί καλή αρχή στο να πάρουμε την απόφαση πότε να χρησιμοποιήσουμε το Hadoop και πότε το Date Warehouse [16].

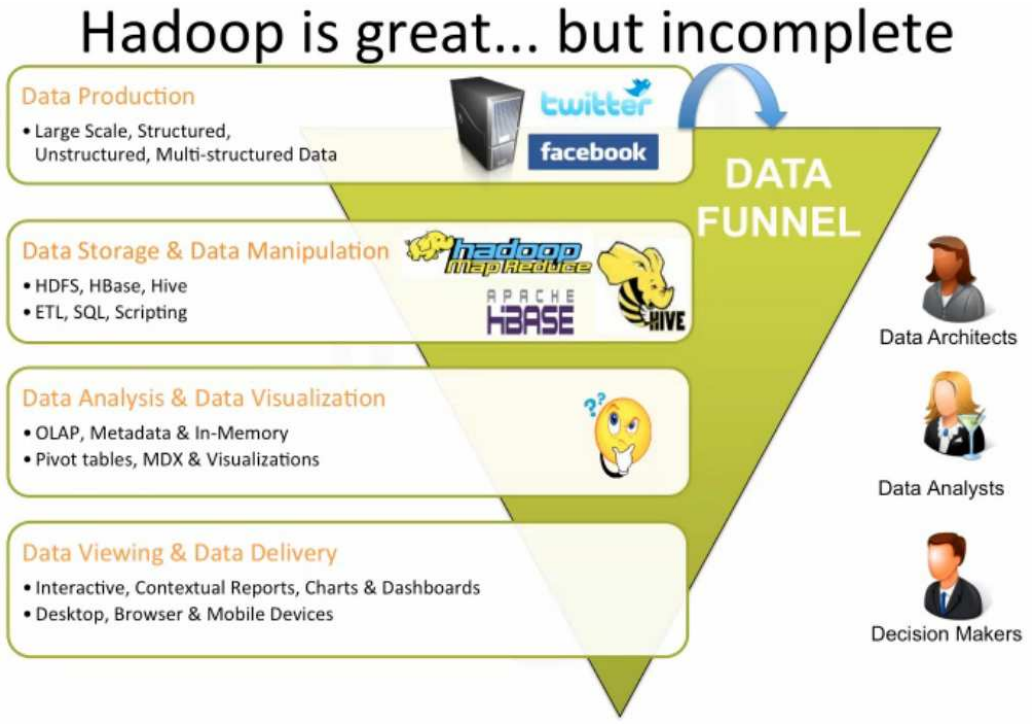
Requirement	Data Warehouse	Hadoop
Low latency, interactive reports, and OLAP	●	
ANSI 2003 SQL compliance is required	●	
Preprocessing or exploration of raw unstructured data		●
Online archives alternative to tape		●
High-quality cleansed and consistent data	●	
100s to 1000s of concurrent users	●	●*
Discover unknown relationships in the data	●	●
Parallel complex process logic		●
CPU intense analysis	●	●
System, users, and data governance	●	
Many flexible programming languages running in parallel		●
Unrestricted, ungoverned sand box explorations		●
Analysis of provisional data		●
Extensive security and regulatory compliance	●	
Real time data loading and 1 second tactical queries	●	●*

*HBase

4. Reporting & Analysis στο Hadoop

Σε αυτή την ενότητα θα εξηγήσουμε την ανάγκη υποβολής εκθέσεων (reporting) και ανάλυσης στο Hadoop και θα περιγράψουμε κάποιες αρχιτεκτονικές προσέγγισης για την αντιμετώπιση διαφόρων περιπτώσεων χρήσης. Η πλατφόρμα Hadoop είναι ένα ισχυρό εργαλείο, αλλά από μόνο του δεν είναι αρκετό για την παροχή γρήγορης και ουσιαστικής γνώσης σε μεγάλα δεδομένα που είναι αποθηκευμένα στο Hadoop [20].

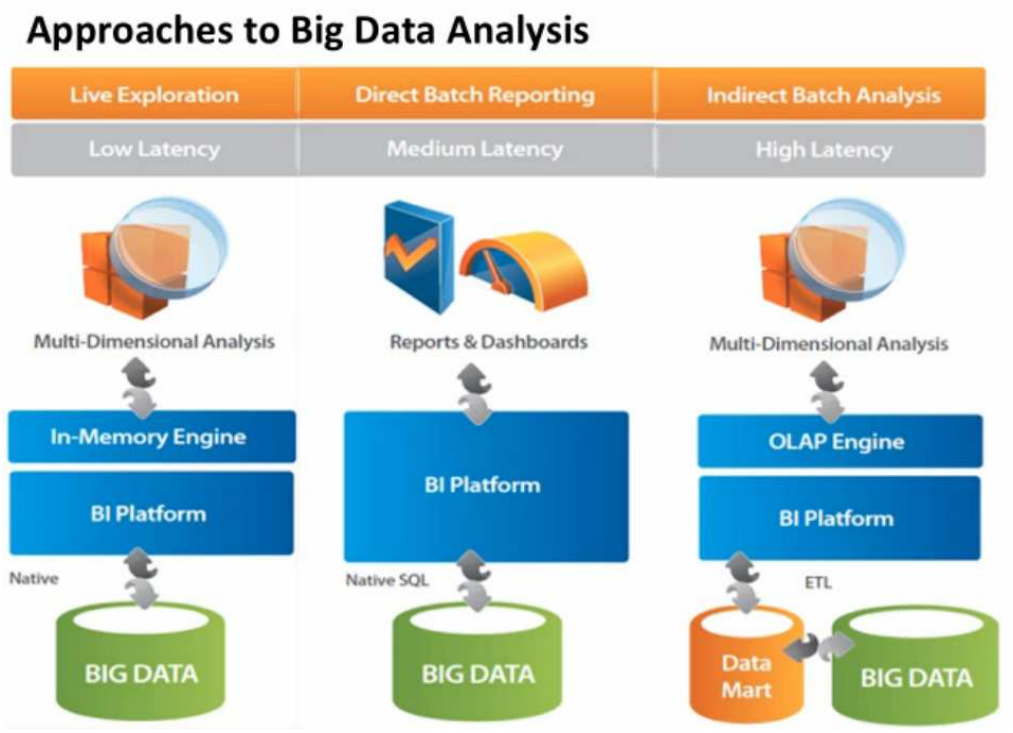
Αρχικά ας θυμηθούμε ότι τα τρία βασικά χαρακτηριστικά των μεγάλων δεδομένων ο όγκος, η ταχύτητα και η ποικιλία. Στο Hadoop αντιμετωπίζονται όλες αυτές οι απαιτήσεις, παρέχοντας ένα κλιμακωμένο οριζοντίως σύστημα για την αντιμετώπιση του όγκου των δεδομένων, όπου χειρίζεται το μεγάλο ρυθμό εισερχόμενων δεδομένων από πολύ μεγάλα συστήματα και υποστηρίζει σύνθετες εργασίες για την επεξεργασία οποιασδήποτε ποικιλίας αδόμητων δεδομένων. Όμως, παρόλη αυτή την ικανότητα, μεγάλη πρόκληση παραμένει η ανάλυση αυτών των δεδομένων και η εξόρυξη πληροφορίας από αυτά. Η πραγματική αξία των μεγάλων δεδομένων έγκειται στην παραγωγή γνώσης σημαντικής για τη λήψη αποφάσεων. Έτσι λοιπόν, με την υποβολή εκθέσεων και αναλύσεων στο Hadoop μπορούμε να κάνουμε τα μεγάλα δεδομένα μικρά, για την διευκόλυνση της διαδικασίας εξόρυξης πληροφορίας, μειώνοντας έτσι το χρόνο και ξεκλειδώνοντας την επιχειρηματική τους αξία. Οι Business Intelligence (BI) σουίτες σήμερα πλέον, μπορούν να ενσωματώσουν μια μεγάλη ποικιλία δεδομένων από εξωτερικές πηγές για τη δημιουργία εκθέσεων και αναλύσεων για μια επιχείρηση. Στην Εικόνα 4.1 βλέπουμε μία χοάνη παραδοσιακών δεδομένων που προσαρμόστηκε στις νεότερες τεχνολογίες που χρησιμοποιούνται στον κόσμο των μεγάλων δεδομένων.



Εικόνα 4.1 Τα στάδια λήψης και επεξεργασίας των Big Data.

