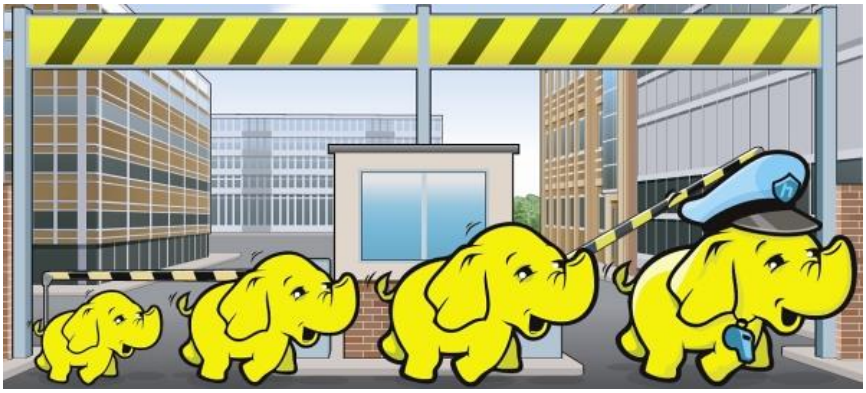
Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Σχεδίαση και ανάπτυξη περιβάλλοντος αποτίμησης μεθόδων εξισορρόπησης φορτίου σε συνθήκες υψηλής ζήτησης πληροφορίας Χωρικού Περιεχομένου**
	Design and development methods of a Load Balancing environment for spatial Big Data content
Ονοματεπώνυμο Φοιτητή	**Δρυμούσης Εμμανουήλ - Χαράλαμπος**
Πατρώνυμο	**Ιωάννης**
Αριθμός Μητρώου	**ΜΠΣΠ 11061**
Επιβλέπων	**Χρ. Δουληγέρης, Καθηγητής**



Πειραιάς, Δεκέμβριος 2014

Επιβλέπων: Χρήστος Δουληγέρης
Καθηγητής Πανεπιστημίου Πειραιά

Τριμελής Εξεταστική Επιτροπή

Χρήστος Δουληγέρης Καθηγητής	Ιωάννης Παπαδάκης Επίκουρος Καθηγητής	Παναγιώτης Κοτζανικολάου Λέκτορας

Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής»

Τα πνευματικά δικαιώματα χρήσης του μη πρωτότυπου υλικού της εργασίας ανήκουν στον φοιτητή και τον επιβλέποντα εις ολόκληρον, δηλαδή εκάτερος μπορεί να κάνει χρήση αυτών χωρίς τη συναίνεση του άλλου.

Τα πνευματικά δικαιώματα χρήσης του πρωτότυπου μέρους μεταπτυχιακής διατριβής ανήκουν στον φοιτητή και τον επιβλέποντα από κοινού, δηλαδή δεν μπορεί ο ένας από τους δύο να κάνει χρήση αυτού χωρίς τη συναίνεση του άλλου. Κατ' εξαίρεση, επιτρέπεται η δημοσίευση του πρωτότυπου μέρους της εργασίας σε επιστημονικό περιοδικό ή πρακτικά συνεδρίου από τον ένα εκ των δύο, με την προϋπόθεση να αναφέρονται τα ονόματα και των δύο ως συν-συγγραφέων. Στην περίπτωση αυτή, προηγείται γραπτή ενημέρωση του μη συμμετέχοντα στη συγγραφή του επιστημονικού άρθρου.

Δεν επιτρέπεται η κατά οποιοδήποτε τρόπο δημοσιοποίηση υλικού το οποίο έχει δηλωθεί εγγράφως ως απόρρητο.

Οι υπογράφοντες

ΕΥΧΑΡΙΣΤΙΕΣ

Με την ολοκλήρωση της μεταπτυχιακής διατριβής, θέλω να ευχαριστήσω όλους όσους βρέθηκαν δίπλα μου, ο καθένας με τον δικό του τρόπο, αρωγοί στην ολοκλήρωση αυτής της προσπάθειας.

Η παρούσα μεταπτυχιακή εργασία πραγματοποιήθηκε υπό την επίβλεψη του κ. Χρήστου Δουληγέρη, Καθηγητή του Μεταπτυχιακού Προγράμματος «Προηγμένα Πληροφοριακά Συστήματα» του τμήματος Πληροφορικής του Πανεπιστημίου Πειραιά. Τον ευχαριστώ για την εμπιστοσύνη που μου έδειξε, καθώς και για την καθοδήγηση και την υποστήριξή του από την αρχή μέχρι και την ολοκλήρωση αυτής της εργασίας.

Πολύτιμες υπήρξαν οι συμβουλές και οι υποδείξεις του κ. Δημητρίου Καλλέργη, υποψήφιου διδάκτορα. Η ανεκτικότητα, υποστήριξη και βοήθεια ήταν καθοριστικές καθ' όλη την διάρκεια. Τον ευχαριστώ ιδιαίτερα γιατί συνέβαλε τα μέγιστα στην ολοκλήρωση αυτής της εργασίας.

Τέλος, ένα πολύ μεγάλο ευχαριστώ οφείλω στους γονείς μου, Βάσω και Γιάννη, στον αδερφό μου Δημήτρη για την συμπαράσταση τους, την υλική και ηθική στήριξη των επιλογών μου καθώς και στην Ευαγγελία που με την συνεισφορά της όλο αυτό το διάστημα με καθησύχαζε. Ένα μεγάλο ευχαριστώ στους φίλους και συνεργάτες μου που με στήριξαν όλο αυτό το διάστημα.

Αφιερώνω αυτή την μεταπτυχιακή διπλωματική εργασία στον πατέρα μου

Πίνακας Περιεχομένων

Ευρετήριο εικόνων

ABSTRACT

This work addresses systems, load balancing behaviour using Hadoop and the MapReduce mechanism. The implemented system relies on Object Oriented Programming techniques focusing on the Java language. Moreover, PostgreSQL used for data manipulation. The current development succeeds in Big Data utilization with xml formatting. Parallel procedures allow (a) big data manipulation, (b) storing of these data in relational database systems and (c) on demand data representation. Furthermore, this work aims to highlight the large volume of data manipulation capabilities using Hadoop.

ΠΡΟΛΟΓΟΣ

Η παρούσα μεταπτυχιακή διατριβή πραγματεύεται την δημιουργία ενός συστήματος εξισορρόπησης φορτίου, μέσω της χρήσης Hadoop, με την ενσωμάτωση MapReduce. Η υλοποίηση του συστήματος πραγματοποιήθηκε με την χρήση προγραμμάτων Java. Αντίστοιχα, η αποθήκευση των δεδομένων έγινε με PostgreSQL. Αξιοποιώντας το συγκεκριμένο σύστημα, επετεύχθη η διαχείριση μεγάλων αρχείων xml (BIG DATA). Οι παράλληλες διεργασίες επέτρεψαν α) την διαχείριση μεγάλου όγκου δεδομένων, β) την αποθήκευσή τους στην βάση δεδομένων και τέλος, γ) την δημιουργία παρουσίασης αυτών των δεδομένων με τρόπο κατ' επιλογή. Η συγκεκριμένη εργασία αποσκοπεί να αναδείξει την χρησιμότητα του συστήματος Hadoop στην διαχείριση μεγάλου όγκου δεδομένων.

1. Εισαγωγή

Η ραγδαία εξέλιξη της τεχνολογίας καθώς και του διαδικτύου (Internet) ανέδειξε την ανάγκη διαχείρισης, αποθήκευσης και παρουσίασης του τεράστιου όγκου δεδομένων, που – αναπόφευκτα – δημιουργήθηκε. Σε καθημερινή βάση, είμαστε αποδέκτες διαφορετικών τύπων δεδομένων (e-mail, ιστοσελίδες, διαφημίσεις και άλλων), τα οποία συχνά απορρίπτουμε λόγω της επαναληψιμότητάς τους. Μέσα από την επιστημονική κοινότητα, μπορεί κανείς να προβεί σε μια σειρά ενεργειών με στόχο την διαχείριση, ανάλυση και αποθήκευση αυτού του τεράστιου όγκου δεδομένων. Επίσης, ερευνητικές προσεγγίσεις επιχειρούν να δώσουν πληροφορίες σχετικά με τρόπους παρουσίασης αυτών των δεδομένων, ούτως ώστε να α) μεγιστοποιηθεί η ποιότητα της εμφάνισης αυτών των δεδομένων και β) υπάρχει μεγαλύτερη αποδοχή τους από τους αποδέκτες αυτών των δεδομένων.

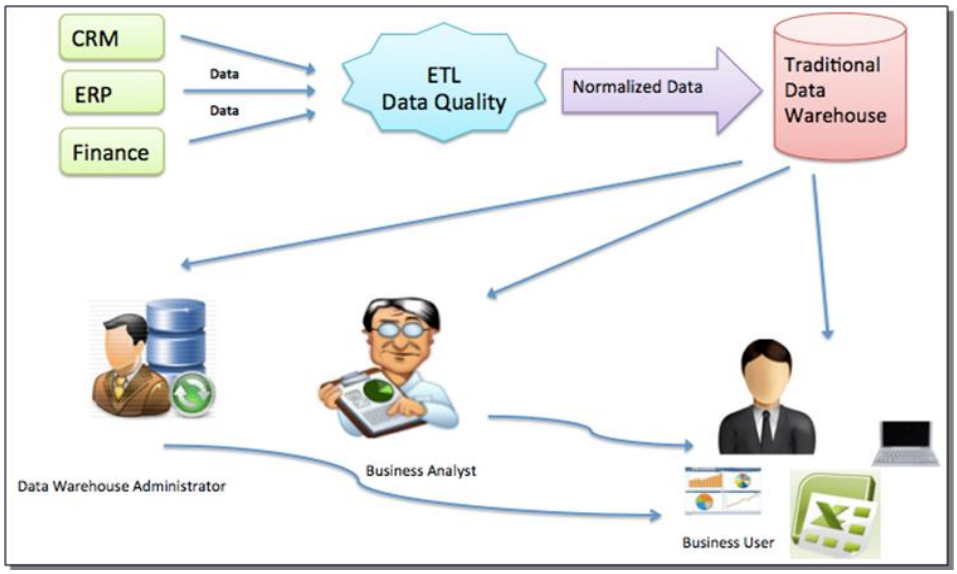
1.1 Σκοπός μεταπτυχιακής διατριβής

Η παρούσα μεταπτυχιακή διατριβή επιχειρεί να αντιμετωπίσει ανάλογο πρόβλημα με την υλοποίηση ενός συστήματος – βασισμένου στο Hadoop και στο MapReduce – το οποίο χρησιμοποιεί παράλληλες διεργασίες και υπολογιστές, με απώτερο στόχο να επεξεργαστεί μεγάλο όγκο δεδομένων, σε μικρότερο χρόνο. Παρουσιάζεται δηλαδή, η μεθοδολογία αντιμετώπισης του προβλήματος. Πιο συγκεκριμένα, χρησιμοποιώντας την γλώσσα προγραμματισμού Java και τις βιβλιοθήκες του Hadoop, υλοποιούμε το MapReduce· δημιουργούμε μια σειρά από διαδοχικά στάδια ενδιάμεσων παρουσιάσεων μέχρι να φτάσουμε στο επιθυμητό στάδιο παρουσίασης των δεδομένων [1].

Στη συνέχεια, παρουσιάζεται μια σύντομη ιστορική αναδρομή για το πώς ήταν τα δεδομένα και ποια ήταν η διαχείρισή τους, πριν το Hadoop.

Οι παραδοσιακές αποθήκες δεδομένων (Data Warehouses) καθώς και ανάλογα εργαλεία διαχείρισης αυτών δεν μπορούν να αναλάβουν την διαδικασία της επεξεργασίας ανάλυσης του μεγάλου όγκου δεδομένων (Big Data), ενώ σε αντίθετη περίπτωση, το κόστος θα ήταν πολύ μεγάλο. Προτεινόμενη λύση είναι μια σειρά από σχεσιακούς πίνακες, όπου θα αποθηκευτούν δεδομένα σε μια μεγάλη αποθήκη δεδομένων. Αυτή όμως, η λύση δεν μπορεί να λειτουργήσει, όταν τα δεδομένα δεν έχουν συγκεκριμένη δομή. Επίσης, στην περίπτωση αύξησης του όγκου των δεδομένων, η αποθήκη δεδομένων πρέπει να μεγαλώσει πολύ περισσότερο για να μπορέσει να φιλοξενήσει τα δεδομένα · και αυτό συνεπάγεται την αύξηση του κόστους. Επίσης, απαιτείται πάρα πολύ ακριβό υλικό (hardware) για να φιλοξενηθεί ένα τέτοιου είδους σύστημα. Παραδοσιακά, τέτοιες λύσεις δίνονται από το Customer Relationship Management (CRM), το Enterprise Resource Planning (ERP) καθώς και από τραπεζικά συστήματα [2].

Τα εργαλεία προσαρμογής των δεδομένων λειτουργούσαν ως εξής: α) ελάμβαναν τα δεδομένα από αυτά τα συστήματα και από τις σχεσιακές βάσεις δεδομένων και β) τα εισήγαγαν σε μια περιοχή, όπου γινόταν η επεξεργασία και η αποθήκευση των δεδομένων σε πίνακες. Μέσα από τους πίνακες, τα δεδομένα θα μπορούσαν να αποθηκευτούν σε καθαρές εταιρικές αποθήκες δεδομένων. Ανάλογο παράδειγμα παρουσιάζεται στην εικόνα 1.1.



Εικόνα 1.1: *Το παραδοσιακό σύστημα διαχείρισης όγκου δεδομένων από την πηγή (CRM, ERP, Finance) στην Βάση Δεδομένων και κατάληξη στους τελικούς χρήστες.*

Η χρήση του διαδικτύου, των φορητών συσκευών και άλλων τεχνολογιών επέφερε μια αλλαγή στην φύση και στον όγκο των δεδομένων. Ο μεγάλος όγκος αυτών των δεδομένων (Big Data) έχει διαφορετικό τύπο και ποιότητα δεδομένων σε σχέση με τα παλιά, εταιρικά δεδομένα που αναφέραμε στο προηγούμενο παράδειγμα. Αυτά τα δεδομένα δεν βασίζονται σε μια κεντρική βάση, δεν έχουν μια συμπαγή δομή και δεν είναι τόσο εύκολα διαχειρίσιμα. Σήμερα, τα δεδομένα έχουν τις εξής ιδιαιτερότητες α) είναι διασκορπισμένα και β) έχουν πάρα πολύ μεγάλο όγκο [3].

Πιο αναλυτικά, τα δεδομένα έχουν τα εξής χαρακτηριστικά:

- **Όγκο**. Τα δεδομένα, τα οποία δημιουργούνται μέσα στις επιχειρήσεις και στο διαδίκτυο (για τις φορητές συσκευές, τα υπολογιστικά συστήματα και τις υπόλοιπες πηγές), αυξάνονται λογαριθμικά κάθε χρόνο.

- **Τύπο**. Η ποικιλία των τύπων περιλαμβάνει μη-δομημένα δεδομένα, τύπου κειμένου. Τα εν λόγω δεδομένα μεταξύ άλλων, προέρχονται από κοινωνικά δίκτυα (social media), από γεωγραφικά δεδομένα (location-based data) και από δεδομένα αρχείων καταγραφής (log-file data).

- **Ταχύτητα**. Η ταχύτητα δημιουργίας νέων δεδομένων, καθώς και η ανάγκη, με την οποία αυτά τα δεδομένα πρέπει να αναλυθούν από ειδικούς ώστε να επιτευχθεί η εμπορική ή και άλλη εκμετάλλευσή τους, μεγαλώνει συνεχώς. Και αυτό συμβαίνει λόγω της ψηφιοποίησης των συναλλαγών, της συνεχούς φορητότητας των υπολογιστικών συσκευών και του ολοένα, αυξανόμενου ρυθμού των χρηστών στο διαδίκτυο.

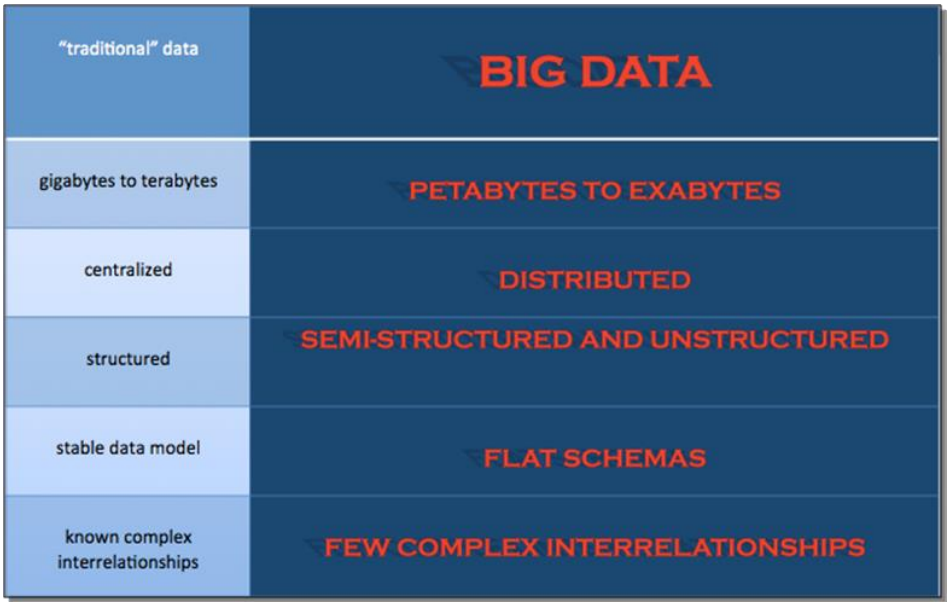
Όσον αφορά τις πηγές από τις οποίες προκύπτει ο μεγάλος όγκος δεδομένων, χρήζει να αναφερθούν τα παρακάτω [4]:

- **Κοινωνικά Δίκτυα**. Υπάρχουν περισσότεροι από 1 δισεκατομμύριο χρήστες στο Facebook, 350 εκατομμύρια χρήστες στο Twitter και πάνω από 250 εκατομμύρια σε blogs. Σε κάθε αλλαγή (Update) αυτών – ακόμη και σε ένα μικρό σχόλιο – δημιουργείται μεγάλος όγκος δεδομένων, τα οποία δύνανται να είναι δομημένα και μη-δομημένα.

▪ **Φορητές Συσκευές**. Περισσότερα από 5 δισεκατομμύρια άτομα χρησιμοποιούν κινητά τηλέφωνα. Σε κάθε κλήση ή σε κάθε γραπτό μήνυμα, δημιουργείται και μια αποθήκευση σε ένα αρχείο καταγραφής. Επίσης, τα smart και τα tablets χρησιμοποιούν social media αλλά και άλλες εφαρμογές, οι οποίες χρησιμοποιούν δεδομένα. Τέλος, συλλέγουν δεδομένα μέσω του GPS.

▪ **Διαδικτυακές Συναλλαγές**. Πραγματοποιούνται καθημερινά, δισεκατομμύρια συναλλαγές, οι οποίες προέρχονται από διαδικτυακές αγορές, από αγορές μετοχών και από άλλες πηγές. Κάθε μια από αυτές δημιουργεί δεδομένα, τα οποία συλλέγονται από επαγγελματίες, τράπεζες, πιστωτικές κάρτες.

▪ **Δικτυακές Συσκευές και Αισθητήρες**. Αυτές οι ηλεκτρονικές συσκευές, οι οποίες συμπεριλαμβάνουν εξυπηρετητές (servers), αλλά και άλλες υπολογιστικές συσκευές δημιουργούν δεδομένα καταγραφής, καθώς αποθηκεύουν κάθε ενέργεια.



Εικόνα 1.2: *Σύγκριση μεταξύ Hadoop και συστημάτων Data Warehouses*

1.2 Δομή πτυχιακής εργασίας

Η εργασία αποτελείται από τα παρακάτω κεφάλαια:

Κεφάλαιο 2ο . Θεωρητικό υπόβαθρο.

Γίνεται ανάλυση όλων των εργαλείων, των προγραμμάτων και των μοντέλων που θα εφαρμοστούν.

Κεφάλαιο 3ο . Σχεδιασμός εργασίας.

Καταγράφονται οι σκέψεις και οι προβληματισμοί κατά την διάρκεια υλοποίησης της εργασίας, καθώς και η μεθοδολογία που ακολουθήθηκε για την επιτυχή ολοκλήρωσή της.

Κεφάλαιο 4ο. Υλοποίηση της εργασίας.

Παρουσιάζεται αναλυτικά, η διαδικασία με την οποία υλοποιήθηκε η εργασία καθώς και τα βήματα χρήσης του συστήματος.

Κεφάλαιο 5ο. Συμπεράσματα.

Παρουσιάζονται τα συμπεράσματα αναφορικά με την παρούσα μεταπτυχιακή διατριβή, με βάση τα αποτελέσματα τα οποία προέκυψαν από προηγούμενο κεφάλαιο.

Κεφάλαιο 6ο. Μελλοντική εργασία.

Παρουσιάζονται προοπτικές βελτίωσης της μεταπτυχιακής διατριβής, μέσω της παραμετροποίησης της.

2. Βιβλιογραφική Ανασκόπηση

Στο κεφάλαιο αυτό, γίνεται μια εκτενής περιγραφή όλων των προγραμμάτων καθώς και των εργαλείων, τα οποία αξιοποιήθηκαν στην παρούσα εργασία. Επιπλέον, αναλύονται τα μοντέλα που χρησιμοποιήθηκαν για την υλοποίηση του έργου, το οποίο πραγματεύεται η εργασία.

2.1 Hadoop

Πρόκειται για μια πλατφόρμα λογισμικού, η οποία μπορεί να χρησιμοποιηθεί για την ανάλυση και την επεξεργασία μεγάλου αριθμού δεδομένων (επιπέδου petabyte). Με το Hadoop τόσο τα δεδομένα όσο και η διαδικασία ανάλυσής τους σε ομάδες υπολογιστών (Clusters) κατανέμονται ώστε η επεξεργασία των δεδομένων να γίνεται παράλληλα, με συνακόλουθη επιτάχυνση των διαδικασιών [5].

Αυτή η πλατφόρμα λογισμικού αξιοποιεί το προγραμματιστικό μοντέλο MapReduce. Η Google ανέπτυξε το εν λόγω μοντέλο, με στόχο την υποστήριξη του συστήματός της στην ανάλυση δεδομένων. Αναλυτικότερα, η τεχνική αυτή επιτρέπει την μεταβίβαση των δεδομένων του προβλήματος στον υπολογιστή master του Cluster· κατόπιν, ο τελευταίος θα διασπάσει το πρόβλημα σε επιμέρους προβλήματα τα οποία θα τα προωθήσει σε καθένα από τους υπόλοιπους υπολογιστές του Cluster. Στην συνέχεια, ο κάθε υπολογιστής του Cluster θα προβεί σε επίλυση του δικού του υπο-προβλήματος και θα επιστρέψει την λύση στον master υπολογιστή. Έτσι, ο master υπολογιστής θα επιλύσει το πρόβλημα που του δόθηκε, μέσα από τον συνδυασμό των λύσεων των υπο-προβλημάτων.

Στην περίπτωση του Hadoop, αυτό βασίζεται στην παραπάνω περιγραφόμενη τεχνική με επιπρόσθετα πλεονεκτήματα. Ενδεικτικά, αναφέρουμε ότι στο ενδεχόμενο ζημίας ενός υπολογιστή του Cluster, αυτό έχει την δυνατότητα

ανάκτησης δεδομένων και μεταβίβασης του υπο-προβλήματος σε άλλον υπολογιστή. Πολλές επιχειρήσεις και οργανισμοί χρησιμοποιούν το Hadoop για ανάγκες έρευνας αλλά και παραγωγής. Μεταξύ αυτών, συγκαταλέγεται η Yahoo και το Facebook. Αξίζει να σημειωθεί ότι ένα – ανοιχτού κώδικα – λογισμικό έχει αναπτύξει το πρόγραμμα Apache Hadoop με στόχο έναν αξιόπιστο, εξελικτικό και κατανεμημένο υπολογισμό.

2.1.1 Εργαλεία & Προγράμματα του Hadoop

Βασικό, χαρακτηριστικό συστατικό του Hadoop είναι το Hadoop Distributed File System (HDFS), το οποίο διαθέτει χαρακτηριστικά ενός κλασσικού συστήματος αρχείων Unix. Με σκοπό την βελτιστοποίηση της απόδοσης του συστήματος, τα πρότυπα του Unix τροποποιήθηκαν για την υλοποίηση του HDFS. Το πρότυπο λειτουργίας του HDFS αξιοποιεί ορισμένα χαρακτηριστικά από τα – ήδη – γνωστά κατανεμημένα συστήματα αρχείων όπως, το GFS, το PVFS και το Lustre. Ο τρόπος που λειτουργεί το HDFS είναι ο εξής: α) αποθηκεύει τα metadata για την διαχείριση των αρχείων και ολόκληρου του συστήματος σε συγκεκριμένο server, που ονομάζεται NameNode και β) διατηρεί τα δεδομένα σε διαφορετικούς servers, που ονομάζονται DataNodes και είναι υπεύθυνοι για την ανάκτηση και την αντιγραφή των block των αρχείων. Υπενθυμίζεται ότι η τεχνική τις αντιγραφής των block σε διαφορετικά DataNodes είναι η αντίστοιχη τεχνική που χρησιμοποιεί και το GFS.

2.1.2 Περιγραφή Λειτουργίας του Hadoop Distributed File System (HDFS)

Στο σημείο αυτό, κρίνεται απαραίτητο να περιγραφεί εκτενώς η αρχιτεκτονική του HDFS. Η περιγραφή αυτή θα συμβάλει α) στην κατανόηση της λειτουργίας του (μεταξύ των άλλων και στην κατανόηση της ευκολίας προσαρμογής του σε διαφορετικές απαιτήσεις) αλλά και γενικότερα, β) στην κατανόηση των δυνατοτήτων καθώς και των περιορισμών του συστήματος,

2.1.3 Hadoop Namenode

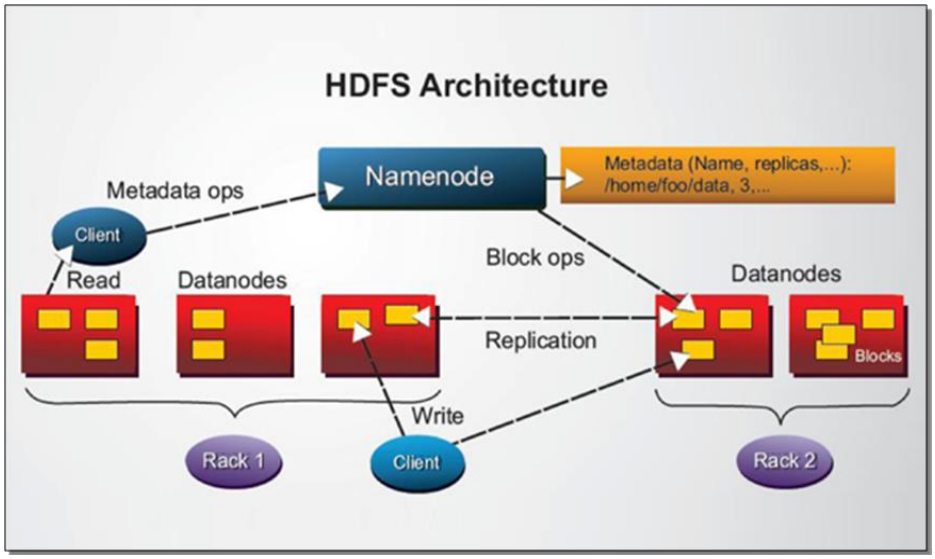
Το Namespace του HDFS ορίζεται από μία ιεραρχία αρχείων και φακέλων, οι οποίοι αποθηκεύονται στο NameNode με την μορφή inodes. Τα inodes αποτελούν δομές δεδομένων - ευρέως χρησιμοποιούμενες σε συστήματα αρχείων Unix - όπου καταγράφονται χαρακτηριστικά όπως, π.χ., τα δικαιώματα των χρηστών, οι ώρες τροποποίησης / πρόσβασης και το μέγεθος [6].

Σε διαφορετικά blocks (τυπικά, μεγέθους 64 Mbytes) κατανέμονται τα περιεχόμενα ενός αρχείου και κάθε block του εκάστοτε αρχείου αντιγράφεται σε διαφορετικά DataNodes. Το μέγεθος του block χαρακτηρίζεται ως μια τροποποιήσιμη παράμετρος, η οποία συμβάλλει σημαντικά στην απόδοση του συστήματος. Αναλυτικότερα, η επιλογή block μεγαλύτερου μεγέθους μειώνει το φορτίο στους DataNodes, λόγω των λιγότερων προσπελάσεων που απαιτούνται για την ανάγνωσή τους. Εντούτοις, όταν δεσμεύονται blocks, τα οποία δεν χρησιμοποιούνται πλήρως, υφίσταται σπατάλη χώρου. Το NameNode διατηρεί τόσο το Namespace Tree όσο και την τοποθεσία των blocks ενός αρχείου στα αντίστοιχα DataNodes. Έχει στην ευθύνη του την εξυπηρέτηση αιτήσεων, τον έλεγχο της λειτουργίας των DataNodes καθώς και την συνοχή του Namespace.

Όταν ένας πελάτης (client) θέλει να διαβάσει ένα αρχείο, επικοινωνεί με το NameNode και κάνει αναζήτηση της τοποθεσίας των blocks του αρχείου. Διαβάζει τα αντίστοιχα blocks από τον εγγύτερο σε αυτόν DataNode, εφόσον ο τελευταίος είναι διαθέσιμος. Άλλως, προχωράει στον επόμενο. Σε περίπτωση που ένας client χρειάζεται να γράψει ένα αρχείο στο Hadoop Distributed File System (HDFS), επικοινωνεί με το NameNode, ο οποίος επιλέγοντας κάποια DataNodes γράφονται τα blocks του αρχείου καθώς και τα αντίγραφα των blocks. Κατόπιν, ο client γράφει με την τεχνική pipelining, απ’ ευθείας στα αντίστοιχα DataNodes. Στην μνήμη διατηρείται ολόκληρο το Namespace. Τα δεδομένα των inodes και η λίστα με τα blocks – που ανήκουν σε κάθε αρχείο – συνιστούν τα μεταδεδομένα του συστήματος ή αλλιώς, η «εικόνα» του συστήματος. Η ενημερωμένη εικόνα του συστήματος αποθηκεύεται στον τοπικό δίσκο του NameNode. Επίσης, στον NameNode διατηρείται ένα «ημερολόγιο» (log), το

οποίο ονομάζουμε journal. Σε αυτό το «ημερολόγιο» γίνεται η αναφορά των τροποποιήσεων που προκύπτουν στην εικόνα του συστήματος. Για να διασφαλιστεί η ανοχή σε σφάλματα δημιουργούνται αντίγραφα των παραπάνω πληροφοριών σε διαφορετικούς κόμβους.
Ο NameNode, πέραν της λειτουργίας του να εξυπηρετεί αιτήσεις από πελάτες (clients) εναλλακτικά, μπορεί να λειτουργήσει είτε ως CheckpointNode, είτε ως BackupNode. Αυτή η επιλογή πραγματοποιείται στην αρχικοποίηση του NameNode.

Όταν ο NameNode λάβει το ρόλο του CheckpointNode, τότε αναλαμβάνει περιοδικά να συνδυάσει την ήδη υπάρχουσα εικόνα του συστήματος με τις αλλαγές, τις καταγεγραμμένες στο αρχείο καταγραφών (log file). Στη συνέχεια, δημιουργείται νέα εικόνα και ένα νέο άδειο αρχείο καταγραφών. Ο CheckpointNode είθισται να βρίσκεται σε διαφορετικό host καθώς έχει τις ίδιες ανάγκες για μνήμη με το NameNode. Ο CheckpointNode αποστέλλει στο NameNode την νέα εικόνα του συστήματος που κατασκεύασε. Η χρήση CheckpointNode βοηθάει στην προστασία των μεταδεδομένων του συστήματος. Επίσης, η συγχώνευση του αρχείου καταγραφών με την εικόνα του συστήματος βοηθάει, καθώς μετά από μεγάλο χρονικό διάστημα αυξάνεται το μέγεθος του αρχείου και δημιουργείται πρόβλημα, εξαιτίας της αυξανόμενης πιθανότητας απώλειας ή καταστροφής του. Επιπλέον, το μέγεθος του ημερολογίου επηρεάζει τον χρόνο που απαιτείται για την επανεκκίνηση του NameNode. Όταν ο NameNode λάβει το ρόλο του BackupNode, επιτελεί την ίδια λειτουργία με τον CheckpointNode· με την διαφορά, ότι διατηρεί στην μνήμη του ένα ενημερωμένο αντίγραφο της εικόνας του συστήματος, το οποίο είναι συνεπές με το NameNode. Έτσι, ο BackupNode, δεν χρειάζεται να επικοινωνήσει με τον NameNode για να αποθηκεύσει την εικόνα του συστήματος στο τοπικό του δίσκο, καθώς έχει στην μνήμη την συνεπή εικόνα του συστήματος. Το παραπάνω καθιστά την λειτουργία πιο αποδοτική.



Εικόνα 2.1: *Η αρχιτεκτονική του HDFS*

2.1.4 Hadoop DataNode

Στην φάση της αρχικοποίησης, ένα DataNode συνδέεται με το NameNode, με την εκτέλεση ενός handshake. Το παραπάνω αποσκοπεί στην επικύρωση της έκδοσης του συστήματος σε χρήση καθώς και του namespaceID του. Εάν κάποιο από τα δύο δεν ταιριάζει με τα ανάλογα στο NameNode, τότε το αντίστοιχο DataNode απενεργοποιείται [7].

Όταν ένα DataNode – το οποίο αρχικοποιείται για πρώτη φορά – εισέρχεται στο Cluster παίρνοντας ένα νέο namespaceID. Να σημειωθεί ότι οι DataNodes έχουν ένα μοναδικό storageID, το οποίο επιτρέπει να τους διαχωρίζουμε ακόμα και στις περιπτώσεις που έχουμε αλλαγή στην IP. Το storageID αρχικοποιείται την στιγμή που ο DataNode συνδεθεί για πρώτη φορά με τον NameNode και έκτοτε δεν αλλάζει. Το block κάθε DataNode συνίσταται από τα εξής δύο αρχεία α) ένα αρχείο που περιέχει τα δεδομένα του block και β) ένα άλλο αρχείο που περιέχει μεταδεδομένα, όπως το checksum του αντίστοιχου block. Ο DataNode server αναγνωρίζει ποια blocks έχει στην κατοχή του και στέλνει σχετική αναφορά ενημέρωσης στον NameNode. Η συγκεκριμένη αναφορά εμπεριέχει διάφορες πληροφορίες, όπως το blockid, την σφραγίδα δημιουργίας, το μέγεθος του block,

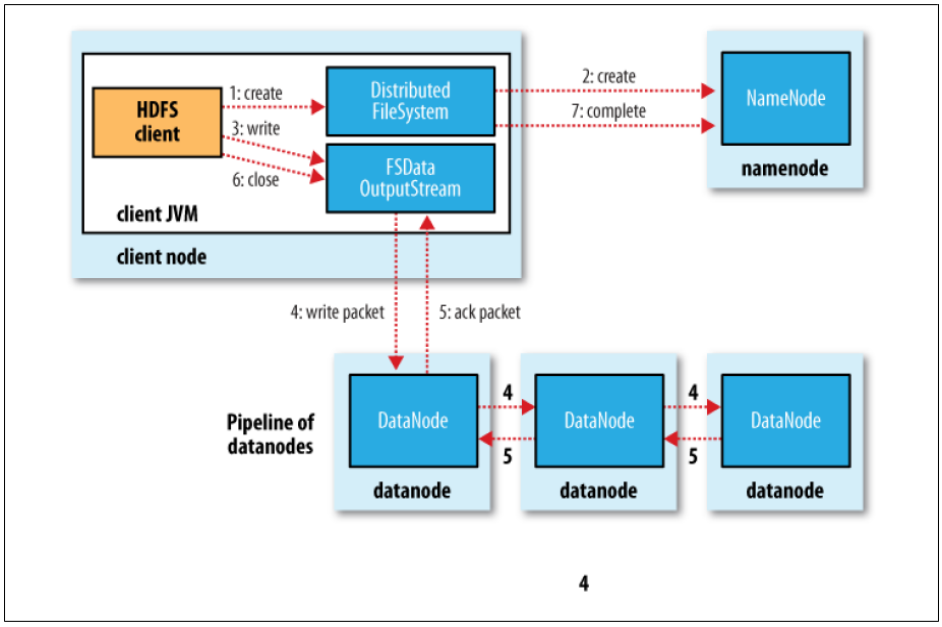
οι οποίες αποστέλλονται στην αρχή της εγγραφής του DataNode και στην συνέχεια, περιοδικά ανά μία ώρα.