Ξεκινάμε με τη φάση της παραγωγής δεδομένων. Τα περισσότερα μεγάλα δεδομένα παράγονται από servers σε μορφή logs, αρχεία συναλλαγών, web εφαρμογές όπως 2.0 και 3.0, δεδομένα αισθητήρων και από επιχειρησιακές εφαρμογές όπως CRMs, ERP, συστήματα χρέωσης και ούτω καθεξής. Τα δεδομένα που προκύπτουν στις επιχειρήσεις σήμερα, είναι δομημένα, ημιδομημένα και αδόμητα και τα οποία αυξάνονται με αστρονομική ταχύτητα. Η επόμενη φάση είναι η αποθήκευση και ο χειρισμός αυτών των δεδομένων. Οι χρήστες σε αυτό το επίπεδο είναι αρχιτέκτονες δεδομένων, οι οποίοι σχεδιάζουν ολόκληρα επιχειρησιακά συστήματα. Οι τεχνολογικές τους επιλογές για τα δεδομένα είναι το κατανεμημένο σύστημα αρχείων του Hadoop και το MapReduce, όπου η HBase χρησιμοποιείται για την αποθήκευση και το Hive ως του Hadoop το Data Warehouse. Λίγο πιο κάτω στη χοάνη δεδομένων είναι ο αναλυτής δεδομένων που βλέπει τα δεδομένα. Η τυπική εργασία του αναλυτή στο επίπεδο αυτό περιλαμβάνει τη δημιουργία κύβων με γεγονότα (facts) και διαστάσεις (dimensions) για την εκτέλεση on-line αναλυτικής επεξεργασίας (OLAP) και την οπτική αναπαράσταση των δεδομένων που βοηθά στην επισήμανση νέων τάσεων και προτύπων. Στη τελική στρώση της χοάνης δεδομένων βρίσκονται οι ιθύνοντες (decision makers) που αναζητούν διαδραστικές, εξειδικευμένες εκθέσεις, γραφήματα και πίνακες εργαλείων με βάση τα οποία να πάρουν σημαντικές στρατηγικές και επιχειρησιακές αποφάσεις.

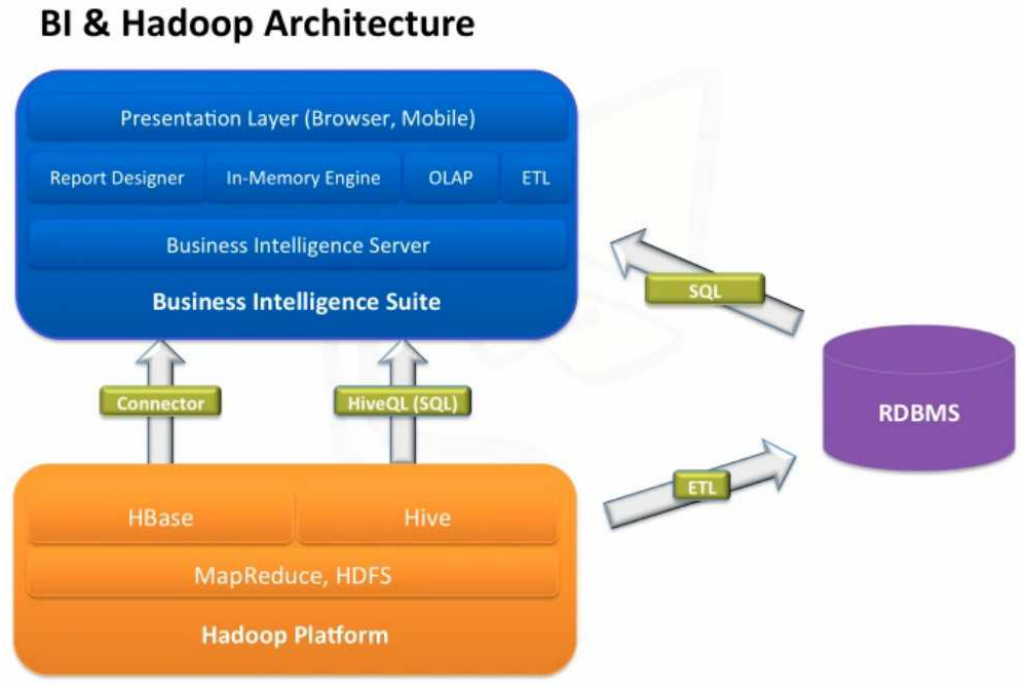
Στη συνέχεια, θα περιγράψουμε διάφορες προσεγγίσεις της ανάλυσης μεγάλων δεδομένων με τρεις περιπτώσεις χρήσης. Στην Εικόνα 4.2 βλέπουμε ότι στη πρώτη προσέγγιση έχουμε τη **Live Exploration**. Ένα παράδειγμα αυτής της προσέγγισης είναι μία οικονομική εταιρεία που προσπαθεί να αντισταθμίσει τις επενδύσεις της και να αναζητήσει νέες τάσεις σε πραγματικό χρόνο για τη λήψη αποφάσεων στην αγορά για οικονομικές συναλλαγές. Αυτή η περίπτωση λοιπόν έχει τη λιγότερη καθυστέρηση και μπορεί να υποστηριχθεί από μία διεπαφή χρήστη (UI) που ελέγχεται από ένα μηχανισμό μνήμης (In-Memory engine) σχεδιασμένο για τη μείωση των καθυστερήσεων. Η δεύτερη προσέγγιση είναι η **Direct Batch Reporting**. Αυτή η περίπτωση χρήσης περιγράφει τις περιπτώσεις όπου τα στελέχη και οι επιχειρησιακοί διευθυντές αναζητούν συνοψισμένες pre-built περιοδικές εκθέσεις για το περιεχόμενο των μεγάλων δεδομένων. Οι εκθέσεις που δημιουργούνται εδώ είναι μέσης καθυστέρησης και απευθύνεται σε ομάδα χρηστών που μπορεί να θέλουν να πάρουν εκθέσεις σχετικά για παράδειγμα: πιο είναι το πιο δημοφιλές περιεχόμενο ιστοσελίδων κατά τη διάρκεια των τελευταίων ημερών. Τέλος, οι **Indirect Batch Analysis** οι οποίες είναι για την ανάλυση ιστορικών τάσεων, όπως όταν ένας πελάτης αγοράζει με βάση τη περιοχή του και τα δημογραφικά του στοιχεία, ή μπορεί να είναι τεράστια περιβαλλοντολογικά σύνολα δεδομένων για τον καθορισμό της τοποθέτησης των ανεμογεννητριών εγκαταστάσεων για τη πράσινη ενέργεια. Αυτές οι αναλύσεις βασίζονται ως επί το πλείστον σε “τεμαχισμό” (slicing) και σε “κύβους” (dicing) ερωτήσεις των μεγάλων δεδομένων. Οι επιχειρησιακοί χρήστες εδώ θα ήταν οι αναλυτές δεδομένων και οι επιχειρησιακοί διευθυντές.



Εικόνα 4.2 Τρεις περιπτώσεις χρήσης για την ανάλυση των Big Data.

Έχοντας περιγράψει τις διάφορες περιπτώσεις χρήσης ανάλυσης των μεγάλων δεδομένων, θα δούμε τις διάφορες τεχνολογίες που μπορούν να ενωθούν για την εφαρμογή των επιμέρους προσεγγίσεων. Στη πρώτη προσέγγιση μια στοίβα αυτής της εφαρμογής μπορεί να περιλαμβάνει Native Connectors για την άντληση δεδομένων από HBase / Hadoop κατανεμημένο σύστημα αρχείων και μια διεπαφή χρήστη που ελέγχεται από ένα μηχανισμό μνήμης για ταχύτερη πρόσβαση στη γνώση. Στη δεύτερη ο χρήστης βλέπει συνοπτικά, προκατασκευασμένες περιοδικές εκθέσεις. Η τυπική στοίβα εφαρμογής εδώ περιλαμβάνει Hive Connectors μέσω των οποίων ένα εργαλείο σχεδιασμού εκθέσεων μπορεί να αξιοποιήσει τα δεδομένα και να αναλάβει το σχεδιασμό και τη δημιουργία χρονοδιαγράμματος εκθέσεων. Τέλος, στη τελευταία περίπτωση, όπου η προσδοκία εδώ είναι η ανάλυση ιστορικών τάσεων. Τα δεδομένα εδώ φτάνουν έως εδώ χρησιμοποιώντας μία τεχνολογία Extract-Transform-Load (ETL) σε μία σχεσιακή βάση για ανάλυση ή ένα μηχανισμό OLAP που χρησιμοποιείται για την απόκτηση γνώσης από ιστορικές τάσεις.

Τις τρεις προσεγγίσεις διακρίναμε κυρίως με βάση την καθυστέρηση, όπου στη πρώτη ο χρόνος είναι συνήθως μερικών δευτερολέπτων, στη δεύτερη ο μέσος χρόνος καθυστέρησης είναι λίγων λεπτών και στη τελευταία ο χρόνος τρέχει με υψηλή καθυστέρηση μερικών ωρών. Στην Εικόνα 4.3 βλέπουμε ένα αρχιτεκτονικό διάγραμμα που αντιπροσωπεύει τις τρεις περιπτώσεις.

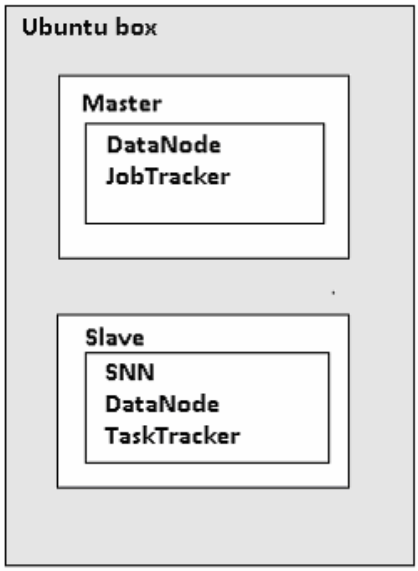


Εικόνα 4.3 Αρχιτεκτονικό διάγραμμα για την ανάλυση των Big Data.