Επιπλέον, υπό κανονική λειτουργία, ο DataNode στέλνει μηνύματα στον NameNode ανά χρονικά διαστήματα, για να τον ενημερώσει ότι λειτουργεί κανονικά. Εάν ο NameNode δεν λάβει το παραπάνω μήνυμα από τον DataNode σε χρόνο μεγαλύτερο από 10 λεπτά, τότε εκτιμά ότι ο DataNode είναι εκτός υπηρεσίας. Συνακόλουθα, διαγράφει τον συγκεκριμένο DataNode και δρομολογεί την δημιουργία νέων DataNodes, τα οποία θα περιέχουν τα αντίστοιχα blocks. Είναι σημαντικό να αναφερθεί ότι τα μηνύματα, τα οποία χρησιμοποιούνται για την επιβεβαίωση της λειτουργίας του DataNode, ενέχουν πληροφορίες για α) την χωρητικότητα του αποθηκευτικού χώρου που είναι διαθέσιμος ή είναι σε χρήση καθώς και β) τις – σε εκκρεμότητα – αιτήσεις για τα blocks. Ο NameNode δεν καλεί άμεσα τα DataNodes, αλλά μέσα από τις απαντήσεις στα μηνύματα επιβεβαίωσης στέλνει οδηγίες όπως, αντιγραφής των blocks σε άλλους κόμβους, διαγραφής τοπικών αντιγράφων, τερματισμού λειτουργίας του κόμβου και αποστολής αναφοράς για τα blocks.

Κάθε DataNode ελέγχει, περιοδικά, την εγκυρότητα των blocks που κατέχει. Ο έλεγχος αυτός επιτυγχάνεται με την χρήση του blockscanner, ο οποίος συγκρίνει τα checksums στα αντίγραφα των άλλων κόμβων ούτως ώστε να επικυρώσει την εγκυρότητά τους. Όταν εντοπιστεί ένα κατεστραμμένο block, τότε ενημερώνεται ο NameNode ο οποίος εντοπίζει το εν λόγω block. Εν συνεχεία, διαγράφει το block αφού πρώτα, δημιουργήσει ένα αντίγραφο του block χωρίς λάθη. Αυτό σημαίνει ότι, ακόμα και όταν τα δεδομένα είναι λανθασμένα, ο χρήστης μπορεί να τα ανακτήσει στο ενδεχόμενο που τα χρειαστεί.

Οι εφαρμογές του χρήστη έχουν πρόσβαση στο HDFS, μέσω του HDFS CLIENT ο οποίος, με την σειρά του, αξιοποιεί την βιβλιοθήκη του Hadoop. Η βιβλιοθήκη περιέχει μία διεπαφή η οποία επιτρέπει την πρόσβαση σε αρχεία του συστήματος. Ο χρήστης δεν απαιτείται να ξέρει πώς λειτουργεί το σύστημα καθώς αυτή η διεπαφή υποστηρίζει την διαφανή πρόσβαση στο σύστημα αρχείων. Όταν ένας client αναγνώσει ένα αρχείο, ζητάει από το NameNode μία

λίστα με τους DataNodes, οι οποίοι περιέχουν τα αντίγραφα των blocks του αρχείου. Κατόπιν, επικοινωνεί άμεσα με τους DataNodes για την ανάκτηση των αντίστοιχων blocks. Κατ' αναλογία, όταν ένας client θέλει να γράψει, ζητάει να επιλέξει τους DataNodes (από το NameNode), όπου θα αποθηκευτούν τα αντίγραφα του πρώτου block. Προχωρά στην εγγραφή του αντίστοιχου στο DataNode που επέλεξε, χρησιμοποιώντας την τεχνική pipelining. Κατά το τέλος της εγγραφής, ο client εκ νέου ζητάει από το NameNode να επιλέξει τους DataNodes, οι οποίοι θα φιλοξενήσουν το επόμενο block. Αυτή η διαδικασία συνεχίζεται έως ότου όλα τα blocks του αρχείου καταγραφούν στο HDFS. Στην εικόνα 2.2 περιγράφεται η διαδικασία που ο client ακολουθεί για την εγγραφή ενός αρχείου.



Εικόνα 2.2 :*Διαδικασία εγγραφής ενός αρχείου σε HDFS*

Η βιβλιοθήκη , η οποία παρέχεται για την επικοινωνία με το HDFS, παρέχει την επιπρόσθετη δυνατότητα αναγνώρισης της τοποθεσίας των blocks στο HDFS. Με αυτόν τον τρόπο, σε εφαρμογές όπως π.χ., στο MapReduce, δύναται να α) εντοπίζουν την τοποθεσία των αρχείων και β) χρησιμοποιούν αυτή την πληροφορία για την οργάνωση των εργασιών, με απώτερο στόχο την βελτίωση της απόδοσης. Επίσης, η συγκεκριμένη βιβλιοθήκη επιτρέπει τον ορισμό του παράγοντα αντιγραφής (replication factor) για τα αρχεία. Η αύξηση της απόδοσης

του συστήματος επιτυγχάνεται με την αύξηση των αντιγράφων. Συνεπώς, μπορούμε σε προσπελάσιμα αρχεία συχνά, να αυξήσουμε το replication factor.

2.1.5 MapReduce

Ο μηχανισμός του MapReduce συμβαδίζει με το μοντέλο αρχιτεκτονικής master / slave [9]. Ο κεντρικός master server, jobtracker, αναλαμβάνει την κατανομή των εργασιών στους υπόλοιπους κόμβους, slaveservers – tasktrackers. Πιο συγκεκριμένα, ο χρήστης στέλνει τις map και reduce εργασίες του στον jobtracker · κατόπιν, ο τελευταίος στέλνει τις εργασίες για εκτέλεσή τους στους tasktrackers με την πολιτική First In First Out (FIFO). Οι tasktrackers έχουν την ιδιότητα να εκτελούν τις – υπό ανάθεση από τον jobtracker – εργασίες. Υπενθυμίζεται ότι, εάν ένας tasktracker αποτύγχανε, τότε η αντίστοιχη εργασία έπρεπε να εκτελεστεί από την αρχή (το παραπάνω ίσχυε, πριν από την έκδοση 0.21).

2.1.6 Λειτουργία Map Reduce

Κατά το αρχικό στάδιο, η είσοδος μοιράζεται σε M κομμάτια. Τα M κομμάτια μπορούν να επεξεργαστούν εν παραλλήλω, με το καθένα να πηγαίνει σε διαφορετική διεργασία map. Τα ενδιάμεσα κλειδιά κατανέμονται στους R Reducers με την χρήση κάποιας συνάρτησης κατακερματισμού (παραδείγματος χάρη hash (key) mod R). Οι ενέργειες της συνολικής εκτέλεσης γίνονται με την εξής σειρά (βλέπε εικόνα 2.3) [8].

i. Η βιβλιοθήκη MapReduce διασπά την είσοδο σε M κομμάτια (συνήθως μεγέθους 16-64Mb) και ξεκινά αρκετά αντίγραφα του προγράμματος στους κόμβους.

ii. Ο master node - jobtracker αναλαμβάνει να αναζητήσει τους μη-ενεργούς κόμβους (tasktrackers) και να τους αναθέσει μια εργασία map είτε reduce.

iii. Εάν έχει ανατεθεί στον κόμβο (tasktracker) μια εργασία map, τότε ο κόμβος α) διαβάζει το κομμάτι εισόδου που του αντιστοιχεί, β) το διαμορφώνει ως κλειδί-τιμή και γ) το δίνει, ως παράμετρο, στην συνάρτηση map. Τα ενδιάμεσα αποτελέσματα παραμένουν στην μνήμη.

iv. Με βάση την συνάρτηση κατακερματισμού, τα ενδιάμεσα αποτελέσματα γράφονται στον τοπικό δίσκο (σε R μέρη). Οι θέσεις των αποτελεσμάτων στέλνονται στον master-jobtracker, ο οποίος – με την σειρά του – είναι υπεύθυνος να τις διαμοιράσει στους R Reducers-tasktrackers.

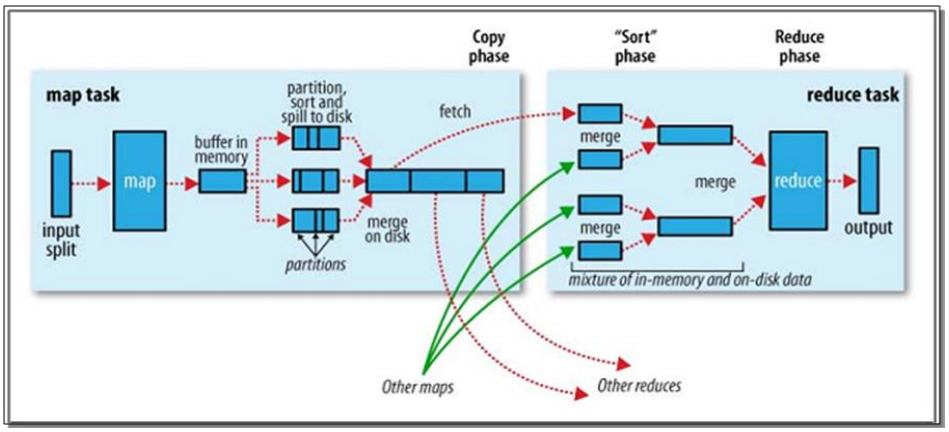
v. Όταν κάποιος Reducer ειδοποιηθεί για αυτές τις τοποθεσίες, διαβάζει απομακρυσμένα τα δεδομένα από τον δίσκο του αντίστοιχου Mapper. Μετά την ανάγνωση όλων των δεδομένων, α) ταξινομεί τα δεδομένα με βάση το κλειδί και β) ομαδοποιεί τις τιμές, οι οποίες αντιστοιχούν στο ίδιο κλειδί. Στην περίπτωση που τα δεδομένα δεν χωράνε στην μνήμη, τότε χρησιμοποιείται κάποιος αλγόριθμος εξωτερικής ταξινόμησης.

vi. Τα ταξινομημένα, ενδιάμεσα αποτελέσματα λαμβάνονται υπόψη στην συνάρτηση reduce ώστε να παραχθεί η τελική έξοδος, η οποία καταγράφεται στο τέλος ενός αρχείου για την έξοδο του συγκεκριμένου διαμερισμού.

vii. Με την ολοκλήρωση όλων των εργασιών στους tasktrackers (map και reduce), ο master-jobtracker επιστρέφει · η εκτέλεση συνεχίζεται από τον κώδικα του χρήστη.

Στο κατάλογο, που έχει ορισθεί για την έξοδο, υπάρχουν R αρχεία (ένα για κάθε Reducer). Συνήθως, αυτά χρησιμοποιούνται ως είσοδοι σε κάποια άλλη MapReduce εργασία και έτσι, δεν απαιτείται η συγχώνευσή τους. Τα ονόματα των αρχείων, ο αριθμός των Reducers καθώς και ο διαμερισμός της εισόδου μπορούν να οριστούν από τον χρήστη.

Σε όλες τις περιγραφόμενες φάσεις εκτελείται η διαχείριση σφαλμάτων. Συνεπώς, σε περίπτωση αποτυχίας, το Hadoop μπορεί είτε να επανεκκινήσει την αποτυχημένη εργασία σε κάποιο άλλο κόμβο είτε να ανακάμψει και να συνεχίσει την εκτέλεση με μικρό overhead (Hadoop 0.21). Το Hadoop μπορεί να επιτύχει ικανοποιητικό Load balancing, εξαιτίας του μεγάλου αριθμού tasktrackers.



Εικόνα 2.3: *Υλοποίηση Map Reducer με load balancing*

Ένα παράδειγμα ψευδό-κώδικα εφαρμογής MapReduce φαίνεται αμέσως παρακάτω. Στο συγκεκριμένο παράδειγμα γίνεται εφαρμογή της wordcount, όπου εντοπίζει το πλήθος των εμφανίσεων κάθε λέξης σε ορισμένα κείμενα που λαμβάνουμε ως είσοδο. Ο Mapper λαμβάνει σαν είσοδο ένα split από τα αντίστοιχα κείμενα. Το key ορίζεται ως ο αριθμός της γραμμής του κειμένου, ενώ το value ορίζεται ως τα δεδομένα της γραμμής. Κατανέμουμε τις γραμμές σε λέξεις · για έξοδο δίνουμε ένα ζευγάρι (key, value) λέξης και μονάδας. Τα ζευγάρια αυτά μοιράζονται σύμφωνα με τον defaultpartitioner στους Reducers, με την χρήση της τεχνικής hashing. Συνεπώς, κάθε reducer παίρνει όλα τα ζευγάρια, συγκεκριμένων λέξεων. Στην συνέχεια, για κάθε λέξη χρησιμοποιείται ένας counter με σκοπό να υπολογιστεί το πλήθος των εμφανίσεων. Τέλος, στην έξοδο αναγράφεται το ζευγάρι (key,value), το οποίο αντιστοιχεί στην λέξη και στην συχνότητα εμφάνισής της. Ένα απλό παράδειγμα ψευδοκώδικα για Mapper - Reducer είναι το παρακάτω :

```
Mapper {

Map (key, value) : words = value.split()

For word in words : output(word, one)

}

Reducer  {

Reduce(key, list<value>): sum = 0

For v in list <value>: sum +=v

output (key, sum)

}
```

Ο βασικός κώδικας της εργασίας βασίζεται σε MapReduce ενώ περιέχει και Mapper και Reducer σαν ξεχωριστές κλάσεις (η ανάλυσή του έπεται σε επόμενο κεφάλαιο, αυτό του σχεδιασμού).

2.2 PostgreSQL

Η PostgreSQL αποτελεί μια – ανοιχτού κώδικα – σχεσιακή βάση δεδομένων, με διάρκεια ανάπτυξής της περισσότερη από 20 χρόνια. Βασίζεται σε μια, αποδεδειγμένα, καλή αρχιτεκτονική με ισχυρή την αντίληψη των χρηστών της γύρω από την αξιοπιστία, την ακεραιότητα δεδομένων και την ορθή λειτουργία. Η PostgreSQL τρέχει σε όλα τα βασικά λειτουργικά συστήματα, περιλαμβάνοντας Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64) καθώς και Windows [10].

Είναι συμβατή με ACID (ατομικότητα, συνέπεια, απομόνωση, μονιμότητα) και συμπεριλαμβάνει τους περισσότερους SQL92 και SQL99 τύπους δεδομένων, συμπεριλαμβανομένων των INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL και TIMESTAMP. Επιπλέον, υποστηρίζει την αποθήκευση μεγάλων δυαδικών αντικειμένων (binary), όπως παραδείγματος χάρη εικόνων, ήχων ή βίντεο. Διαθέτει α) περιβάλλοντα προγραμματισμού για τις γλώσσες προγραμματισμού C, C++, Java, Perl, Python, Ruby, Tcl, καθώς και β) υποστήριξη για την πλατφόρμα .NET και το πρότυπο ODBC.

Στην παρούσα εργασία, χρησιμοποιείται PostgreSQL ως την βάση αποθήκευσης των δεδομένων με το Parsing των XML αρχείων (πιο εκτενή περιγραφή στο κεφάλαιο του σχεδιασμού).

Η διαχείριση της βάσης δεδομένων γίνεται μέσω του εργαλείου pgAdmin αλλά και μέσω άλλων εργαλείων, όπως παραδείγματος χάρη των PgAccess, PhpPgAdmin και WinSQL. Η συγκεκριμένη υπηρεσία εγκαθίσταται ταυτόχρονα, με την βάση δεδομένων και υποστηρίζεται σε πολλές πλατφόρμες υπολογιστών (Windows, MacOS X, Linux, Solaris, FreeBSD κ.α.) [11].

Πιο συγκεκριμένα, η pgAdmin έχει σχεδιαστεί για να ικανοποιήσει τις βασικές ανάγκες των χρηστών, δηλαδή, από απλό γράψιμο sql ερωτημάτων έως και την ανάπτυξη πολύπλοκων βάσεων δεδομένων. Ήταν αρχικά γνωστή με την ονομασία pgManager· ενώ, αργότερα (συγκεκριμένα, από το 1998) πήρε τη σημερινή επωνυμία (δηλαδή, pgAdmin). Είναι πλήρως γραμμένη στην γλώσσα

προγραμματισμού C++, με χρήση των wxWidgets ώστε να διασφαλίζεται η συμβατότητά της με τα κοινά λειτουργικά συστήματα (που αναφέρθηκαν παραπάνω).

Η σύνδεση με την βάση δεδομένων επιτυγχάνεται, επιλέγοντας το εικονίδιο "add a connection to a server". Κατόπιν, εισάγουμε το όνομα χρήστη – το οποίο είναι "postgres" – καθώς και τον κωδικό πρόσβασης, το οποίο είχε επιλεγεί κατά την διαδικασία εγκατάστασης. Όσον αφορά την σύνδεση στο διακομιστή μπορεί να γίνει με την χρήση του πρωτοκόλλου TCP / IP είτε με την χρήση Unix Sockets· επίσης, δύναται να είναι κρυπτογραφημένη (SSL) για μεγιστοποίηση της ασφάλειας.

2.3 HadoopDB

Το Hadoop DB ορίζεται ως μία υβριδική αρχιτεκτονική μεταξύ του MapPeduce και των σχεσιακών βάσεων δεδομένων (RDBMS) [12]. Η χρήση του Hadoop DB μας επιτρέπει να:

- "τρέχουμε" εφαρμογές πάνω από Clusters, όπου δεν απαιτείται ο διαμερισμός τους, είτε από Cloud.

- επιτυγχάνεται η αναβάθμισή του πιο εύκολα από τα καθιερωμένα parallel DBMS.

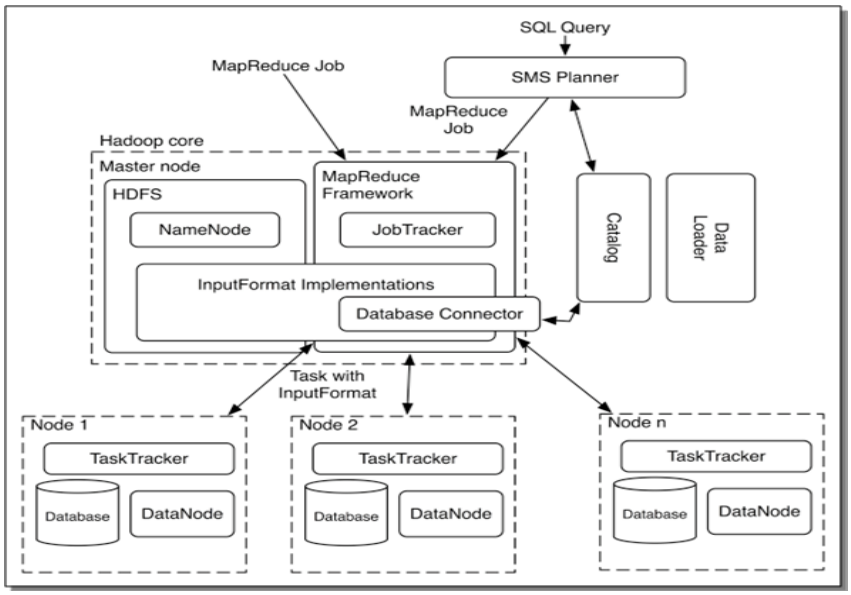
- επεκτείνεται τόσο εύκολα όσο το Hadoop, με συνακόλουθη την καλύτερη απόδοση σε σχέση με τα καθιερωμένα συστήματα (και ειδικά, στην ανάλυση δομημένων δεδομένων).