Εδώ παρατηρούμε ότι μία Business Intelligence σουίτα χρησιμοποιεί ETL, HiveQL (SQL), καθώς επίσης και ένα HBase Connector. Έτσι με ETL τα δεδομένα πηγαίνουν σε μια σχεσιακή βάση και ενώ η Hive είναι μία SQL-τύπου γλώσσα, δεν υπάρχει γλώσσα ερωτημάτων για την HBase. Τα ερωτήματα στην HBase γίνονται μέσω ενός Java API άμεσα ή μέσω του Thrift ή REST server interfaces.

ΠΑΡΑΡΤΗΜΑ Α – Ψευδο-κατανεμημένη Hadoop συστοιχία

Θα διαμορφώσουμε ένα ψευδο-κατανεμημένη Hadoop συστοιχία με την χρήση μίας εικονικής μηχανής που θα δημιουργήσουμε στο VMware Player, στην οποία θα τρέξουμε ένα λειτουργικό σύστημα Linux [10].



Αρχικά θα εγκαταστήσουμε το λογισμικό VMware Player for Windows 32bit & 64bit.
https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_player/5_0

Ως “guest” λειτουργικό σύστημα θα εγκαταστήσουμε το Ubuntu 12.10 (Quantal Quetzal) Desktop 32bit.
http://releases.ubuntu.com/quantal/

Το Hadoop είναι γραμμένο σε JAVA, επομένως θα πρέπει να κατεβάσουμε από την Oracle την πλατφόρμα Java 7 SE και να την εγκαταστήσουμε στο Ubuntu box (guest OS) που θα έχουμε δημιουργήσει.
http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

Το Hadoop χρησιμοποιεί SSH πρόσβαση για την απομακρυσμένη διαχείριση των κόμβων στη συστοιχία. Στη περίπτωση αυτή θα χρειαστεί και εδώ για να έχουμε πρόσβαση στο `localhost` για τον χρήστη `hduser` που θα δημιουργήσουμε.

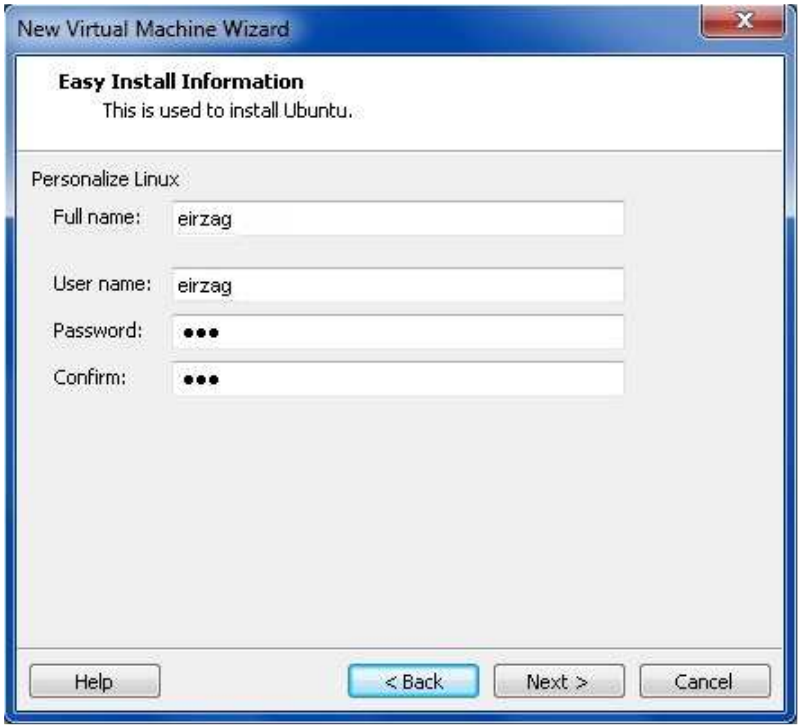
Στη συνέχεια θα κατεβάσουμε μία σταθερή έκδοση της πλατφόρμας Hadoop.
http://apache.cc.uoc.gr/hadoop/common/stable/hadoop-1.1.2.tar.gz

Διαμόρφωση του περιβάλλοντος στο VMware Player

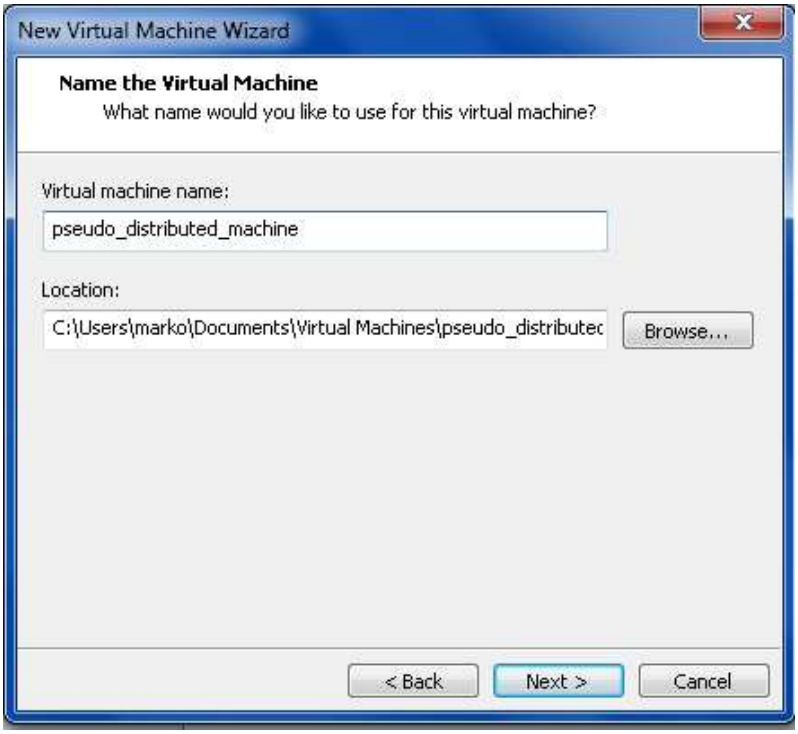
Αρχικά θα δημιουργήσουμε μία νέα εικονική μηχανή – Create a New Virtual Machine.

Επιλέγουμε για εγκατάσταση το Ubuntu 12.10 που έχουμε κατεβάσει στον υπολογιστή μας – Next >.

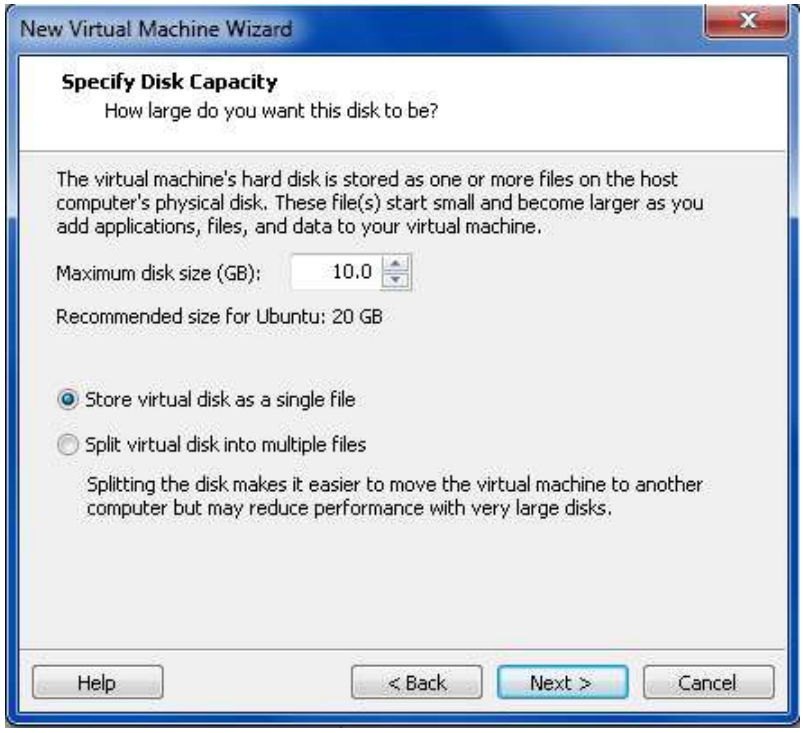
Δίνουμε τα στοιχεία που μας ζητά το guest λειτουργικό σύστημα – Next >.