- καλυφθεί το κενό στην αγορά, μεταξύ ενός δωρεάν και ενός ανοικτού κώδικα parallel DBMS.

2.3.1 Λειτουργία HadoopDB

Η βασική ιδέα του HadoopDB (βλέπε εικόνα 2.4) είναι ότι παρέχει στο Hadoop πρόσβαση σε πολλούς single-node εξυπηρετητές (για παράδειγμα, PostgresSQL), οι οποίοι είναι διασκορπισμένοι γύρω από το Cluster. Το Hadoop έχει την δυνατότητα να "σπρώχνει" όσα περισσότερα δεδομένα μπορεί, για επεξεργασία. Το παραπάνω υλοποιείται μέσα στην μηχανή της βάσης δεδομένων, κάνοντας SQL ερωτήματα. Στη συνέχεια, δημιουργείται ένα σύστημα, το οποίο μοιάζει με ένα parallel RDBMS και δεν έχει κοινοχρηστία με κάτι άλλο. Αξιοποιώντας τεχνικές από τον χώρο των βάσεων δεδομένων, η συγκεκριμένη αρχιτεκτονική επιφέρει καλύτερη απόδοση και ειδικά, σε πιο πολύπλοκα συστήματα ανάλυσης δεδομένων (Βλέπε εικόνα 2.4, την αρχιτεκτονική του HadoopDB).

Ταυτόχρονα, το γεγονός ότι τα αρχεία του HadoopDB βασίζονται πάνω στο mapPeduce επιτυγχάνεται η επεκτασιμότητα και μια επαρκής (fault tolerance) ανοχή σε σφάλματα (όπως ομοίως και το Hadoop). Καταλήγοντας, πρέπει να σημειωθεί ότι το HadoopDB δημιουργήθηκε πάνω σε προγράμματα ανοικτού κώδικα όπως είναι για παράδειγμα το Hive, το οποίο προσδίδει SQL διεπαφή στο σύστημα.



Εικόνα 2.4: *Η αρχιτεκτονική του HadoopDB*

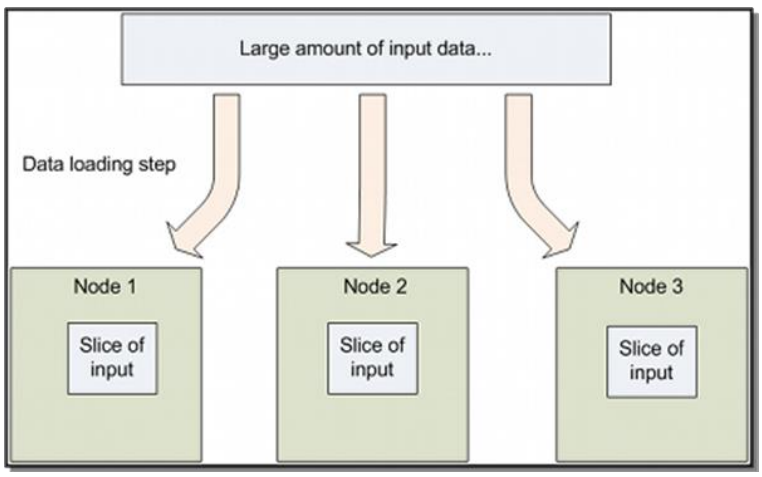
2.4 Πλατφόρμα Hortonworks

Σε αυτή την εργασία έγινε εκτενή περιγραφή της χρήσης και της εγκατάστασης του Hadoop με σκοπό στην καλύτερη κατανόησή του [13]. Εντούτοις, όπως θα αναλυθεί και στο επόμενο κεφάλαιο, θεωρήθηκε σκόπιμο να προβούμε στην υλοποίηση της εργασίας, βασισμένοι στην χρήση της πλατφόρμας Hortonworks.

Η πλατφόρμα της Hortonworks αναδύθηκε, στην αγορά, το 2011 από την Yahoo και την Benchmark Capital. Έχει βασιστεί στο Apache Hadoop και έχει σχεδιαστεί ως ένα λογισμικό ανοιχτού κώδικα. Αυτό το λογισμικό επιτρέπει την ενσωμάτωση του Apache Hadoop μέσα από καθιερωμένες αρχιτεκτονικές διαχείρισης δεδομένων όπως, RDBMS συστήματα και Data Warehouses. Το έτος 2012 σε μία συνεργασία με την εταιρία Rackspace παρουσιάστηκε το OpenStack-based Hadoop με δυνατότητα χρήσης δημόσιου και ιδιωτικού Cloud στις πλατφόρμες.

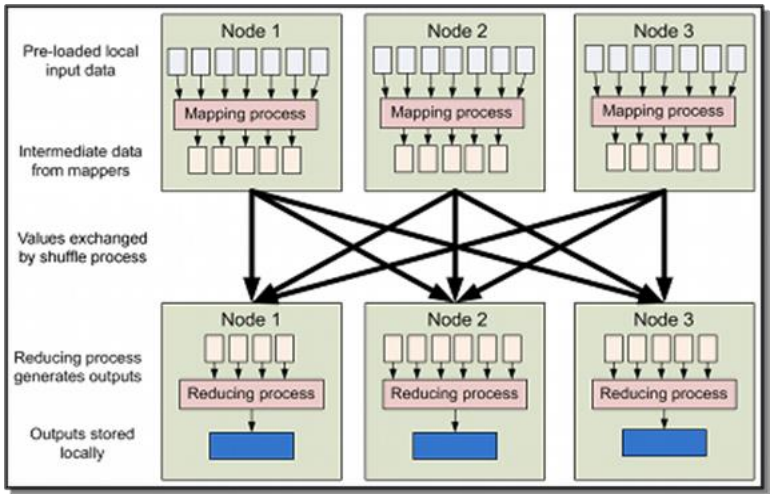
2.4.1 Λειτουργία της Πλατφόρμας Δεδομένων Hortonworks

Το Sandbox της Hortonwoks αφορά σε υλοποίηση της πλατφόρμας των δεδομένων της Hortonworks (HDP). Βασίζεται σε ένα Virtual Machine, το οποίο επιτρέπει την χρήση του HDP, με άμεσο τρόπο [14]. Η λειτουργία του απεικονίζεται στην εικόνα 2.5.



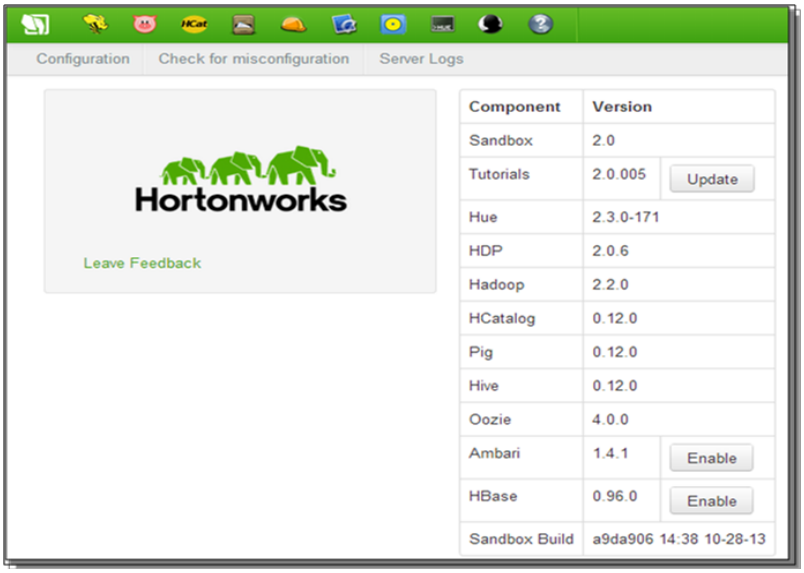
Εικόνα 2.5: *Βασική Υλοποίηση Hortonworks Hadoop*

Το project Hortonworks Hadoop περιλαμβάνει μία σειρά εργαλείων, τα οποία χρησιμοποιούνται για προβλήματα διαχείρισης μεγάλου όγκου δεδομένων. Το Cluster της πλατφόρμας υλοποιεί παράλληλα επεξεργαστικά συστήματα, βασισμένο στην αρχιτεκτονική του Apache Hadoop και χωρίς την χρήση ακριβού υπολογιστικού υλικού. Με τον διασκορπισμό του Cluster σε διαφορετικούς κόμβους, η επεξεργασία κατευθύνεται προς τα δεδομένα· αυτό σημαίνει ότι οι κόμβοι δεν χρειάζεται να αναμένουν τα δεδομένα ούτως ώστε να αρχίσει η επεξεργασία τους. Με αυτό τον τρόπο, διασφαλίζεται και η παράμετρος της επεκτασιμότητας (scalability). Οι κόμβοι επεξεργάζονται μέρος των δεδομένων κάνοντας χρήση του MapReduce (Βλέπε εικόνα 2.6).



Εικόνα 2.6: *Hortonworks Hadoop υλοποίηση του Mapreduce.*

Η υλοποίηση του Hortonworks Hadoop βασίζεται στο Cent OS. Επίσης, η εγκατάστασή του πραγματοποιήθηκε με την χρήση VirtualBox της Oracle (βλέπε εικόνα 2.7).



Εικόνα 2.7: *HortonWorks Web browser presentation*

Στο σημείο αυτό, ολοκληρώθηκε η ανάλυση των σημαντικότερων εργαλείων της παρούσας εργασίας. Στο επόμενο κεφάλαιο θα γίνει αναφορά α) της διαδικασίας σχεδιασμού του συστήματός μας αξιοποιώντας το Hadoop MapReduce καθώς και β) της προτεινόμενης μεθοδολογίας [15].

3. Σχεδιασμός Συστήματος

Ο σχεδιασμός της εργασίας πραγματοποιήθηκε επί τη βάσει μιας σειράς βημάτων, καθώς και εργαλείων, τα οποία θα αναλυθούν στην συνέχεια, ως το μεθοδολογικό πλαίσιό της.

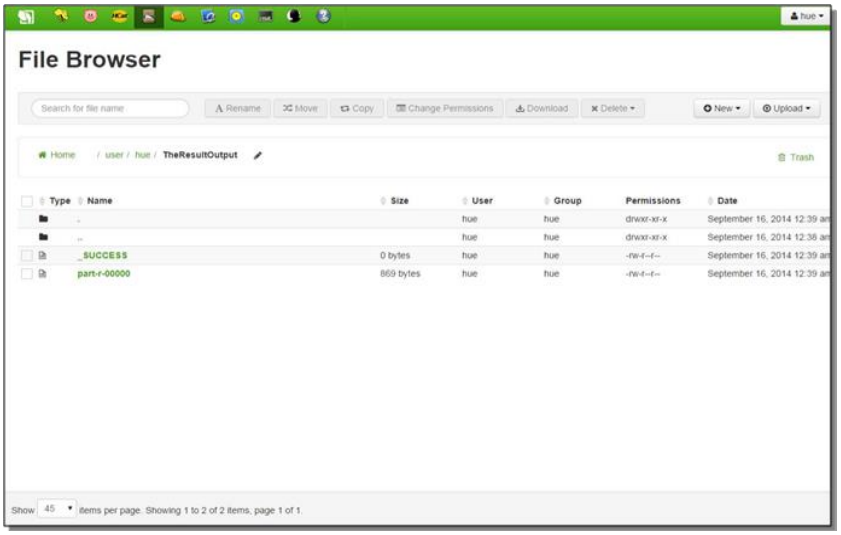
3.1 Εγκατάσταση Hadoop & Υποσυστημάτων

Η αρχική εγκατάσταση του Hadoop πραγματοποιήθηκε με την λήψη του αρχείου Apache Hadoop, σε συστήματα Ubuntu Linux 14.06. Παρ' όλη την ομαλή ολοκλήρωση της εγκατάστασης, προέκυψαν πολλαπλά προβλήματα κατά την διάρκεια της α) μεταγλώττισης και β) εκτέλεσης των προγραμμάτων Hadoop και MapReduce.

Όμοια προβλήματα δημιουργήθηκαν και όταν η εγκατάσταση υλοποιήθηκε σε περιβάλλον Debian 6. Στη συνέχεια, γίνεται μικρή αναφορά στα εν λόγω προβλήματα.

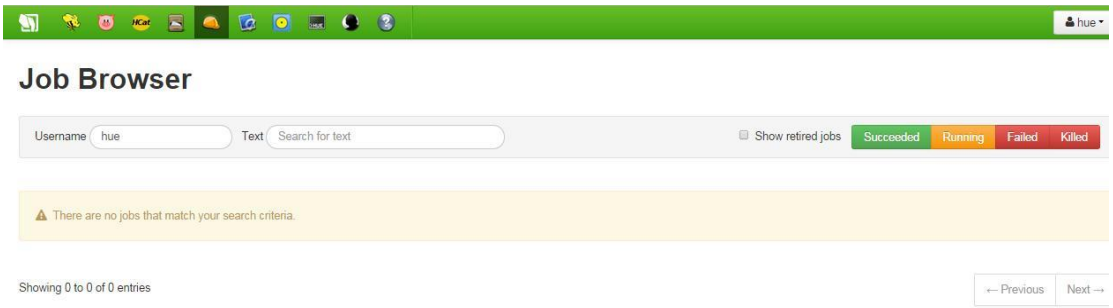
- προγράμματα Java με ClassNotFoundException,
- περιορισμένες πηγές, με σωστές υλοποιήσεις του Hadoop MapReduce,
- καμία υλοποίηση, η οποία χρησιμοποιεί τις hadoop.util.ToolRunner και hadoop.util.Tool βιβλιοθήκες, δεν ″τρέχει″,
- ασταθές σύστημα (ειδικά, σε Debian) και
- μεγάλος χρόνος εγκατάστασης με σωστές παραμέτρους.

Λαμβάνοντας υπόψη τα παραπάνω, ελήφθη η απόφαση χρήσης του Hortonworks (ανατρέξτε, για την ανάλυσή του, στο προηγούμενο κεφάλαιο). Η χρήση εργαλείων του Hortonworks, όπως το File Browser απεικονίζεται στην εικόνα 3.1.



Εικόνα 3.1: *Ο File Browser του Hortonworks*

Ένα σημαντικό εργαλείο του Hortonworks είναι ο JobBrowser. Ο JobBrowser επιτρέπει και την επισκόπηση όλων των εργασιών που τρέχουν παράλληλα, στο Hortonworks (**Εικόνα 3.2**).



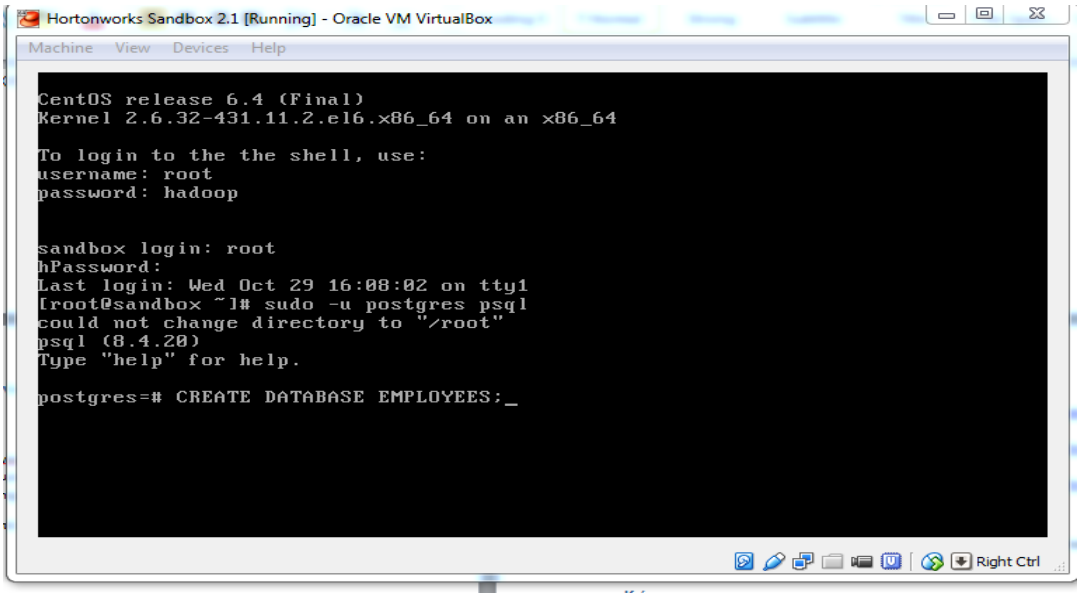
Εικόνα 3.2: *Ο Job Browser του Hortonworks*

Η τελική υλοποίηση της εφαρμογής πραγματοποιήθηκε στο Hortonworks, καθώς παρουσίασε πιο σταθερή και αξιόπιστη πλατφόρμα [16].

3.2 Εγκατάσταση Βάσης Δεδομένων PostgreSQL & Δημιουργία Βάσης δεδομένων

Η PostgreSQL προϋπήρχε στο Hortonworks στην έκδοση 8.4. Επίσης, επισημαίνεται ότι είναι ήδη εγκατεστημένη μέσα στο Hortonworks. Έπρεπε να δημιουργηθεί μία βάση δεδομένων ώστε αρχικά, να κρατήσει τα στοιχεία -που γίνονται parsing- από τα XML έγγραφα [17].