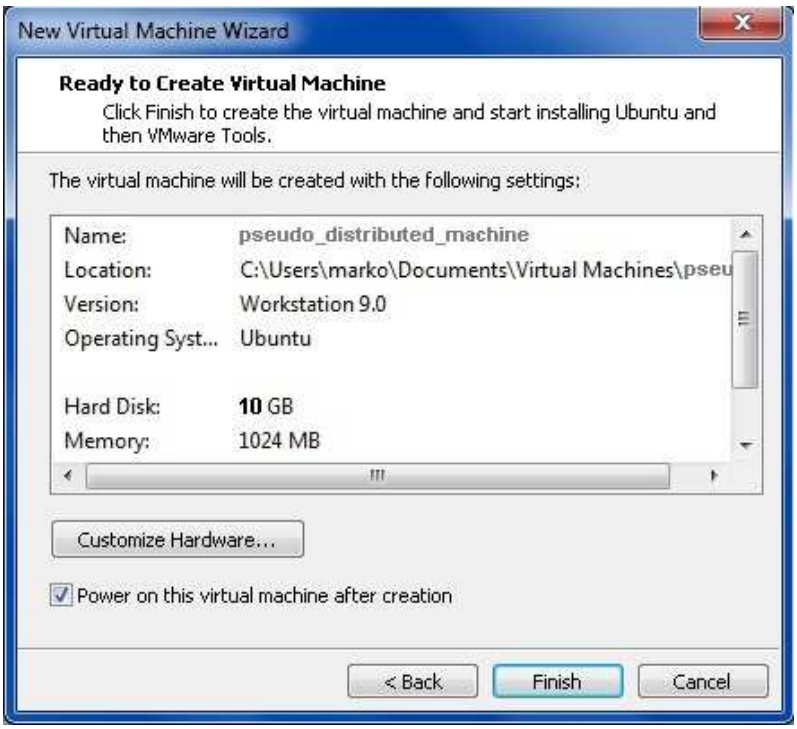
Ορίζουμε το όνομα της εικονικής μηχανής που θα δημιουργήσουμε – Next >.

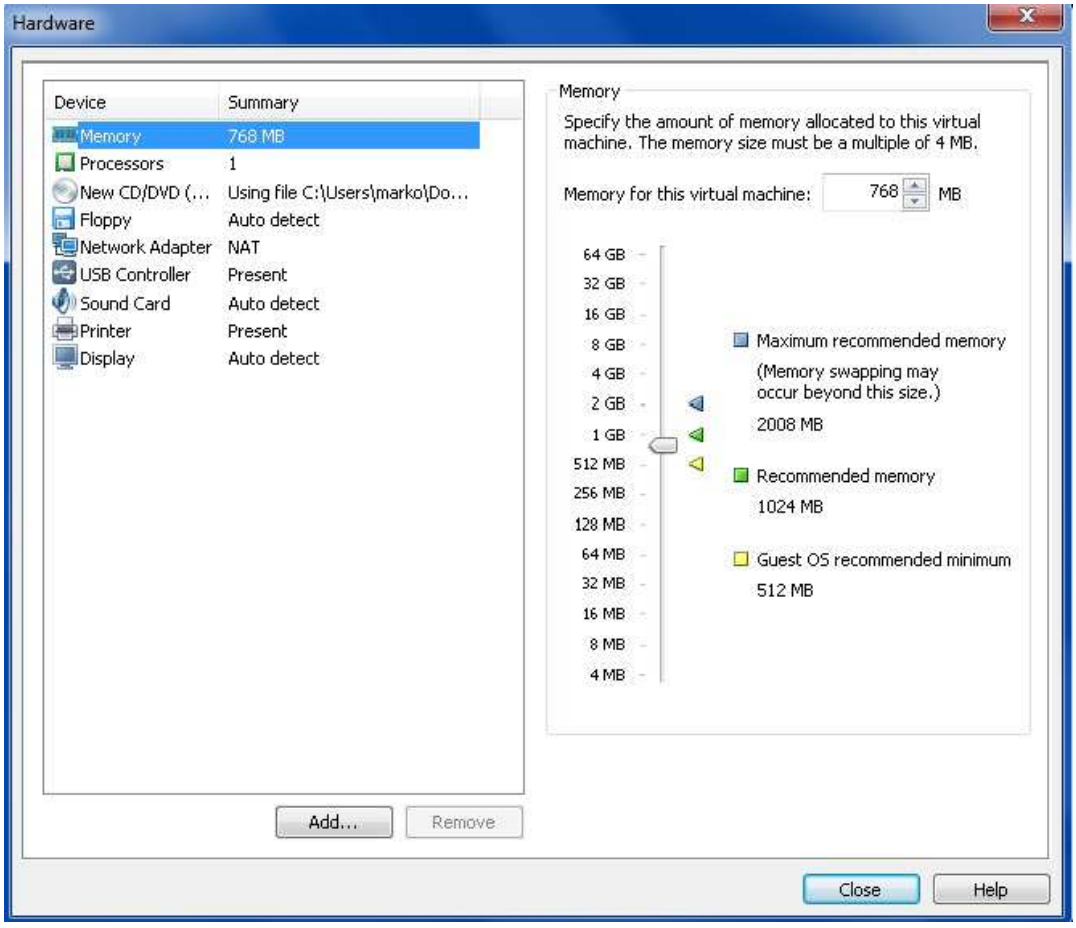


Δηλώνουμε το μέγεθος του σκληρού δίσκου που θέλουμε να χρησιμοποιηθεί και επιπλέον τσεκάρουμε την επιλογή δημιουργίας εικονικού δίσκου σε ένα μόνο αρχείο – Next >.



Βλέπουμε τις ρυθμίσεις της εικονικής μηχανής, όπου μας δίνεται η δυνατότητα να διαχειριστούμε το υλικό της με την επιλογή του κουμπιού Customize Hardware…





Επιλέγοντας Finish αρχίζει η εγκατάσταση του λειτουργικού Linux.

Αλλαγή του `hostname` στο μηχάνημα

Ανοίγοντας το πρόγραμμα Terminal στο Ubuntu θα αλλάξουμε το όνομα hostname του Ubuntu box σε masterslave για λόγους κατανόησης του συστήματος που θέλουμε να χρησιμοποιήσουμε.

```
eirzag@ubuntu:~$ sudo nano /etc/hostname
```

(αλλάζουμε το ubuntu σε masterslave και αποθηκεύουμε - η ενημέρωση γίνεται με επανεκκίνηση του συστήματος)

```
eirzag@masterslave:~$ sudo nano /etc/hosts
```

(όπου 127.0.1.1 βάζουμε masterslave αντί για ubuntu)

Δημιουργία χρήστη που ανήκει σε μία ομάδα του Hadoop συστήματος

Η δημιουργία χρήστη που ανήκει σε μία συγκεκριμένη ομάδα ονόματι Hadoop βοηθά να διαχωριστεί η εγκατάσταση Hadoop από άλλες εφαρμογές λογισμικού και άλλους λογαριασμούς χρήστη που τρέχουν στο ίδιο μηχάνημα (για λόγους ασφάλειας, τα δικαιωμάτων, δημιουργίας αντιγράφων ασφαλείας, κ.λ.π.).

```
eirzag@masterslave:~$ sudo addgroup hadoop
eirzag@masterslave:~$ sudo adduser --ingroup hadoop hduser
```

Εγκατάσταση της JAVA

```
eirzag@masterslave:~/Downloads$ tar -xvf jdk-7u21-linux-i586.tar.gz

eirzag@masterslave:~/Downloads$ sudo mkdir -p /usr/lib/jvm

eirzag@masterslave:~/Downloads$ sudo mv jdk1.7.0_21/ /usr/lib/jvm/

eirzag@masterslave:~/Downloads$
sudo update-alternatives --install "/usr/bin/java" "java"
"/usr/lib/jvm/jdk1.7.0_21/bin/java" 1

eirzag@masterslave:~/Downloads$
sudo update-alternatives --install "/usr/bin/javac" "javac"
"/usr/lib/jvm/jdk1.7.0_21/bin/javac" 1

eirzag@masterslave:~/Downloads$
sudo update-alternatives --install "/usr/bin/javaws" "javaws"
"/usr/lib/jvm/jdk1.7.0_21/bin/javaws" 1

eirzag@masterslave:~/Downloads$ sudo update-alternatives --config java

eirzag@masterslave:~$ jps
```

```
ask your admin to install openjdk-7-jdk
```

```
eirzag@masterslave:~$ sudo apt-get install openjdk-7-jdk

eirzag@masterslave:~/Downloads$ java -version
```

```
java version "1.7.0_21"
OpenJDK Runtime Environment (IcedTea 2.3.9) (7u21-2.3.9-0ubuntu0.12.10.1)
OpenJDK Client VM (build 23.7-b01, mixed mode, sharing)
```

Και επιπλέον, πρέπει να ελένξουμε αν στον Mozilla Firefox είναι εγκατεστημένη η JAVA βάζοντας στη γραμμή διεύθυνσης το εξής: about:plugins. Αν όχι, τότε δημιουργούμε symlink στο μέρος που μας ενδιαφέρει:

```
eirzag@masterslave:/usr/lib/firefox/plugins$
sudo ln -s /usr/lib/jvm/jdk1.7.0_21/jre/lib/i386/libnpjp2.so
```

Ενημέρωση του `$HOME/.bashrc` του hduser

```
eirzag@masterslave:~/$ su hduser
hduser @masterslave:~/$ nano .bashrc
```

Στο τέλος του αρχείου προσθέτουμε:

```
#Set JAVA_HOME
export JAVA_HOME=/usr/lib/jvm/jdk1.7.0_21/

#Set Hadoop-related enviroment variables
export HADOOP_PREFIX=/usr/local/hadoop
export HADOOP_HOME=/usr/local/hadoop

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

Εγκατάσταση SSH

```
eirzag@masterslave:~$ sudo apt-get install ssh
```

Δημιουργία SSH κλειδιού για το χρήστη hduser και αντιγραφή του στο αρχείο authorized_keys, διότι δεν θέλουμε να εισάγουμε τον κωδικό κάθε φορά που το Hadoop αλληλεπιδρά με τους κόμβους της συστοιχίας.

```
eirzag@masterslave:~$ su hduser

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

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

hduser@masterslave:~$ ssh localhost
```

Εγκατάσταση του Hadoop

```
eirzag@masterslave:~/Downloads$ tar -xvf hadoop-1.1.2.tar.gz
```

```
eirzag@masterslave:~/Downloads$ sudo mv hadoop-1.1.2 /usr/local

eirzag@masterslave:~/Downloads$ cd /usr/local

eirzag@masterslave:/usr/local$ sudo chown -R hduser:hadoop
/usr/local/hadoop-1.1.2

eirzag@masterslave:/usr/local$ sudo ln -s /usr/local/hadoop-1.1.2
/usr/local/Hadoop

eirzag@masterslave:/usr/local$ cd /

eirzag@masterslave:/$ sudo mkdir -p /app/hadoop/tmp

eirzag@masterslave:/$ sudo chown -R hduser:hadoop /app

eirzag@masterslave:/$ sudo chmod -R 750 /app
```

Ενημέρωση του `hadoop-env.sh` του `hduser`

```
eirzag@masterslave:/user/local/hadoop$ su hduser

hduser@masterslave:/usr/local/hadoop$ cd conf

hduser@masterslave:/usr/local/hadoop/conf$ nano hadoop-env.sh
```

Κάνουμε τις εξής αλλαγές:

```
#JAVA_HOME and replace whole line with below to configure JAVA
HOME path for #hadoop usage.
export JAVA_HOME=/usr/lib/jvm/jdk1.7.0_21/

#Extra Java runtime options. Empty by default
export HADOOP_OPTS=-Djava.net.preferIPv4Stack=true
```

Με τις παρακάτω εντολές βλέπουμε τις host μηχανές που είναι προρυθμισμένα για `master` και `slave` αντίστοιχα στη συστοιχία:

```
hduser@masterslave:/usr/local/hadoop/conf$ cat masters
```

```
localhost
```

```
hduser@masterslave:/usr/local/hadoop/conf$ cat slaves
```

```
localhost
```

Αλλαγές που πρέπει να γίνουν στα conf/*-site.xml αρχεία

Στον κατάλογο conf/ θα ρυθμίσουμε βασικά αρχεία που αφορούν την αποθήκευση αρχείων δεδομένων του Hadoop, τις θύρες δικτύου που ακούει κ.λ.π.

core-site.xml

```
hduser@masterslave:/usr/local/hadoop/conf$ nano core-site.xml
```

```
<?xml version="1.0"?>

...........
<configuration>
<property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
    <description>
           The name of the default file system.
           A URI whose    scheme and    authority determine the
           FileSystem implementation.
    </description>
</property>
<property>
    <name>hadoop.tmp.dir</name>
    <value>/app/hadoop/tmp</value>
    <description>
          Base folder for other temporary directories.
    </description>
</property>
</configuration>
...........
```

mapred-site.xml

```
hduser@masterslave:/usr/local/hadoop/conf$ nano mapred-site.xml
```

```
<?xml version="1.0"?>

...........
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>localhost:9001</value>
    <description>
           The host and port that the MapReduce job tracker runs
```

```
         at.
     </description>
</property>
</configuration>
..........
```

hdfs-site.xml

```
hduser@masterslave:/usr/local/hadoop/conf$ nano hdfs-site.xml
```

```
<?xml version="1.0"?>

...........
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
    <description>Default data blocks replication</description>
  </property>

<property>
    <name>dfs.name.dir</name>
    <value>${hadoop.tmp.dir}/dfs/name</value>
    <description>
          Determines where on the local filesystem the DFS
          NameNodeshould store the name table(fsimage). If this is
          a comma-delimited list of directories then the name
          table is replicated in all of the directories, for
          redundancy.
     </description>
</property>
<property>
    <name>dfs.data.dir</name>
    <value>${hadoop.tmp.dir}/dfs/data</value>
    <description>
          Determines where on the local filesystem an DFS DataNode
          should store its blocks. If this is a comma-delimited
          list of directories, then data will be stored in all
          named directories, typically on different devices.
          Directories that do not exist are ignored.
    </description>
</property>
</configuration>
..........
```

Εκκίνηση της Hadoop συστοιχίας

Το πρώτο βήμα για την εκκίνηση του Hadoop είναι η διαμόρφωση (format) του συστήματος αρχείων Hadoop που υλοποιείται πάνω από το τοπικό σύστημα αρχείων της συστοιχίας μας (περιλαμβάνεται μόνο στο τοπικό υπολογιστή μας). Θα πρέπει να το κάνουμε αυτό την πρώτη φορά που θα δημιουργήσουμε μία συστοιχία Hadoop.

```
hduser@masterslave:~$ cd /usr/local/hadoop/bin

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

Τρέχουμε τον μόνο κόμβο της συστοιχίας για την εκκίνηση των Namenode, Datanode, SecondaryNamenode, TaskTracker, JobTracker δαιμόνων.

```
hduser@masterslave:/usr/local/hadoop/bin$ start-all.sh
```

```
starting namenode, logging to /usr/local/hadoop-
1.1.2/libexec/../logs/hadoop-hduser-namenode-masterslave.out
localhost: starting datanode, logging to /usr/local/hadoop-
1.1.2/libexec/../logs/hadoop-hduser-datanode-masterslave.out
localhost: starting secondarynamenode, logging to
/usr/local/hadoop-1.1.2/libexec/../logs/hadoop-hduser-
secondarynamenode-masterslave.out
starting jobtracker, logging to /usr/local/hadoop-
1.1.2/libexec/../logs/hadoop-hduser-jobtracker-masterslave.out
localhost: starting tasktracker, logging to /usr/local/hadoop-
1.1.2/libexec/../logs/hadoop-hduser-tasktracker-masterslave.out
```

Οι διεργασίες JAVA που τρέχουν στο σύστημά μας είναι:

```
hduser@masterslave:/usr/local/hadoop/bin$ jps
```

```
6215 NameNode
6977 Jps
6905 TaskTracker
6615 SecondaryNameNode
6699 JobTracker
6414 DataNode
```

Εκτέλεση μίας MapReduce job

Θα χρησιμοποιήσουμε το παράδειγμα WordCount job (ΠΑΡΑΡΤΗΜΑ Δ) που περιλαμβάνεται με την εγκατάσταση του Hadoop, το οποίο διαβάζει αρχεία κειμένου και μετρά το πόσες φορές εμφανίζεται κάθε λέξη. Έτσι λοιπόν, θα δημιουργήσουμε ένα κατάλογο `tmp` που θα περιέχει ένα αρχείο κειμένου με όνομα `inputtextfile.txt` και με το εξής περιεχόμενο “`This is an example to test the wordcount job.`”

Αντιγραφή του αρχείου από το τοπικό σύστημα αρχείων στο HDFS του Hadoop.

```
hduser@masterslave:/tmp$ hadoop dfs -copyFromLocal
/tmp/inputtextfile.txt /user/hduser/inputtextfile.txt

hduser@masterslave:/tmp$ hadoop dfs -ls /user/hduser
```

```
-rw-r--r--   1 hduser supergroup         51 2013-05-22 10:03
/user/hduser/ inputtextfile.txt
```

```
hduser@masterslave:/usr/local/hadoop$ hadoop jar hadoop-examples-
1.1.2.jar wordcount /user/hduser/inputtextfile.txt /user/hduser/
inputtextfile.txt-output
```

```
13/05/22 10:09:05 INFO input.FileInputFormat: Total input paths to
process : 1
13/05/22 10:09:05 INFO util.NativeCodeLoader: Loaded the native-
hadoop library
13/05/22 10:09:05 WARN snappy.LoadSnappy: Snappy native library
not loaded
13/05/22 10:09:07 INFO mapred.JobClient: Running job:
job_201305220948_0001
13/05/22 10:09:08 INFO mapred.JobClient:  map 0% reduce 0%
13/05/22 10:09:26 INFO mapred.JobClient:  map 100% reduce 0%
13/05/22 10:09:40 INFO mapred.JobClient:  map 100% reduce 100%
13/05/22 10:09:44 INFO mapred.JobClient: Job complete:
job_201305220948_0001
..................
```

```
hduser@masterslave:/usr/local/hadoop$
 hadoop dfs -ls /user/hduser/inputtextfile.txt-output
```

```
-rw-r--r--   1 hduser supergroup          0 2013-05-22 10:09
/user/hduser/inputtextfile.txt-output/_SUCCESS

drwxr-xr-x   - hduser supergroup          0 2013-05-22 10:09
/user/hduser/inputtextfile.txt-output/_logs

-rw-r--r--   1 hduser supergroup         71 2013-05-22 10:09
/user/hduser/inputtextfile.txt-output/part-r-00000
```

```
hduser@masterslave:/usr/local/hadoop$
hadoop dfs -cat /user/hduser/inputtextfile.txt-output/part-r-00000
```