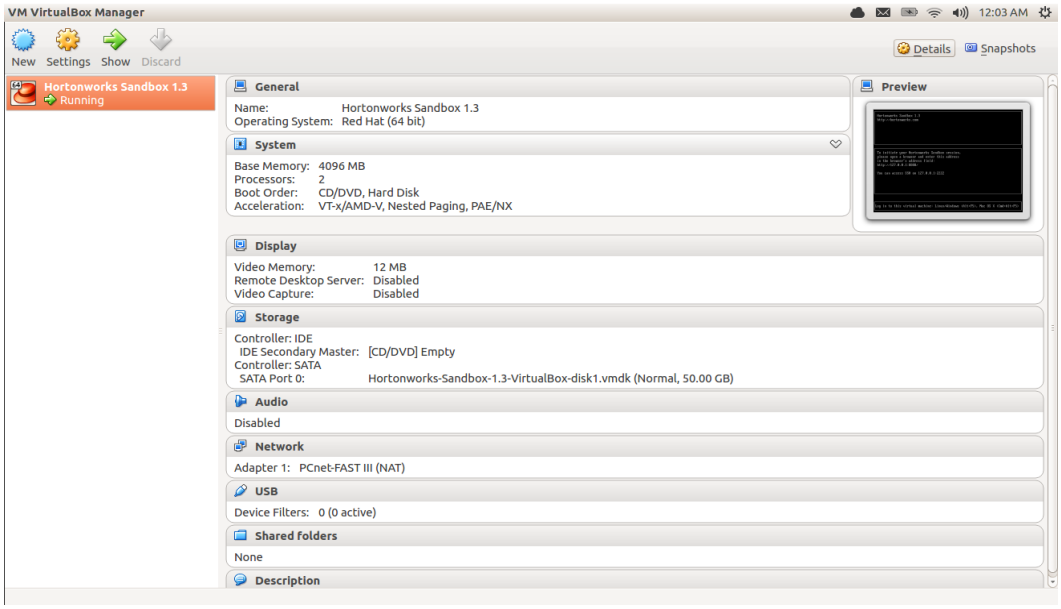
Η χρήση του PostgreSQL έγινε μέσα από το Cent OS και συνεπώς, η βάση δημιουργήθηκε σε αυτό το σημείο. Ένα παράδειγμα χρήσης του Cent OS του Hortonworks φαίνεται στην **Εικόνα 3.3**.



Εικόνα 3.3: *Το Cent OS και η χρήση της PostgreSQL.*

3.3 Oracle VirtualBox και HadoopSandbox

Η εγκατάσταση των κόμβων του Hadoop πραγματοποιήθηκε με την χρήση του Oracle VirtualBox. Μέσω του image, το οποίο κατεβάσαμε από το site της Hortonworks, έγινε εγκατάσταση στο VirtualBox, ως ένα ξεχωριστό λειτουργικό σύστημα (αφού βασίζεται σε Cent OS) [18].



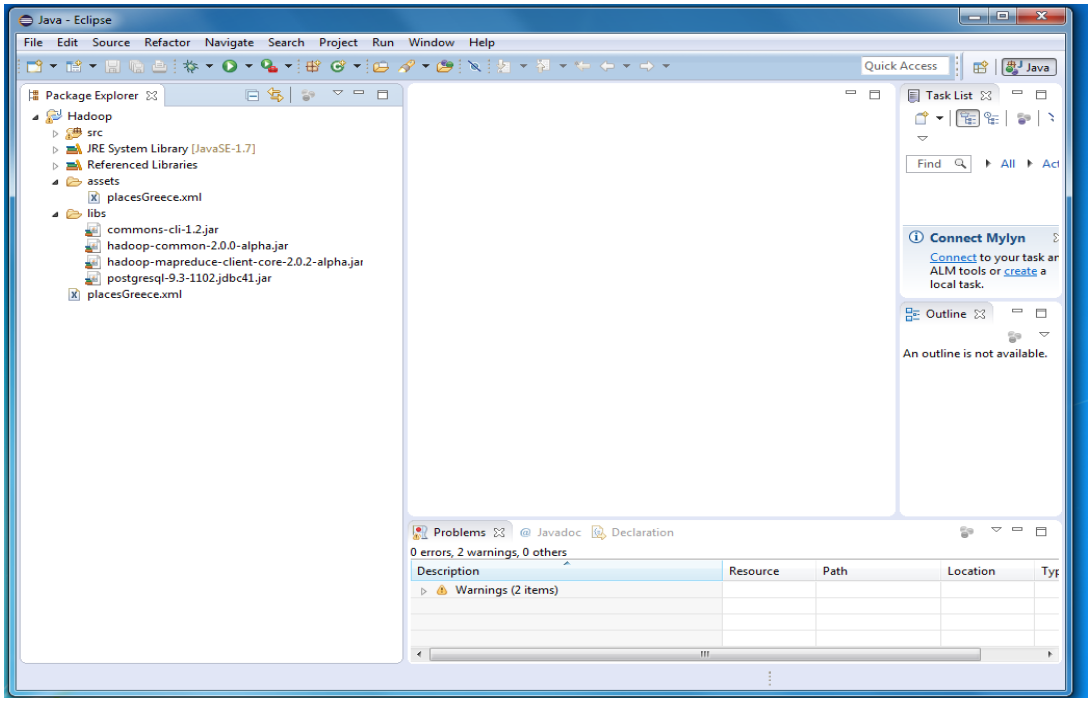
***Εικόνα 3.4**: Oracle VirtualBox με εγκατεστημένο το λογισμικό Hadoop της Hortonworks*

Μια επιπρόσθετη εγκατάσταση έγινε για την χρήση Multi-node MapReduce, βελτιώνοντας περαιτέρω την ταχύτητα ολοκλήρωσης του προγράμματος. Παρόλα αυτά, όπως προαναφέραμε, οι πηγές του υπολογιστή επιβαρύνθηκαν σημαντικά (περαιτέρω ανάλυση σε επόμενο κεφάλαιο).

3.4 Δημιουργία εφαρμογής με Eclipse

Η διαδικασία υλοποίησης με το Eclipse έλαβε χώρα στα Windows. Ακολούθησε η μετατόπιση των αρχείων σε περιβάλλον Hadoop, όπου και έγινε η τελική εκτέλεση του προγράμματος. Τα πακέτα που χρησιμοποιήθηκαν κατανέμονται α) στα πακέτα του Hadoop, τα οποία έγιναν Import στο περιβάλλον, ως External Jars, β) στα πακέτα του postgressql και γ) στα πακέτα της Java, τα οποία χρησιμοποιήθηκαν (Εικόνα 3.5). Τα βασικά πακέτα του Hadoop για χρήση Map αξιοποιούνται και για χρήση mapReduce. Οι βιβλιοθήκες, οι οποίες συμπεριλήφθηκαν στον κώδικα, βασίζονται κυρίως στο hadoop-mapreduce-client-core-2.0.2-alpha.jar.

Στο Jar αρχείο postgresql-9.3-110.jdbc.jar υπάρχουν όλοι οι οδηγοί σύνδεσης με την βάση δεδομένων αναφορικά με την αποθήκευση των δεδομένων από το XML πρωτογενές αρχείο.



Εικόνα 3.5:*Το πρόγραμμα Eclipse με το ProjectHadoop. Οι βιβλιοθήκες*

Τα αρχεία του κώδικα εμπεριέχονται στο φάκελο src και χωρίζονται σε υποκατηγορίες: α) η 1η υποκατηγορία περιλαμβάνει αρχεία σχετικά με τον κώδικα για το MapReduce και β) η 2η υποκατηγορία περιέχει αρχεία τα οποία υλοποιούν parsing στο αρχείο καθώς και την αποθήκευση στην Postgres βάση δεδομένων. Τέλος, σε αυτό το στάδιο ολοκληρώνεται η δημιουργία του LoadBalancingtree. **(Εικόνα 3.6).**

Υποκατηγορία **hadoop.geo:**

- **AggregateJob.java**: σε αυτό το class «τρέχουν» οι συναρτήσεις, οι οποίες αφορούν στο parsing του αρχείου καθώς και στην εκκίνηση του MapReducer.

- **LongSumReducer.java**: πρόκειται για τον κώδικα του Reducer όπου – όπως προαναφέρθηκε – δημιουργεί τα ζευγάρια μεταξύ των τιμών οι οποίες

έχουν επιλεγεί από τον Mapper. Επίσης, ξεκινά η δημιουργία του δέντρου με βάση το LoadBalancing.

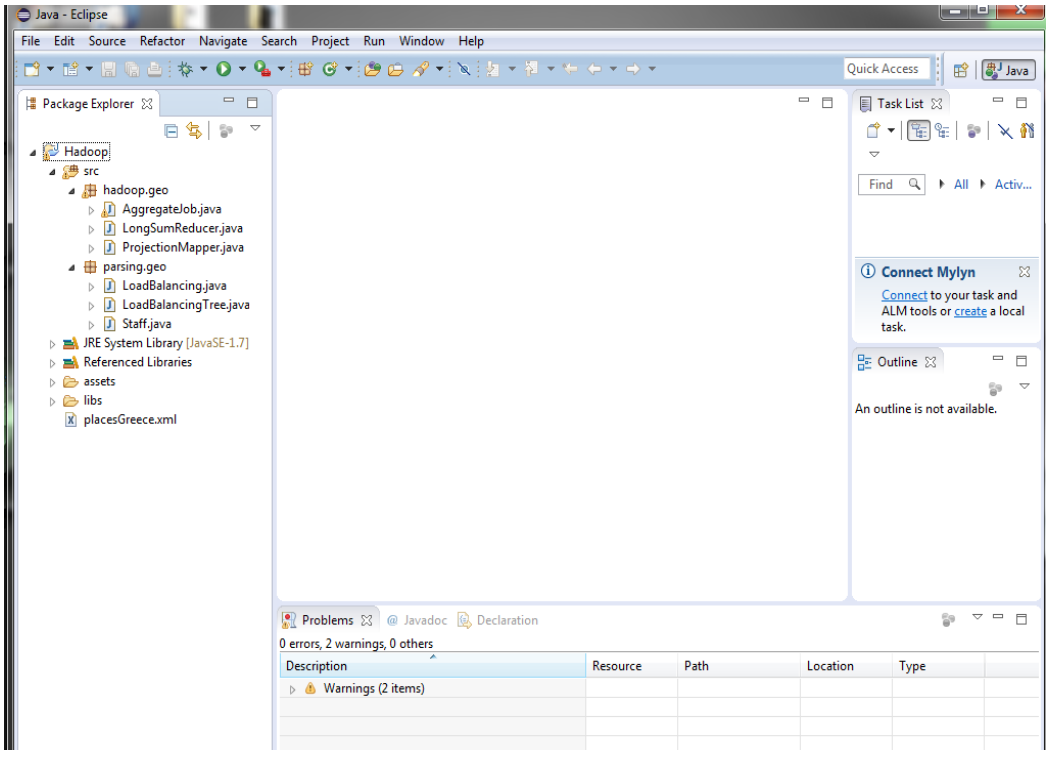
• **ProjectionMapper.java**: πρόκειται για τον κώδικα του Mapper, όπου ξεκινά και το Mapreduce να επεξεργάζεται το αρχείο XML.

Υποκατηγορία **parsing.geo:**

• **LoadBalancing.java**: σε αυτό το σημείο γίνεται η δημιουργία του LoadBalancing δέντρου. Η κατασκευή του δέντρου βασίζεται σε Singleton δέντρα καθώς και σε χρήση του αλγόριθμου το οποίο παρουσιάζεται εκτενώς στο [21].

• **LoadBalancingTree.java**: πραγματοποιείται η υλοποίηση των μεθόδων-τάξεων του Loadbalancing.java αρχείου με στόχο την υλοποίηση του δέντρου. Επιπρόσθετα, πραγματοποιείται και η εκτύπωση των κόμβων (Nodes), οι οποίοι περιέχονται για την επαλήθευση της σωστής κατασκευής του δεντρου.

• **Staff.java**: σε αυτό το σημείο εκτελείται η αποθήκευση στην βάση και η επαναφορά συγκεκριμένων – από την βάση- δεδομένων για την εισαγωγή στους Mappers του Hadoop.



Εικόνα 3.6:*Το πρόγραμμα Eclipse με το ProjectHadoop.Τα αρχεία java*

3.5 Parsing Εγγράφου XML

Η διαδικασία επεξεργασίας του εγγράφου από το Hadoop αφορούσε σε μία πρώτη μετατροπή του εγγράφου bs2.osm από το Openstreetmaps (το οποίο και είχε προταθεί). Αρχικά, η μετατροπή έγινε σε μία μορφή που θα επέτρεπε την σωστή επεξεργασία από το Hadoop [19].

Η σύνθετη και δύσχρηστη αρχική μορφή των bs2.osm αρχείων -κατά το πρώτο parsing- μεγιστοποίησε την ανάγκη για επεξεργαστική ισχύ. Υπό αυτή την συνθήκη, ένας υπολογιστής με δυνατά χαρακτηριστικά (PentiumQuad 3.2 16 GB RAM) δεν μπόρεσε να ανταποκριθεί· πόσο μάλλον, οι υπολογιστές στον Ωκεανό, οι οποίοι διαθέτουν και πιο περιορισμένα χαρακτηριστικά. Για αυτό τον λόγο, προτάθηκε αλλαγή στην σχεδίαση όσο αναφορά στον τρόπο λήψης των δεδομένων από το πρόγραμμα. Πέραν αυτού του νέου δεδομένου, οι τιμές – οι οποίες αναζητήθηκαν – είναι οι παρακάτω [20]:

```
<lat>38,624472</lat>

<long>21,409593</long>

<name>ΑΓΡΙΝΙΟ</name>
```

Η μορφή του ενός osm.bz2 (XML) είναι οι εξής:

osm.bz2 (XML)

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version="0.6" generator="osmconvert 0.8" timestamp="2014-11-
20T21:21:02Z">
        <bounds minlat="42.4276" minlon="1.412368" maxlat="42.65717"
maxlon="1.787481"/>
        <node id="625022" lat="42.5128977" lon="1.5513077" version="5"
timestamp="2013-05-06T20:39:19Z" changeset="16002872" uid="115931"
user="yodeima"/>
```

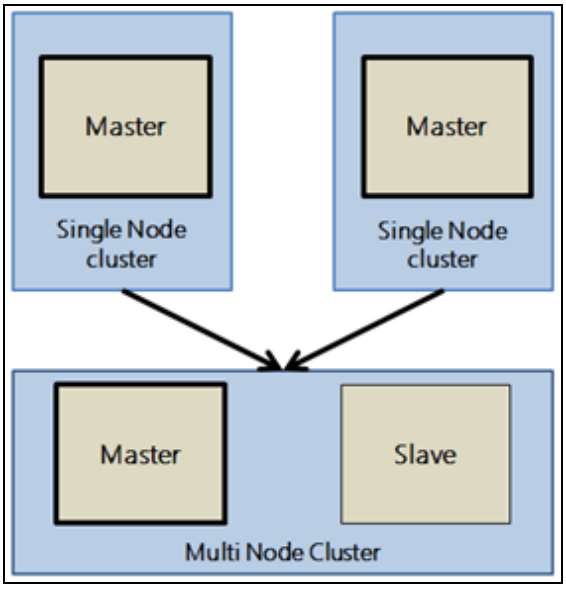
<node id="625023" lat="42.5130943" lon="1.5516419" version="5" timestamp="2014-04-16T13:38:14Z" changeset="21730124" uid="90780" user="Verdy_p"/>

<node id="625024" lat="42.5131808" lon="1.5518583" version="4" timestamp="2014-04-16T13:38:14Z" changeset="21730124" uid="90780" user="Verdy_p"/>

Όπως έχει ήδη ειπωθεί, η παραπάνω συνθήκη έγινε για επεξεργαστικούς λόγους, καθώς ακόμη και σε Multi-node το Hadoop ανέδειξε ανεπαρκή ανταπόκριση δημιουργώντας πρόβλημα τόσο στο λειτουργικό σύστημα όσο και στον υπολογιστή. Μετά από σύντομο χρονικό διάστημα (60-90 δευτερολέπτων), ο επεξεργαστής πλησίαζε το 99% της ισχύος του και το σύστημα “κόλλαγε“. Συνακόλουθα, η ολοκλήρωση του προγράμματος δεν μπορούσε να επιτευχθεί. Βέβαια, σε αυτό το σημείο υπενθυμίζεται η παραδοχή ότι και οι δύο κόμβοι βρίσκονται ως Sandbox στο ίδιο μηχάνημα. Εάν η εγκατάσταση γινόταν σε διαφορετικά μηχανήματα, πιθανά να είχαμε διαφορετικά αποτελέσματα.

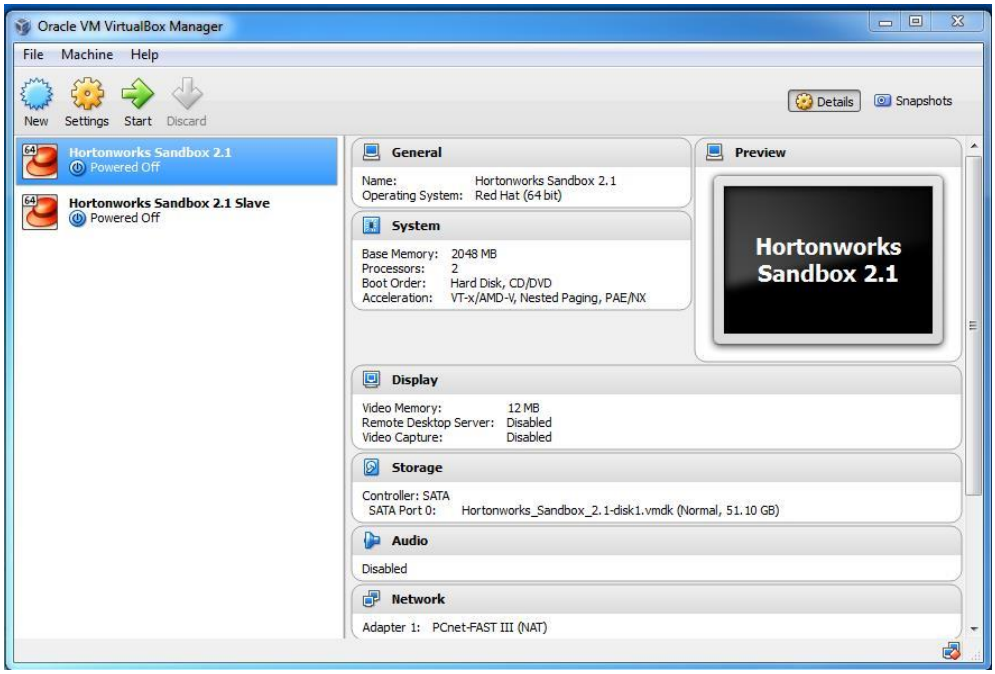
3.6 Hadoop Multi Node και μείωση δικτυακού φόρτου εργασίας

Για την σχεδίαση της πολιτικής εξισορρόπησης φορτίου δικτυακής κίνησης, συνετέλεσε σημαντικά το Multi-Node του Hadoop. Με την εισαγωγή περισσότερων κόμβων στο Hortonworks Hadoop α) οι διεργασίες (process) εκτελέστηκαν πιο γρήγορα και ταυτόχρονα, β) επετεύχθη μία πολιτική εξισορρόπησης του φορτίου εργασίας. Το Hadoop – όπως έχει ήδη ειπωθεί – διαμοιράζει το φορτίο εργασίας σε διαφορετικούς κόμβους. Με αυτό τον τρόπο, επιτρέπεται η λειτουργία περισσότερων Mapper – Reducers, οι οποίοι δουλεύουν παράλληλα σε ένα Cluster, ως master-slave (***Εικόνα* 3.7**). Με την χρήση των οδηγιών για το Horton Hadoop Multi-Node [21].



Εικόνα 3.7: *Multi Node και χρήση του στο Hortonworks Hadoop.*

Αρχικά, δημιουργήσαμε ένα ακόμη Sandbox Hortonworks Hadoop με την ίδια ακριβή διαδικασία. Ονομάστηκε Hortonworks Sandbox 2.1 Slave (**Εικόνα 3.8**).



Εικόνα 3.8: *Εγκατάσταση 2ου Hortonworks Hadoop Sandbox στο Virtual Box*

Η διαδικασία δημιουργίας επιπλέον κόμβων ξεκίνησε με την αλλαγή των IP διευθύνσεων των υπαρχόντων κόμβων (**Εικόνα 3.9**). Η αλλαγή αυτή έγινε μέσω α) της εντολής: sudo vi /etc/hosts καθώς και β) της τροποποίησης του αρχείου Hosts [22].

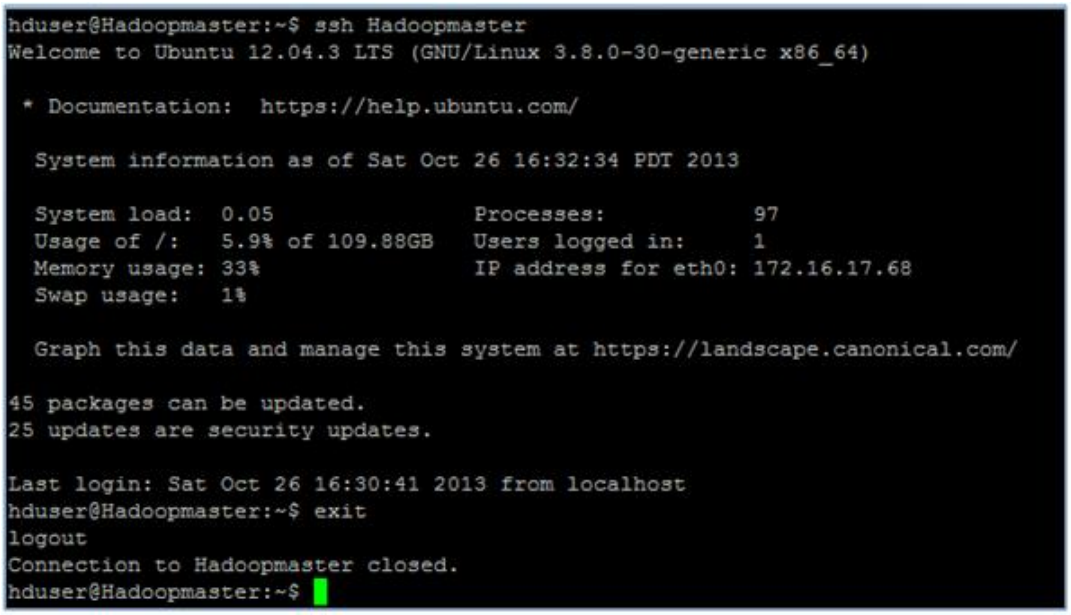
```
172.16.17.68      Hadoopmaster
172.16.17.61      hadoopnode
127.0.0.1       localhost localhost.localdomain