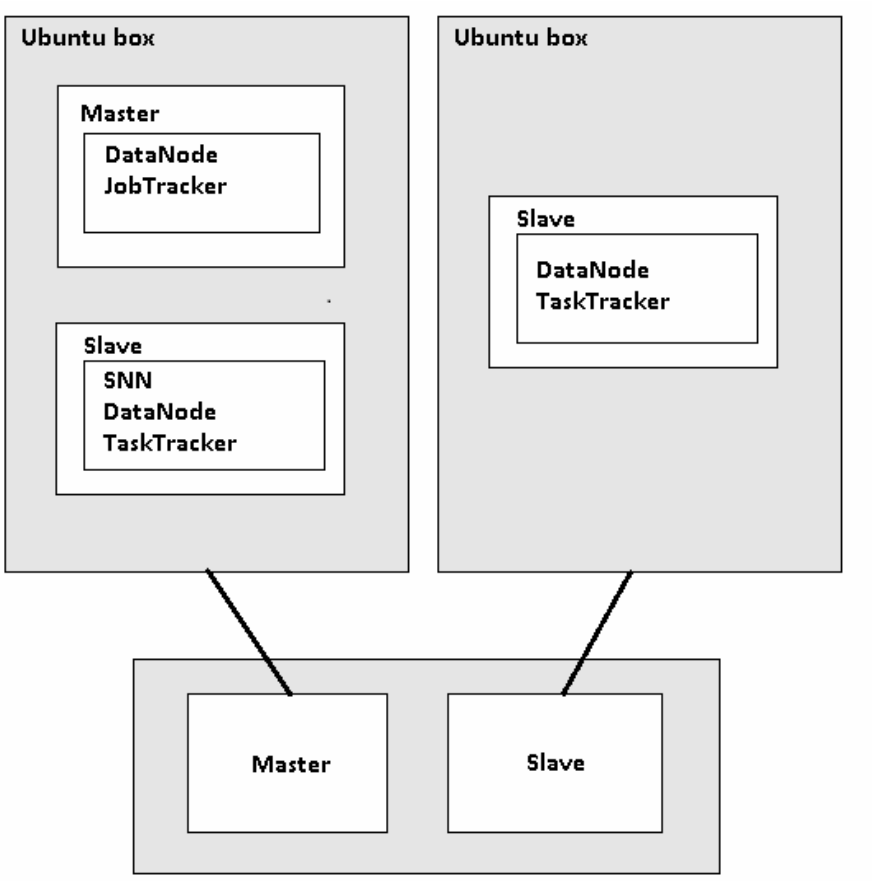
```
This 1
an   1
example   1
```

```
file 1
is   1
job. 1
test 1
the  1
to   1
wordcount 1
```

```
hduser@masterslave:/usr/local/hadoop$ stop-all.sh
```

ΠΑΡΑΡΤΗΜΑ Β – Πλήρως-κατανεμημένη Hadoop συστοιχία

Θα διαμορφώσουμε μία πλήρως-κατανεμημένη Hadoop συστηχία υπολογιστών με την χρήση δύο εικονικών μηχανών που θα δημιουργήσουμε στο VMware Player, στις οποίες θα τρέξουμε ένα λειτουργικό σύστημα Linux. Ακολουθούμε την ίδια διαδικασία με το ΠΑΡΑΡΤΗΜΑ Α, με κάποιες επιπλέον ρυθμίσεις [11].



Στις δύο εικονικές μηχανές, την μία θα την ονομάσουμε master και την άλλη slave.

Διαμόρφωση των IP των μηχανημάτων

Έχοντας πρώτα αλλάξει το `hostname` των μηχανημάτων master και slave, ενημερώνουμε το αρχείο `hosts` και στα δύο μηχανήματα:

master

```
eirzag@master:~$ ifconfig
```

```
eth0      Link encap:Ethernet  HWaddr 00:0c:29:50:0f:b9
          inet    addr:192.168.138.133        Bcast:192.168.138.255
Mask:255.255.255.0
```

slave

```
eirzag@:slave~$ ifconfig
```

```
eirzag@slave:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:b3:bd:5b
          inet    addr:192.168.138.132        Bcast:192.168.138.255
Mask:255.255.255.0
```

Στο /etc/hosts γράφουμε τα εξής:

```
127.0.0.1       localhost
192.168.138.133  master
192.168.138.132  slave

#127.0.1.1       ubuntu
# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Διαμόρφωση των εικονικών μηχανών για τους χρήστες hduser αντίστοιχα

- Δημιουργούμε τον χρήστη hduser που θα ανήκει στη Hadoop ομάδα.
- Εγκατάσταση της JAVA στα μηχάνηματα αλλά και στον Mozilla Firefox (plugin).
- Ενημέρωση του $HOME/.bashrc του hduser
- Εγκατάσταση SSH

SSH επικοινωνία master – slave για τους χρήστες hduser χωρίς προτροπή για password

Αντιγράφουμε το δημόσιο κλειδί του master μηχανήματος από το αρχείο ~/.ssh/authorized_keys και το προσθέτουμε στο αντίστοιχο αρχείο του μηχανήματος slave.

```
hduser@master:~$ ssh slave
```

Έχοντας κάνει εγκατάσταση του Hadoop για τους χρήστες hduser σε κάθε μηχάνημα, ενημερώνουμε και το αρχείο hadoop-env.sh αντίστοιχα.

Στο master και για το χρήστη hduser κάνουμε τις εξής αλλαγές:

```
hduser@master:/usr/local/hadoop/conf$ nano masters
```

```
master
```

```
hduser@master:/usr/local/hadoop/conf$ nano slaves
```

```
master
slave
```

Αλλαγές που πρέπει να γίνουν στα conf/*-site.xml αρχεία και στα δύο μηχανήματα των hduser

Στον κατάλογο conf/ θα ρυθμίσουμε βασικά αρχεία που αφορούν την αποθήκευση αρχείων δεδομένων του Hadoop, τις θύρες δικτύου που ακούει κ.λ.π.

core-site.xml

```
hduser@master:/usr/local/hadoop/conf$ nano core-site.xml
```

```
<?xml version="1.0"?>

...........
<property>
  <name>fs.default.name</name>
  <value>hdfs://master:54310</value>
  <description>
          The name of the default file system.  A URI whose
          scheme and authority determine the FileSystem
          implementation. The uri's scheme determines the config
          property (fs.SCHEME.impl) naming the FileSystem
          implementation class. The uri's authority is used to
          determine the host, port, etc. for a filesystem.
  </description>
</property>
...........
```

mapred-site.xml

```
hduser@master:/usr/local/hadoop/conf$ nano mapred-site.xml
```

```
<?xml version="1.0"?>

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

```
      and reduce task.
  </description>
</property>
.......
```

hdfs-site.xml

```
hduser@master:/usr/local/hadoop/conf$ nano hdfs-site.xml
```

```
<?xml version="1.0"?>

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

Εκκίνηση της Hadoop συστοιχίας στον master

Το πρώτο βήμα για την εκκίνηση του Hadoop είναι η διαμόρφωση (format) του συστήματος αρχείων Hadoop που υλοποιείται πάνω από το τοπικό σύστημα αρχείων της συστοιχίας μας (περιλαμβάνεται μόνο στο τοπικό υπολογιστή μας). Θα πρέπει να το κάνουμε αυτό για την πρώτη φορά που θα δημιουργήσουμε μία συστοιχία Hadoop.

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

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

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

```
starting namenode, logging to /usr/local/hadoop-
1.1.2/libexec/../logs/hadoop-hduser-namenode-master.out
slave: starting datanode, logging to /usr/local/hadoop-
1.1.2/libexec/../logs/hadoop-hduser-datanode-slave.out
master: starting datanode, logging to /usr/local/hadoop-
1.1.2/libexec/../logs/hadoop-hduser-datanode-master.out
master: starting secondarynamenode, logging to /usr/local/hadoop-
1.1.2/libexec/../logs/hadoop-hduser-secondarynamenode-master.out
```

```
starting jobtracker, logging to /usr/local/hadoop-
1.1.2/libexec/../logs/hadoop-hduser-jobtracker-master.out
slave: starting tasktracker, logging to /usr/local/hadoop-
1.1.2/libexec/../logs/hadoop-hduser-tasktracker-slave.out
master: starting tasktracker, logging to /usr/local/hadoop-
1.1.2/libexec/../logs/hadoop-hduser-tasktracker-master.out
```

Οι διεργασίες JAVA που τρέχουν στον master είναι:

```
hduser@master:/usr/local/hadoop/bin$ jps
```

```
6215 NameNode
6977 Jps
6905 TaskTracker
6615 SecondaryNameNode
6699 JobTracker
6414 DataNode
```

Οι διεργασίες JAVA που τρέχουν στον slave είναι:

```
hduser@slave:/usr/local/hadoop/bin$ jps
```

```
7965 TaskTracker
7790 DataNode
8020 Jps
```

Εκτέλεση μίας MapReduce job

Θα χρησιμοποιήσουμε δεδομένα από ένα ebook " The Adventures of Sherlock Holmes", που θα βρούμε στον ιστότοπο http://www.gutenberg.org/ebooks/1661.txt.utf-8 [14].

```
hduser@master:/tmp$ mkdir sherlock_holmes

hduser@master:/tmp$ cd sherlock_holmes

hduser@master:/tmp/sherlock_holmes$
wget http://www.gutenberg.org/ebooks/1661.txt.utf-8

hduser@master:/tmp/sherlock_holmes$ ls
```

```
1661.txt.utf-8
```

```
hduser@master:/tmp/sherlock_holmes$
hadoop dfs -copyFromLocal /tmp/sherlock_holmes/1661.txt.utf-8
/user/hduser/1661.txt.utf-8

hduser@master:/tmp/sherlock_holmes$ hadoop dfs -ls
```

```
-rw-r--r--   2 hduser supergroup     594933 2013-05-29 09:47
/user/hduser/1661.txt.utf-8
```

```
hduser@master:/usr/local/hadoop$
hadoop jar hadoop-examples-1.1.2.jar wordcount
/user/hduser/1661.txt.utf-8 /user/hduser/1661.txt.utf-8-output
```

```
13/05/29 09:52:52 INFO input.FileInputFormat: Total input paths to
process : 1
13/05/29 09:52:52 INFO util.NativeCodeLoader: Loaded the native-
hadoop library
13/05/29 09:52:52 WARN snappy.LoadSnappy: Snappy native library
not loaded
13/05/29 09:52:55 INFO mapred.JobClient: Running job:
job_201305290923_0001
13/05/29 09:52:56 INFO mapred.JobClient:  map 0% reduce 0%
13/05/29 09:54:09 INFO mapred.JobClient:  map 100% reduce 0%
13/05/29 09:54:40 INFO mapred.JobClient:  map 100% reduce 100%
13/05/29 09:54:45 INFO mapred.JobClient: Job complete:
job_201305290923_0001
..................
```

```
hduser@master:/usr/local/hadoop$
hadoop dfs -ls /user/hduser/1661.txt.utf-8-output
```

```
-rw-r--r--   2 hduser supergroup          0 2013-05-29 09:54
/user/hduser/1661.txt.utf-8-output/_SUCCESS
drwxr-xr-x   - hduser supergroup          0 2013-05-29 09:52
/user/hduser/1661.txt.utf-8-output/_logs
-rw-r--r--   2 hduser supergroup     158031 2013-05-29 09:54
/user/hduser/1661.txt.utf-8-output/part-r-00000
```

```
hduser@master:/usr/local/hadoop$
hadoop dfs -cat /user/hduser/1661.txt.utf-8-output/part-r-00000

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

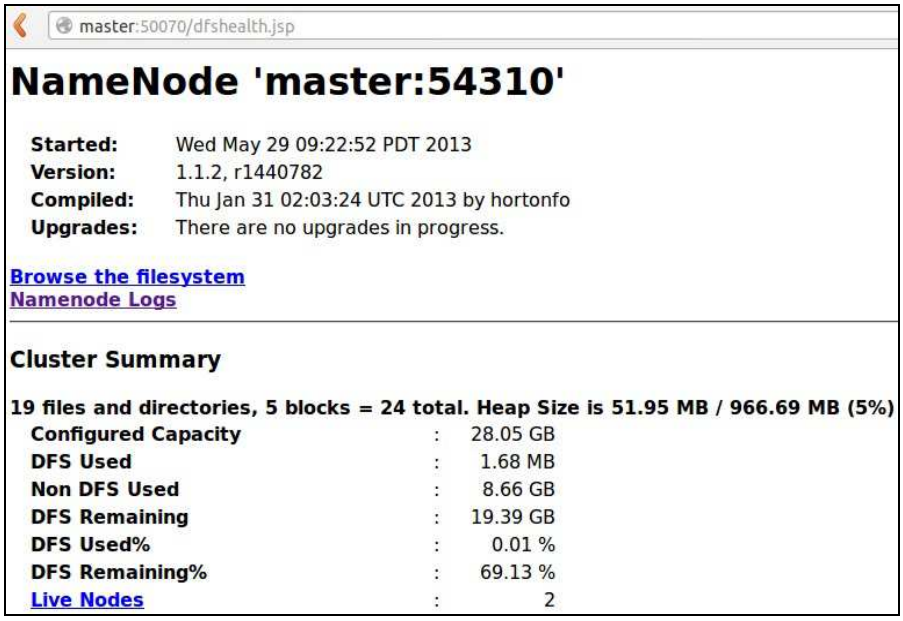
Hadoop Web Interfaces

Οι παρακάτω προκαθορισμένες ιστοσελίδες μας πληροφορούν τι συμβαίνει στη συστοιχία μας:

- http://localhost:50070/ – web UI για τον NameNode daemon
- http://localhost:50030/ – web UI για τον JobTracker daemon
- http://localhost:50060/ – web UI για τον TaskTracker daemon

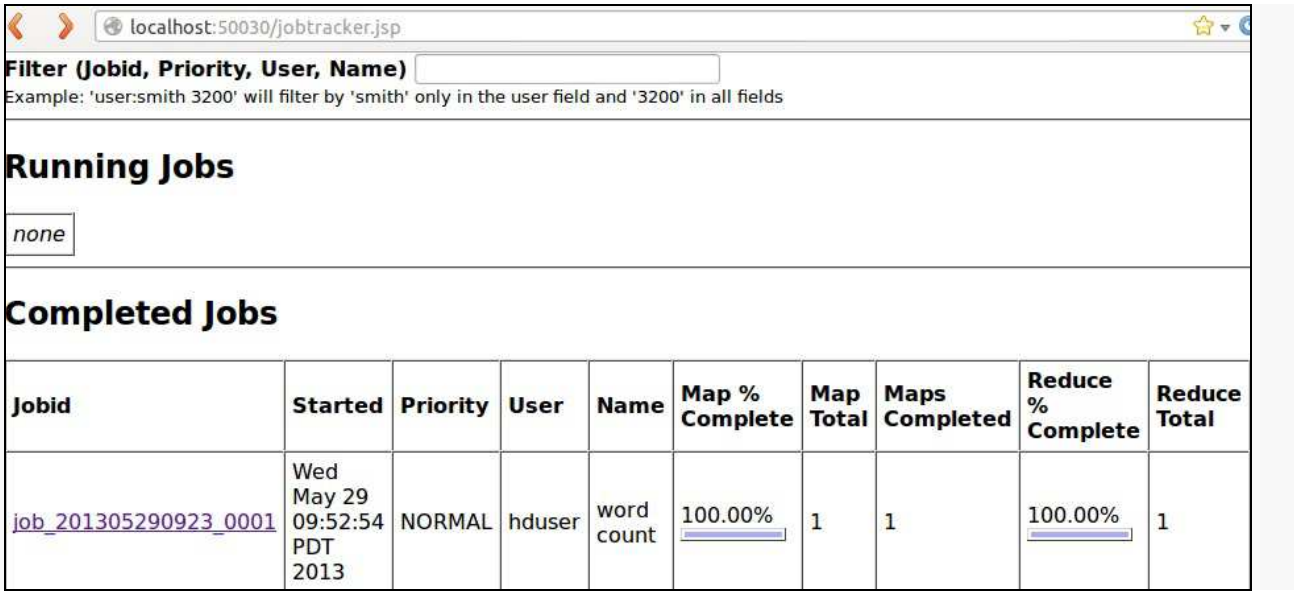
NameNode Web Interface (HDFS layer)

Στο NameNode web UI μας δείχνει συνοπτικά χαρακτηριστικά της συστοιχίας όπως: τη χωρητικότητα, την κατάσταση των κόμβων κ.α. Και επιπρόσθετα μας επιτρέπει να δούμε το σύστημα αρχείων, το HDFS namespace και το περιεχόμενο των αρχείων στον Web Browser. Επίσης μας δίνει πρόσβαση σε log αρχεία που βρίσκονται τοπικά στο μηχάνημα Hadoop.