# 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
```

Εικόνα 3.9: *Αλλάζοντας τις IP addresses σε Hadoop Master – Slave*

Με την χρήση του ssh συνδέσαμε τα δύο nodes (ssh Hadoopmaster ssh hadoopnode). Αρχικά, λοιπόν, επιτρέψαμε την χρήση του με την εντολή:
hduser@Hadoopmaster:~$ ssh-copy-id -i ~/.ssh/id_rsa.pub hduser@hadoonode

Με την ολοκλήρωση της διαδικασίας, οι δύο κόμβοι είχαν συνδεθεί (**Εικόνα 3.10**).



Εικόνα 3.10: *Σύνδεση Master - Slave μέσω SSH*

Το τελευταίο στάδιο αφορά στην διαδικασία αλλαγής κάποιων αρχείων στον slave.

Τα αρχεία, τα οποία τροποποιήθηκαν, είναι τα παρακάτω :

i. **core-site.xml**

ii. **mapred-site.xml**

iii. **hdfs-site.xml**

Το core - site.xml στην αρχή, τροποποιείται μέσω της εντολής vi core-site.xml όπως φαίνεται στην **Εικόνα 3.11**. Η τιμή του <value> τροποποιείται σε **hdfs://Hadoopmaster:54310**

```
<configuration>

<property>
  <name>hadoop.tmp.dir</name>
  <value>/data/tmp</value>
 <description>A base for other temporary directories.</description>
</property>

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

Εικόνα 3.11: *Αλλαγή του αρχείου core-site.xml*

Το δεύτερο αρχείο προς αλλαγή είναι το hdfs-site.xml. Η αλλαγή έγινε με τον ίδιο τρόπο, δηλαδή, με την εντολή vi hdfs-site.xml. Σε αυτό το σημείο, ορίζεται ο αριθμός των κόμβων ο οποίος θα αξιοποιηθεί στο Cluster. Αφού υπάρχουν 2 κόμβοι, η τιμή <value> αλλάζει σε 2 (όπως φαίνεται στην **Εικόνα 3.12)**.

```
<configuration>

<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>

</configuration>
```

Εικόνα 3.12: *Η αλλαγή του αριθμού των κόμβων στο αρχείο hdfs-site.xml*

Κατόπιν, τροποποιείται το αρχείο mapred-site.xml. Τοποθετείται ο jobTracker του Master καθώς και το port του. Εκ νέου, με την εντολή vi mapred-site.xml, στο αρχείο τροποποιείται η τιμή <value> από localhost σε Hadoopmaster:54311 (**Εικόνα 3.13**).

```
<configuration>

<property>
  <name>mapred.job.tracker</name>
  <value>Hadoopmaster: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>

</configuration>
```

Εικόνα 3.13: *Η αλλαγή του αριθμού των κόμβων στο αρχείο hdfs-site.xml*

3.7 Εκκίνηση των κόμβων

Εκτελείται ένα format ώστε να ενημερωθεί το τοπικό σύστημα αρχείων στο cluster. Η παραπάνω διαδικασία γίνεται με την hadoop namenode –format (**Εικόνα 3.14**).

```
hduser@Hadoopmaster:~$ hadoop namenode -format
Warning: $HADOOP_HOME is deprecated.

13/10/26 16:43:48 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = Hadoopmaster/127.0.1.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 1.2.1
STARTUP_MSG:   build = https://svn.apache.org/repos/asf/hadoop/common/branches/b
ranch-1.2 -r 1503152; compiled by 'mattf' on Mon Jul 22 15:23:09 PDT 2013
STARTUP_MSG:   java = 1.7.0_25
************************************************************/
Re-format filesystem in /data/tmp/dfs/name ? (Y or N) Y
13/10/26 16:43:51 INFO util.GSet: Computing capacity for map BlocksMap
13/10/26 16:43:51 INFO util.GSet: VM type       = 64-bit
13/10/26 16:43:51 INFO util.GSet: 2.0% max memory = 932118528
13/10/26 16:43:51 INFO util.GSet: capacity      = 2^21 = 2097152 entries
13/10/26 16:43:51 INFO util.GSet: recommended=2097152, actual=2097152
13/10/26 16:43:51 INFO namenode.FSNamesystem: fsOwner=hduser
13/10/26 16:43:51 INFO namenode.FSNamesystem: supergroup=supergroup
13/10/26 16:43:51 INFO namenode.FSNamesystem: isPermissionEnabled=true
13/10/26 16:43:51 INFO namenode.FSNamesystem: dfs.block.invalidate.limit=100
13/10/26 16:43:51 INFO namenode.FSNamesystem: isAccessTokenEnabled=false accessK
```

Εικόνα 3.14: *Υλοποίηση Format στο Namenode*

Για την εκκίνηση των HDFS daemons γίνεται αρχικά η εκκίνηση του NameNode, του Hadoopmaster και του DataNode και μετά σε όλους τους υπόλοιπους κόμβους.

Στη συνέχεια, θα θέσουμε σε λειτουργία τους MapReduce daemons. Αρχικά γίνεται η εκκίνηση στο JobTracker, έπειτα στο Hadoopmaster και μετά στον TaskTracker daemon και ακολούθως σε όλους τους υπόλοιπους κόμβους.

Για την εκκίνηση των HDFS κόμβων χρησιμοποιούμε την εντολή start-dfs.sh. Ελέγχεται ο τρόπος που “τρέχουν“ όλες οι διαδικασίες Java, σε Master και σε Slaves, με την εντολή jps (Εικόνα 3.15).

Εικόνα 3.15: *Οι διαδικασίες Java για το HDFS*

Σταματάμε τους daemons, του mapreduce και του hdfs. Το σύστημα είναι έτοιμο με δύο κόμβους.

stop-mapred.sh (map reduce)

stop-dfs.sh (hdfs)

Καθώς έχει στηθεί το σύστημα με ένα και με δύο κόμβους Master- Slave, δύναται να «τρέξει» το σύστημα με Hadoop (εκτενή αναφορά θα γίνει στο επόμενο κεφάλαιο)

Η εντολή – με την οποία θα “τρέξουμε” τα αρχεία.jar σε περιβάλλον Hadoop – καταγράφεται παρακάτω :

hadoop jar hadoop όνομα αρχείου.jar όνομα project / διαδρομή φακέλου για Input /διαδρομή φακέλου για output

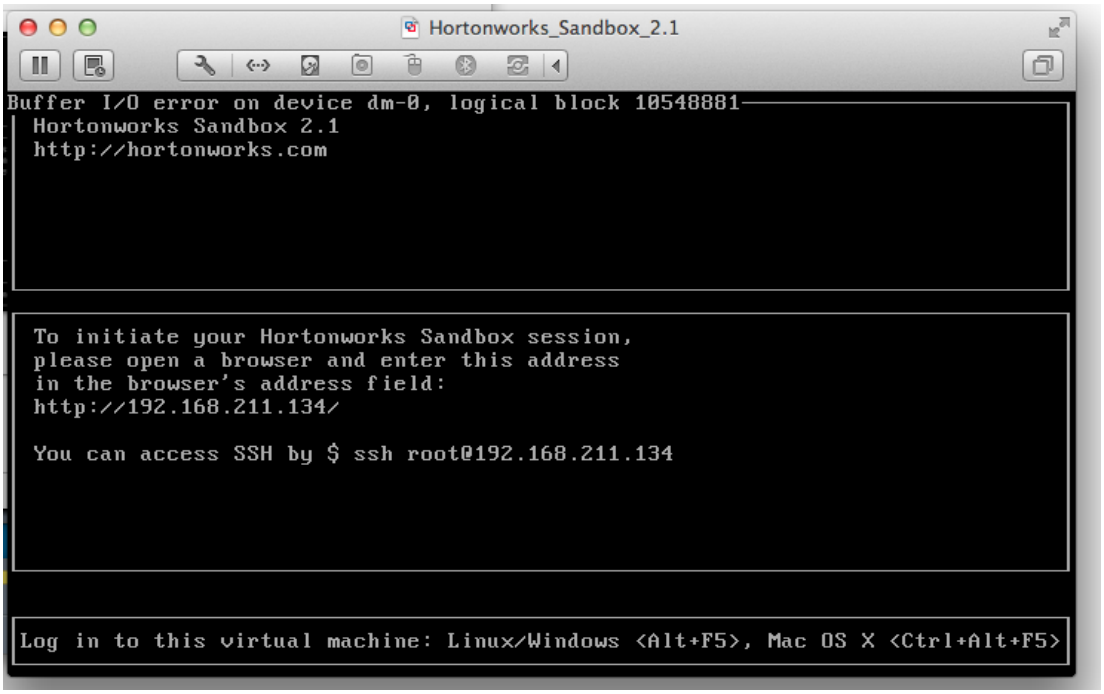
Να επισημανθεί ότι η παρακολούθηση των αρχείων μπορεί να γίνει και μέσω κάποιου Browser (σε υπολογιστή, συνδεδεμένο σε δίκτυο) στα παρακάτω URL:

- http://Hadoopmaster:50070/ – το interface του NameNode daemon
- http://Hadoopmaster:50030/ – το interface του JobTracker daemon
- http://Hadoopmaster:50060/ – το Interface του TaskTracker daemon

4. Υλοποίηση Συστήματος

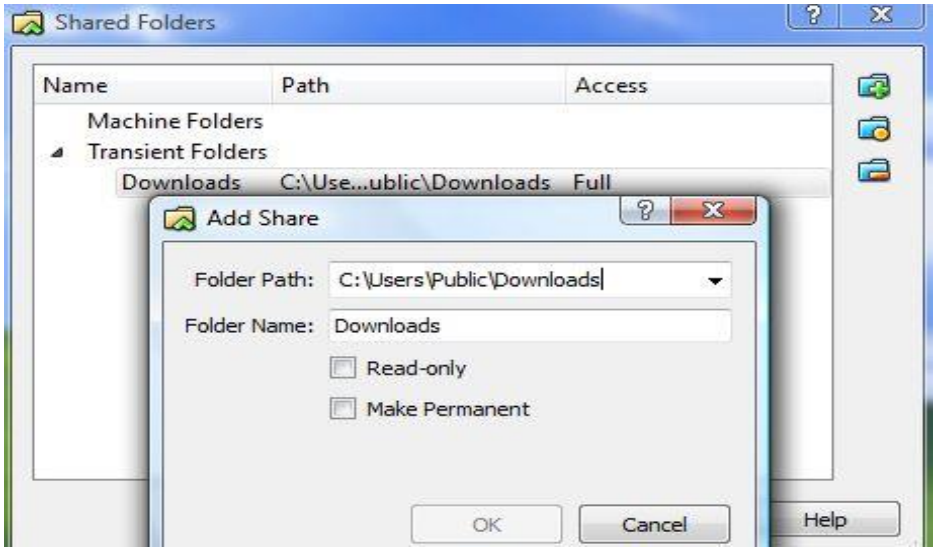
4.1 Horton Hadoop

Λαμβάνοντας υπόψη ότι η υλοποίηση του συστήματος έγινε στο Horton Hadoop, παρέχεται η δυνατότητα να «τρέξουμε» την εφαρμογή στο Cent OS. Το σύστημα επιτρέπει να μπούμε σε σύστημα Linux, μέσω του Alt + F5 (**Εικόνα 4.1)**. Επιπλέον, τονίζεται η υπάρχουσα δυνατότητα να δούμε τις λειτουργίες του Hortonworks Sandbox Session, μέσω μιας διεύθυνσης IP (η οποία τροποποιείται ανά υπολογιστή με χρήση του Horton Hadoop).



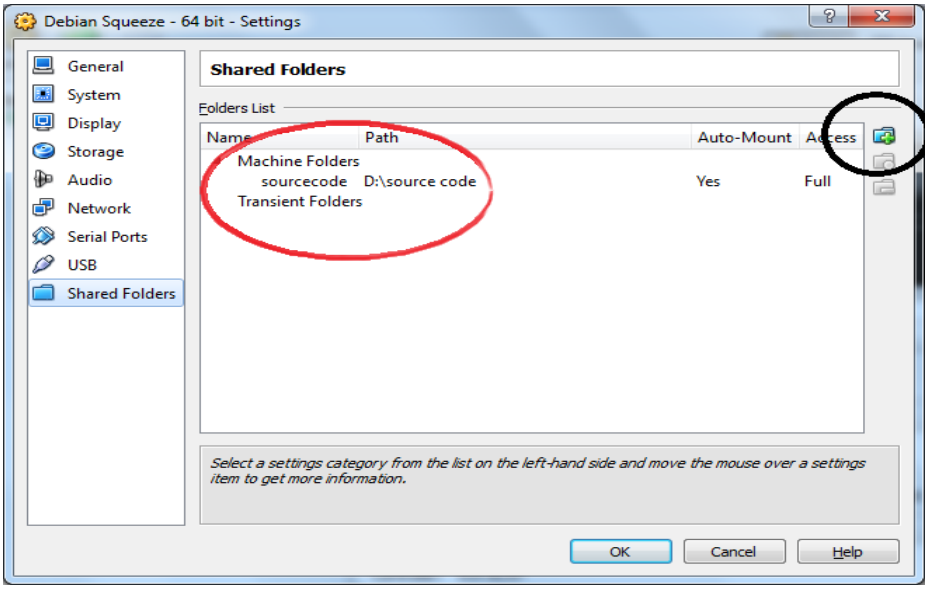
Εικόνα 4.1: *Η εισαγωγική εικόνα του Hortonworks Hadoop*

Δημιουργούμε shared folders μέσω του VirtualBox, ούτως ώστε να είναι εφικτή η αποστολή και η λήψη αρχείων από το λειτουργικό σύστημα, στο οποίο έχουμε κάνει εγκατάσταση το VirtualBox (**Εικόνα 4.2)**.



***Εικόνα 4.2**: Το παράθυρο με τα Shared folders, παραμετροποίηση*

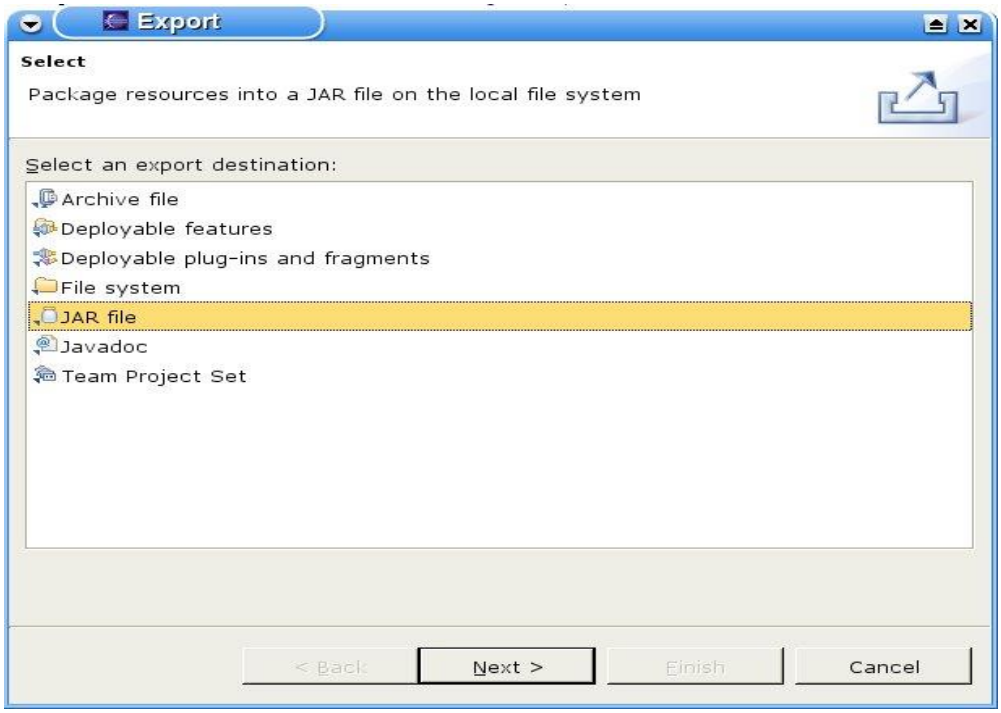
Με την ολοκλήρωση της διαδικασίας, περιγράφεται αναλυτικά το τι έχουμε δημιουργήσει μέσα στον φάκελο του λειτουργικού συστήματος (**Εικόνα 4.3)**. Κατά αναλογία, με την εισαγωγή του add share, τα αρχεία μπορεί να εντοπιστούν και σε μονοπάτι του Cent OS [23].