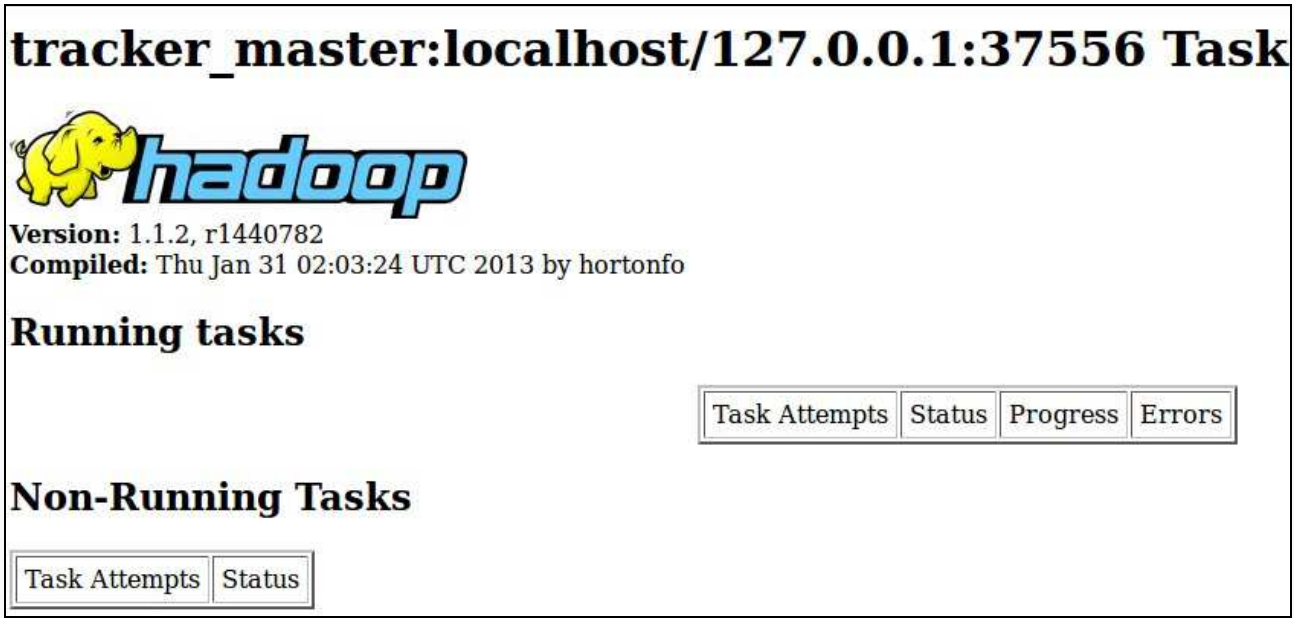
JobTracker Web Interface (MapReduce layer)

Στο JobTracker web UI πληροφορούμαστε στατιστικά για τις job όπως για το ποιες εκτελούνται, ολοκληρώθηκαν ή απέτυχαν και το ιστορικό τους σε log αρχείο.



TaskTracker Web Interface (MapReduce layer)

Στο TaskTracker web UI βλέπουμε εκτελέσιμες και μη εκτελέσιμες εργασίες (tasks).



ΠΑΡΑΡΤΗΜΑ Γ – MapReduce με Python

Το πρόγραμμα που θα δημιουργήσουμε θα μιμείται το ήδη υπάρχον WordCount.java (ΠΑΡΑΡΤΗΜΑ Δ). Εδώ θα χρησιμοποιήσουμε το Hadoop Streaming API [12].

Θα ξεκινήσουμε δημιουργώντας τον κώδικα mapper.py ο οποίος θα διαβάζει δεδομένα από το `stdin`, θα διαχωρίζει τις λέξεις και θα εξάγει μία λίστα από γραμμές, όπου σε κάθε γραμμή θα αντιστοιχεί μία λέξη και ο αριθμός 1.

```
hduser@master:/home/hduser$ nano mapper.py

hduser@master:~$ chmod u+x mapper.py
```

mapper.py

```
#!/usr/bin/env python

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # split the line into words
    words = line.split()

    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

Στη συνέχεια θα δημιουργήσουμε τον κώδικα reducer.py που θα διαβάζει τα αποτελέσματα του mapper.py από το `stdin` (η μορφή της εξόδου του mapper.py και η αναμενόμενη μορφή εισόδου στο reducer.py πρέπει να ταιριάζουν) και θα αθροίζει τις εμφανίσεις της κάθε λέξης σε ένα συνολικό μετρητή. Τα αποτελέσματα εξόδου θα στέλνονται στο `stdout`.

```
hduser@masterslave:/home/hduser$ nano reducer.py
hduser@masterslave:~$ chmod u+x reducer.py
```

reducer.py

```
#!/usr/bin/env python

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

Αρχικά θα εξετάσουμε τα mapper.py και reducer.py τοπικά πριν τα χρησιμοποιήσουμε σε μία MapReduce job:

```
hduser@master:~$ echo “foo foo quux labs foo bar quux”
| ./mapper.py
```

```
“foo 1
foo  1
quux 1
labs 1
foo  1
bar  1
quux”     1
```

```
hduser@master:~$ echo “foo foo quux labs foo bar quux”
| ./mapper.py  | sort -k1,1 | /home/hduser/reducer.py
```

```
bar  1
foo  2
“foo 1
labs 1
quux 1
quux”     1
```

Θα χρησιμοποιήσουμε δεδομένα από ένα ebook “Ulysses by James Joyse” που θα βρούμε στον ιστότοπο http://www.gutenberg.org/ebooks/4300.txt.utf-8 [15].

```
hduser@master:/tmp/gutenberg$
wget http://www.gutenberg.org/ebooks/4300.txt.utf-8

hduser@master:/tmp$ ls -l

hduser@masterslave:/usr/local/hadoop$ start-all.sh
```

Πριν τρέξουμε μία MapReduce job, πρέπει πρώτα να αντιγράψουμε το αρχείο από το τοπικό σύστημα αρχείων στο HDFS.

```
hduser@masterslave:/usr/local/hadoop$ hadoop dfs -copyFromLocal
/tmp/gutenberg/4300.txt.utf-8
/user/hduser/gutenberg/4300.txt.utf-8

hduser@master:/tmp/gutenberg$ hadoop dfs -ls
/user/hduser/gutenberg

hduser@master:/usr/local/hadoop$
bin/hadoop jar contrib/streaming/hadoop-*streaming*.jar
-file /home/hduser/mapper.py    -mapper /home/hduser/mapper.py
-file /home/hduser/reducer.py   -reducer /home/hduser/reducer.py
-input /user/hduser/gutenberg/* -output /user/hduser/gutenberg-
output
```

```
http://master:50030/jobdetails.jsp?jobid=job_201305290923_0002
13/05/29 11:57:36 INFO streaming.StreamJob:  map 0%  reduce 0%
13/05/29 11:59:58 INFO streaming.StreamJob:  map 8%  reduce 0%
```

```
13/05/29 12:00:28 INFO streaming.StreamJob:  map 50%  reduce 0%
13/05/29 12:01:12 INFO streaming.StreamJob:  map 100%  reduce 0%
13/05/29 12:01:21 INFO streaming.StreamJob:  map 100%  reduce 33%
13/05/29 12:01:25 INFO streaming.StreamJob:  map 100%  reduce 73%
13/05/29 12:01:28 INFO streaming.StreamJob:  map 100%  reduce 100%
13/05/29 12:01:36 INFO streaming.StreamJob: Job complete:
job_201305290923_0002
13/05/29 12:01:36 INFO streaming.StreamJob: Output:
/user/hduser/gutenberg-output
```

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

ΠΑΡΑΡΤΗΜΑ Δ – WordCount job σε Java

```
package org.apache.hadoop.examples;

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

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException
{
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
```

```
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf,
args).getRemainingArgs();
    if (otherArgs.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

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

[2] Chuck Lam, “Hadoop in Action”, Manning, 2011

[3] Robert D.Schneider, “Hadoop for DUMMIES”, John Willey & Sons Canada, Ltd, 2012

[4] Chris Eaton, Dirk Deroos, Tom Deutsch, George Lapis, “Understanding Big Data – Analytics for Enterprise Class Hadoop and Streaming Data”, 2012

[5] Paul Zikopoulos, Dirk Deroos, Krishnan Parasuraman, Thomaw Deutshc, David Corrigan, James Giles, “Harness the power of Big Data – The IBM Big Data Platform”, 2013

[6] Neeraj Sharma, Abhishek Iyer, Rajib Bhattacharya, Niraj Modi, Wagner Crivelini, “Getting Started with Data Warehousing” (DRAFT), February 2012

[7] HDFS Architecture Guide, http://hadoop.apache.org/docs/r1.0.4/hdfs_design.html

[8] Hadoop MapReduce, http://hadoop.apache.org/mapreduce

[9] Yahoo! Hadoop Tutorial, http://developer.yahoo.com/hadoop/tutorial/

[10] Running Hadoop on Ubuntu Linux (Single-Node Cluster) http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/

[11] Running Hadoop on Ubuntu Linux (Multi-Node Cluster) http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/

[12] Writing an Hadoop MapReduce Program in Python http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/

[13] Free eBooks by Project Gutenberg, http://www.gutenberg.org/

[14] Project Gutenberg's The Adventures of Sherlock Holmes, by Arthur Conan Doyle, http://www.gutenberg.org/cache/epub/1661/pg1661.txt

[15] The Project Gutenberg EBook of Ulysses, by James Joyce, http://www.gutenberg.org/cache/epub/4300/pg4300.txt

[16] Hadoop and the Data Warehouse, When to use Which, https://www.clouder.com/content/dam/clouder/Resources/PDF/Hadoop_and_the_Data_Warehouse_Whitepaper.pdf

[17] Oracle: Big Data for the Enterprise, http://www.oracle.com/us/products/database/big-data-for-enterprise-519135.pdf

[18] Learn Hadoop & Big Data with Free Courses, http://www.bigdatauniversity.com/

[19] OLAP vs OLTP, http://datawarehouse4u.info/OLTP-vs-OLAP.html

[20] Big Data University, http://bigdatauniversity.com/courses/ - Hadoop Reporting and Analysis

[21] Apache Flume, http://flume.apache.org/