***Εικόνα 4.3**: Προσθήκη φακέλων από το περιβάλλον του λειτουργικού συστήματος στο VM*

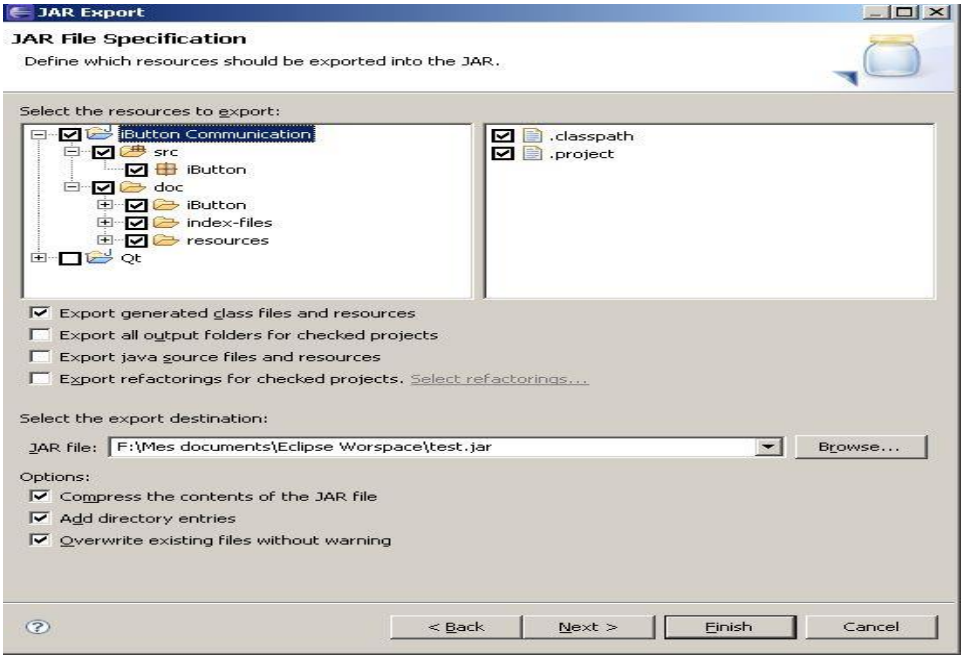
4.2 Δημιουργία Jar Αρχείου

Πριν την έναρξη της διαδικασίας, δημιουργήθηκε το Jar αρχείο για το συγκεκριμένο project (**Εικόνα 4.4)**. Μέσω του File -> Export, στο Eclipse, γίνεται η επιλογή και η εξαγωγή όλων των αρχείων του project σε μορφή .jar.



Εικόνα 4.4: *Η επιλογή της μορφής του αρχείου jar*

Η παραμετροποίηση του jar αρχείου παράλληλα, με την επιλογή των αρχείων είναι το επόμενο βήμα για την δημιουργία του jar αρχείου (**Εικόνα 4.5)**.



Εικόνα 4.5: *Η τελική παραμετροποίηση της διαδικασίας δημιουργίας του αρχείου Jar*

4.3 Διαδικασία Εκτέλεσης προγράμματος

Η εντολή εκτέλεσης του προγράμματος, η οποία εκτελείται στο Cent OS του Hortonworks Hadoop, είναι η εξής :

hadoop jar hadoopPlaces.jar hadoop.geo.AggregateJob latest-greece.osm.bz2 result

όπου α) placesGreece.xml είναι το path του xml αρχείου, β) hadoopPlaces.jar είναι το jar αρχείο που εκτελείται, γ) το αρχείο εισόδου είναι το latest-greece.osm.bz2 και δ) το result είναι το αρχείο, όπου θα δημιουργηθεί το output αρχείο με τα αποτελέσματα.

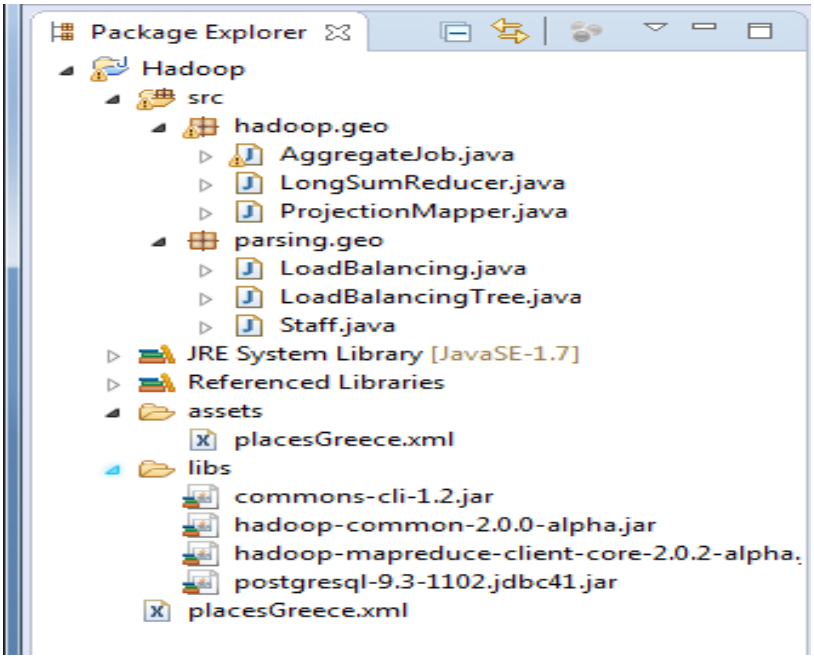
4.4 Διαδικασία Εκτέλεσης (Walkthrough)

Το πρώτο αρχείο προς εκτέλεση είναι το AggregateJob.java όπου βρίσκεται και η main. Δημιουργείται ένα νέο instance της κλάσης Staff (Staff.java) και χρησιμοποιούνται οι 3 μέθοδοι τους οποίους περιέχει (**Εικόνα 4.6)**:

Αρχικά, καλείται η μέθοδος parseXmlFile, η οποία παρσάρει το XML αρχείο. Σε κάθε loop – κατά τη ανάγνωσή του το XML – τοποθετούνται τα αποτελέσματα στην database με την μέθοδο insertToDatabase.

Όταν ολοκληρωθεί το parsing, καλείται η μέθοδος printSQLResults για την εκτύπωση των αποτελεσμάτων της database καθώς και για την καταχώρησή τους σε ένα αρχείο (το result.tsv). Τα αποτελέσματα θα αξιοποιηθούν από το Hadoop.

Αμέσως μετά, θα κληθεί η run μέθοδος της κλάσης AggregateJob η οποία καλεί όλες τις λειτουργίες του Hadoop. Ο κώδικας, ο οποίος την καλεί, είναι ο int rc = ToolRunner.run (new AggregateJob(), args). Σε αυτό το σημείο, θα γίνει launch ένα new job (managed by the Job class from theorg.apache.hadoop.mapreduce package). Οι Mapper (ProjectionMapper.class) και Reducer (LongSumReducer.class) κλάσεις καλούνται.

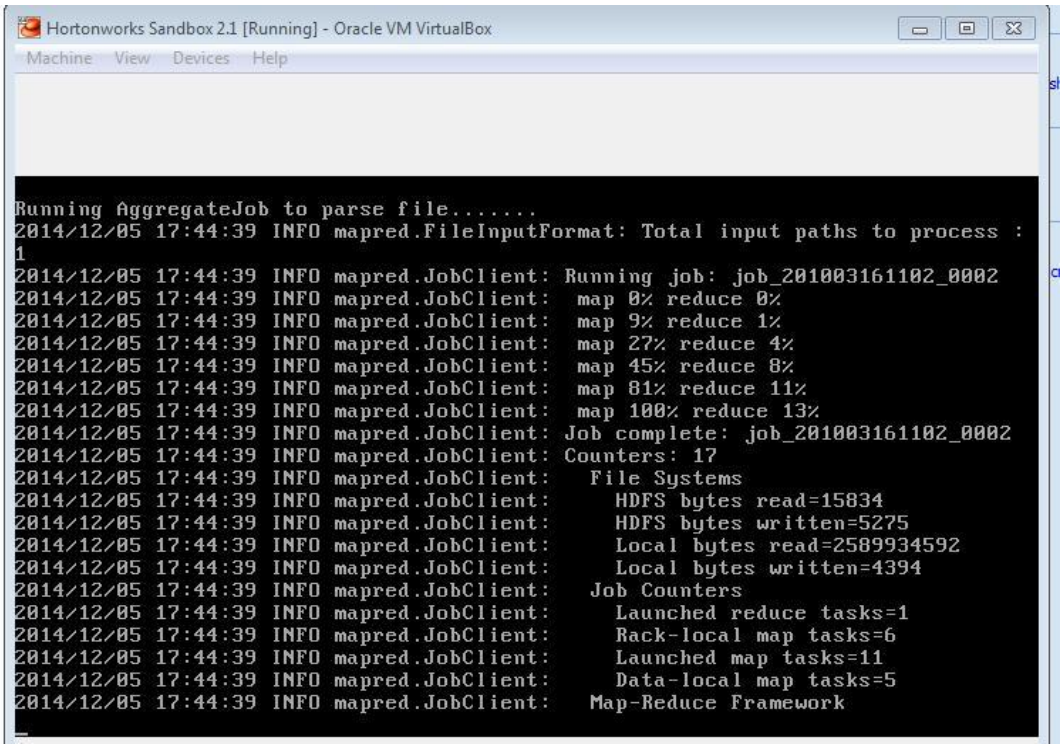


Εικόνα 4.6: *Τα αρχεία που εκτελούνται και οι υπόλοιπες πηγές του project.*

Ο Mapper διαβάζει το result.tsv. Αρχικά, υπάρχουν 2 instance variables α) τα word και β) τα count, τα οποία χρησιμοποιούνται για αποθήκευση του map output key και value. Η μέθοδος map() καλείται μια φορά για κάθε input record, ώστε να αποφευχθεί δημιουργία αντικειμένου (object creation) χωρίς αιτία. Η λειτουργία του map είναι απλή. Πιο συγκεκριμένα, η map α) κάνει split τα tab-separated input line into field, β) χρησιμοποιεί το όνομα (name) σαν λέξη (word) και το ASCII αριθμό του πρώτου γράμματος του word σαν count. Το map output αναγράφεται με την χρήση της write method του Context.

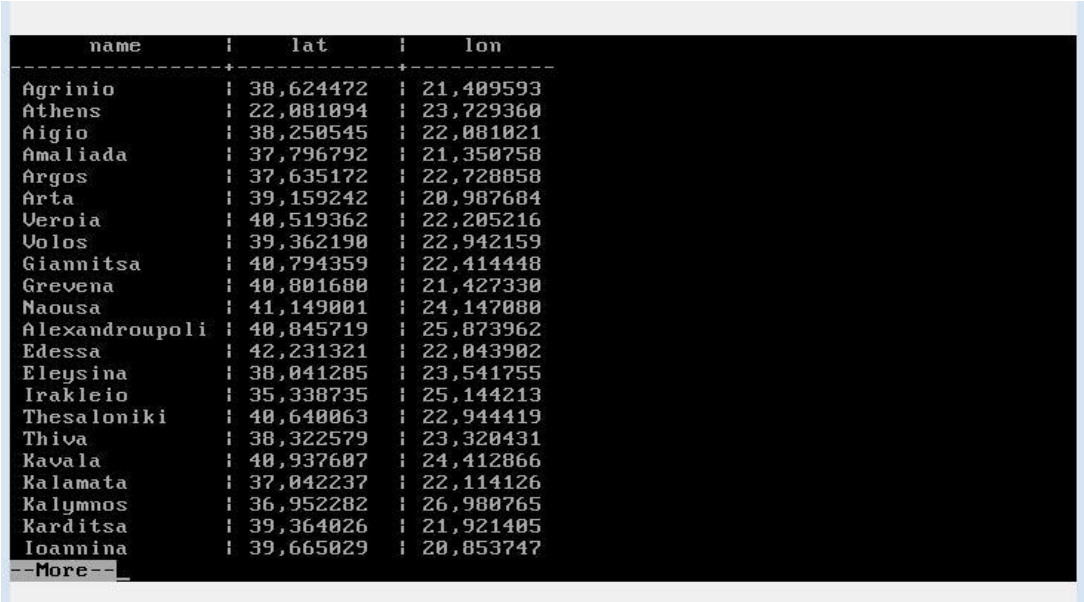
Η LongSumReducer εντοπίζει το άθροισμα των values. Καλεί την LoadBalancingTree (LoadBalancingTree.java), όπου μου τοποθετεί τα values στο δυαδικό δέντρο (binary tree) χρησιμοποιώντας την insert() μέθοδο. Όταν ολοκληρωθούν όλες οι εισαγωγές, εκτυπώνονται τα αποτελέσματα σε ordered μορφή με την χρήση της inorderBinaryTreeForReducer(). Έπειτα, αναγράφεται το αποτέλεσμα sum, το οποίο υπολογίστηκε κατά τις εισαγωγές στο binary tree, χρησιμοποιώντας το ίδιο key as the input.

Η τελευταία γραμμή της run () μεθόδου που βρίσκεται στο AggregateJob.class, κάνει launch το job και αναμένει να ολοκληρωθεί. Όσο τρέχει, εκτυπώνει το progress στην κονσόλα. Το σύστημα ζητά την εντολή να τρέξει το σύστημα hadoop το εκτελέσιμο jar. Η εντολή Hadoop jar HadoopFinal.jar hadoop.geo.AggregateJob greece-latest.osm.bz2 results ξεκινάει το Mapreduce. Με αυτό τον τρόπο, το σύστημα αρχίζει να εντοπίζει τα ζητούμενα ζευγάρια εγγραφών (name, lat, lon) μέσα στο osm.bz2 και να πραγματοποιεί το mapping-reducing (**Εικόνα 4.7**).



Εικόνα 4.7: *Το αρχείο osm.bz2 γίνεται parsing, και ξεκινά η διαδικασία mapreduce*

Η διαδικασία του mapreduce ολοκληρώνεται. Γίνεται αποθήκευση μέσα στην βάση δεδομένων Postgres, του πίνακα Places των τιμών Name, lat και long (**Εικόνα 4.8**).

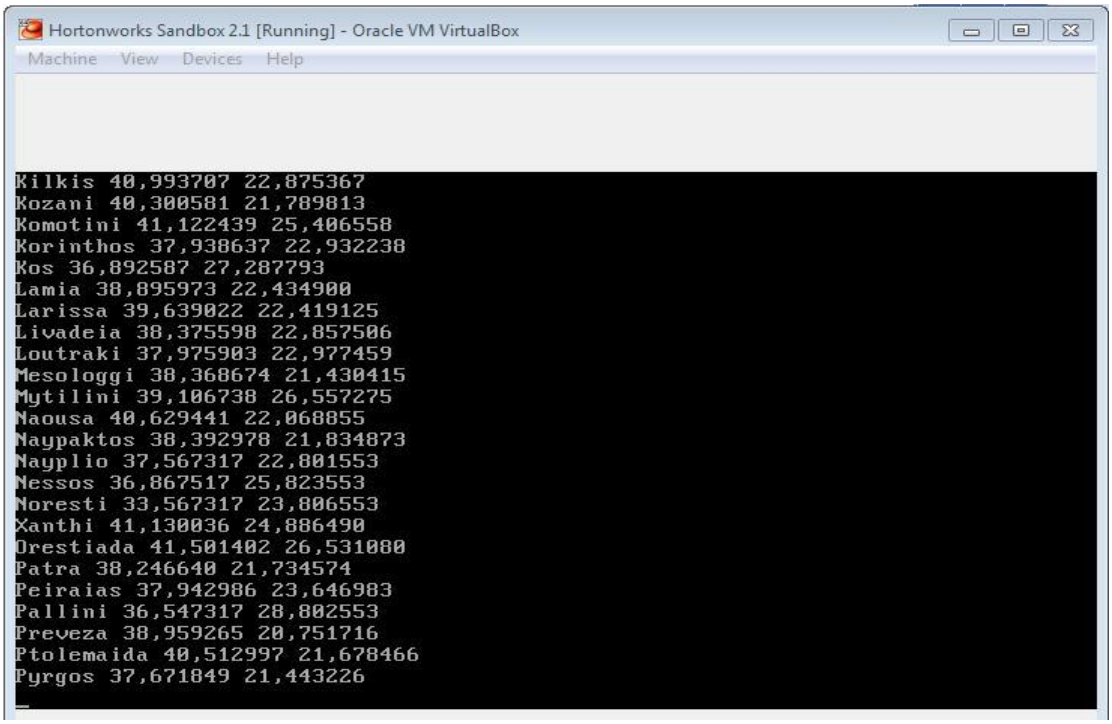


name	lat	lon
Agrinio	38,624472	21,409593
Athens	22,081094	23,729360
Aigio	38,250545	22,081021
Amaliada	37,796792	21,350758
Argos	37,635172	22,728858
Arta	39,159242	20,987684
Veroia	40,519362	22,205216
Volos	39,362190	22,942159
Giannitsa	40,794359	22,414448
Grevena	40,801680	21,427330
Naousa	41,149001	24,147080
Alexandroupoli	40,845719	25,873962
Edessa	42,231321	22,043902
Eleysina	38,041285	23,541755
Irakleio	35,338735	25,144213
Thesaloniki	40,640063	22,944419
Thiva	38,322579	23,320431
Kavala	40,937607	24,412866
Kalamata	37,042237	22,114126
Kalymnos	36,952282	26,980765
Karditsa	39,364026	21,921405
Ioannina	39,665029	20,853747

--More--

Εικόνα 4.8: *Ο πίνακας της βάσης δεδομένων Postgres, places όπου αποθηκεύονται οι τιμές του προγράμματος*

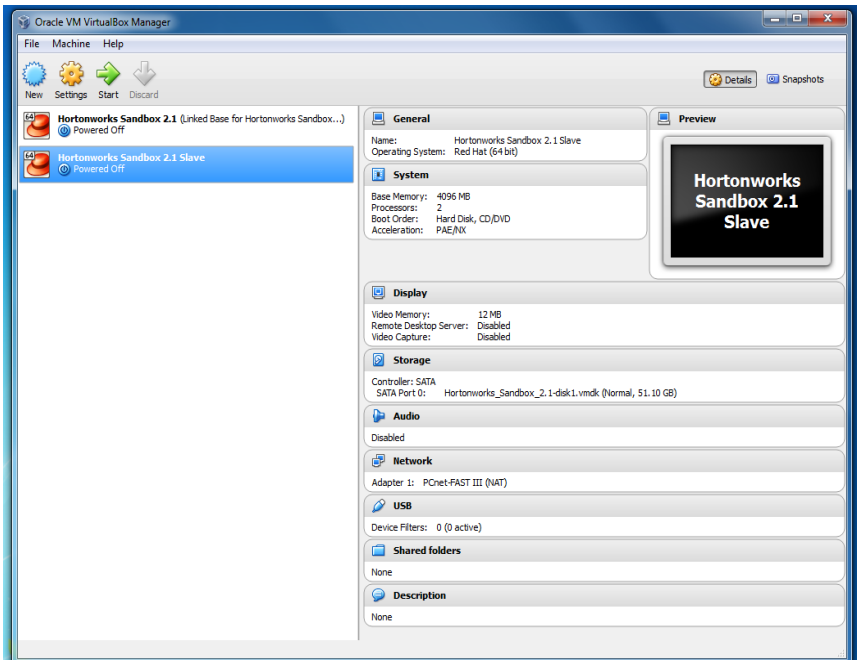
Μετά την αποθήκευση των δεδομένων προχωράμε στην δημιουργία του ισορροπημένου δυαδικού δέντρου. Εδώ πάλι οι κόμβοι δημιουργούνται με το όνομα του latitude και longitude. Το πρόγραμμα, στο τέλος της εκτέλεσης του, εκτυπώνει τους κόμβους του δέντρου τους οποίους εντοπίζει μέσω inorder (**Εικόνα 4.9**).



Εικόνα 4.9: *Εκτύπωση των κόμβων του δέντρου που δημιουργήθηκε, με inorder*

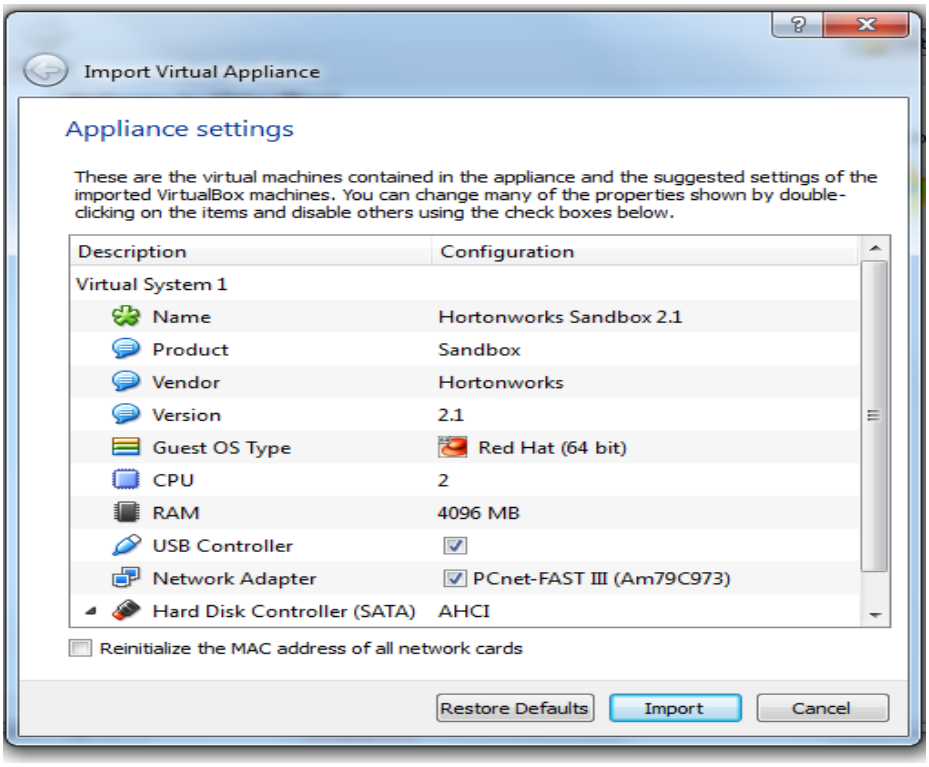
Με αυτό τον τρόπο, ολοκληρώθηκε η χρήση του MapReduce και της αποθήκευσης των δεδομένων και σε βάση δεδομένων Postgres. Επίσης, δημιουργήθηκε το δέντρο εξισορρόπησης. Παρόλο που το εκτυπωμένο δείγμα δεδομένων ήταν μικρό, α) η διαδικασία που ακολουθήθηκε με ένα αρχείο osm.bz2 > 2.5 GB καθώς και β) το άνοιγμα και ο χειρισμός του μέσω του προγράμματος, αναδεικνύουν τις μεγάλες δυνατότητες του Hadoop και του αλγόριθμου που αξιοποιήθηκε. Η ίδια διαδικασία ακολουθήθηκε και με την χρήση Multi-node. Σε αυτή την περίπτωση, τα αποτελέσματα ήταν πραγματικά

καλύτερα. Χρειάστηκε όμως, να τρέχουν ταυτόχρονα 2 VirtualBox με ένα Hortonworks Sandbox το καθένα (**Εικόνα 4.10**). Είναι σημαντικό να τονιστεί ότι απαιτείται ιδιαίτερα δυνατό μηχάνημα για να μπορέσει να τρέξει ένα τέτοιο σύστημα. Και αυτό γιατί το κάθε Hortonworks Hadoop χρειάζεται τουλάχιστον διπύρηνο επεξεργαστή 2 - core CPU, με 4 GB ram (**Εικόνα 4.11**).



Εικόνα 4.10: *Χρήση δύο Virtual Box που τρέχουν Horton Hadoop ως Master / Slave*

Μετά τις διαδικασίες οι οποίες αναφέρθηκαν λεπτομερώς στο κεφάλαιο της σχεδίασης, προέκυψαν δύο διαφορετικοί κόμβοι α) ένας master και β) ένας slave. Ο καθένας έχει την δική του διεύθυνση IP. Εκτελώντας την εφαρμογή στον Master, το Hadoop αντιλαμβάνεται τον επιπλέον κόμβο χρησιμοποιώντας Mappers από εκεί.

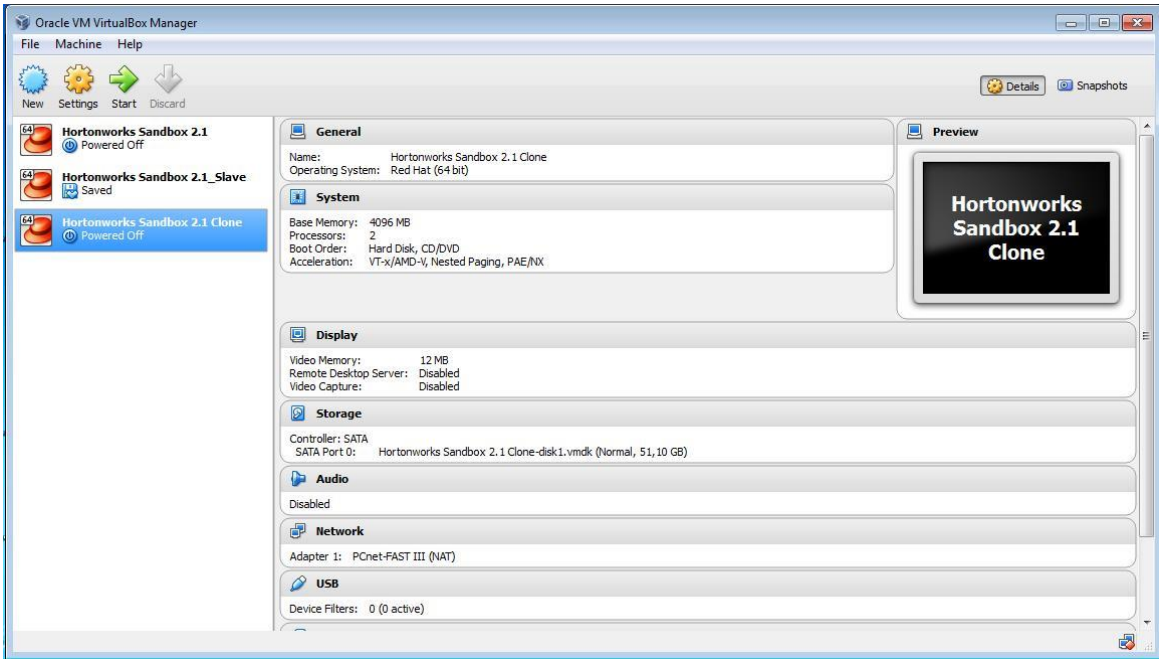


Εικόνα 4.11: *Χαρακτηριστικά Horton Hadoop που επιτρέπουν μόνο την εκτέλεση σε υπολογιστές δυνατότερων χαρακτηριστικών*

Συνεπώς, έχουμε γρηγορότερα αποτελέσματα και γρηγορότερη κατασκευή του εξισορροπημένου δέντρου μας. Δεν μπορούμε να έχουμε συγκεκριμένη εικόνα, μόνο εμπειρική (δηλαδή, μέσα από την παρατήρηση του συστήματος). Σε μελλοντική εργασία θα μπορούσε να προστεθεί κώδικας, ο οποίος θα μετρούσε την ολοκλήρωση του αλγόριθμου μέσα σε συγκεκριμένα χρονικά αποτελέσματα.

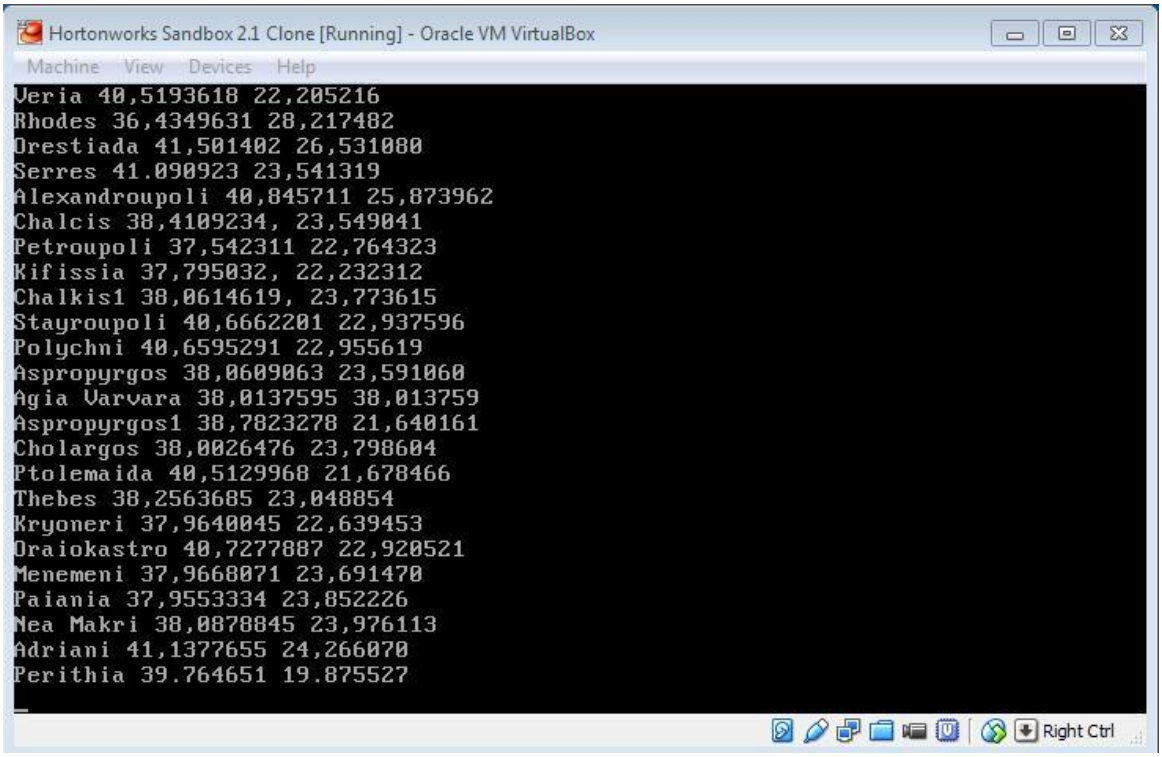
4.5 Τελικό Στάδιο εργασίας - Αποτελέσματα

Το τελικό στάδιο της εργασίας επέτρεπε στον χρήστη να δώσει τον αριθμό των ζευγαριών Mappers, τα οποία θα έπαιρνε το Hadoop MapReduce. Και αυτό ακούγεται αρχικά, ιδιαίτερα δύσκολο. Όμως, έγινε εφικτό, καθώς μέσα στον κώδικα Java, στην μέθοδο του parsing του αρχείου, μπορέσαμε να βάλουμε τις κατάλληλες εντολές if, οι οποίες θα περιόριζαν την ανάγνωση του αρχείου από τον parser έως το σημείο επιλογής του χρήστη (βλ. εικόνα 4.12).



***Εικόνα 4.12*: Δημιουργία κλώνου του Horton Hadoop, για να ολοκληρωθεί η διαδικασία χωρίς περιορισμούς στο δείγμα**

Με την χρήση μεταβλητών καταφέραμε να ζητήσουμε έναν αριθμό του χρήστη και βάσει αυτού του αριθμού να ορίσουμε τον αριθμό των αποτελεσμάτων. Μετά από κάποιο σημείο, όμως ο υπολογιστής ξεκίνησε να φτάνει στο τέρμα της υπολογιστικής του ισχύος, αναδεικνύοντας την αυξημένη επεξεργαστική ισχύ που χρησιμοποιούσε. Έτσι, η διαδικασία δεν ολοκληρώθηκε με επιτυχία (βλ. Εικόνα 4.13).



Εικόνα 4.13: Τα αποτελέσματα κάποια στιγμή σταματάνε αφού το σύστημα φτάνει σχεδόν σε τέλμα απόδοσης

Παρόλα αυτά, η εκτέλεση του προγράμματος επέτρεψε την διαδικασία να προχωρήσει χωρίς περιορισμό. Κατ' επέκταση, επετεύχθη ελεύθερη χρήση του MapReducer. Με αυτό τον τρόπο, μπορέσαμε να δούμε την χρήση του και να αποδειχτούν τα πλεονεκτήματα του (τα οποία και αναφέρθηκαν στην θεωρία).

5. Συμπεράσματα

Το Hadoop είναι η εξέλιξη των παραδοσιακών Warehouses, όπως αναφέρθηκε και στην εισαγωγή. Αυτό αποδείχθηκε μέσω της υλοποίησης ενός αλγόριθμου, ο οποίος επιτρέπει α) την χρήση του Hadoop και των υποσυστημάτων του, β) την διαδικασία αποθήκευσης των δεδομένων σε βάση δεδομένων και γ) την κατασκευή ενός εξισορροπημένου δυαδικού δέντρου.

Αυτή η διαδικασία ανέδειξε την παραδοσιακή αποθήκευση δεδομένων ενός μεγάλου αρχείου δεδομένων σε μία βάση δεδομένων (εξόρυξη – αποθήκευση). Η διαδικασία εξόρυξης βασίστηκε στο MapReduce. Σε αυτό το σημείο, έγινε εμφανής η ανωτερότητα του συστήματος, καθώς μία διαδικασία που συνήθως πραγματοποιείται σε μεγάλο χρονικό διάστημα εδώ ολοκληρώθηκε σε μερικά λεπτά. Βέβαια, η διαδικασία αυτή είχε και τα μειονεκτήματά της, αφού η απαίτηση σε υπολογιστικό υλικό ήταν πολύ μεγάλη. Επιπλέον, με την χρήση συμβατικών συστημάτων, δεν επετεύχθη η επιτυχής ολοκλήρωση της διαδικασίας. Τέλος, απαιτείται πολύς χρόνος για παραμετροποίηση των συστημάτων, τα οποία φιλοξενούν το Hadoop.

Η παραμετροποίηση ενός συστήματος Linux, για την χρήση Hadoop, δημιούργησε πολλά προβλήματα και έτσι, εγκαταλείφθηκε υποδηλώνοντας την δυσκολία για την εγκατάσταση Hadoop. Την ίδια στιγμή, η χρήση του συστήματος Hadoop μέσω μιας πλατφόρμας όπως είναι το Hortonworks επέτρεψε την ολοκλήρωση της εργασίας και την υλοποίηση του αλγορίθμου. Και σε αυτή την περίπτωση, η εγκατάσταση του τελικού jar αρχείου χρειάστηκε πάλι ιδιαίτερη παραμετροποίηση στο σύστημα.

Σίγουρα τώρα, με την νέα έκδοση του HortonWorks για Windows Server, ανοίγει ένα παράθυρο και για ευκολότερη εγκατάσταση του συστήματος και σε Windows συστήματα. Λαμβάνοντας αυτό υπόψη, τα θέματα μιας δύσκολης παραμετροποίησης θα επιλυθούν.

Καταλήγοντας, δεν μπορεί να ειπωθεί ότι η διαδικασία της διαχείρισης μεγάλου όγκου δεδομένων με το Hadoop είναι πανάκεια στα προβλήματα αυτά. Δεν μπορεί επίσης, να υποτεθεί ότι η χρήση του αντικαθιστά πλήρως όλα τα προηγούμενα συστήματα, τα οποία υπάρχουν αυτή την στιγμή. Σίγουρα όμως, ο συνδυασμός των παραδοσιακών συστημάτων διαχείρισης συνδυαστικά, με την χρήση του Hadoop είναι μία πολύ θετική εξέλιξη της τεχνολογίας της πληροφορίας και με πολλά οφέλη (ειδικά, σε θέματα χρόνου ολοκλήρωσης της διαδικασίας).

6. Μελλοντική Εργασία

Λαμβάνοντας υπόψη α) την εξέλιξη του HortonWorks Hadoop και β) την συμβατότητα του Hadoop και με πλατφόρμες Windows σίγουρα θα είναι εφικτή η ευκολότερη εγκατάσταση του συστήματος σε περισσότερα συστήματα. Ταυτόχρονα, μπορεί να αξιοποιηθεί για την επίλυση πιο δύσκολων και χρονοβόρων αλγορίθμων.

Επιπλέον, σε δίκτυα με την χρήση Windows, θα είναι ευκολότερη η παράλληλη επεξεργασία και η χρήση Multi-node από τα συστήματα. Και το παραπάνω θα γίνεται χωρίς πολλές παραμετροποιήσεις, δίνοντας ακόμη περισσότερες δυνατότητες στο Hadoop MapReduce.

Ολοκληρώνοντας την παρούσα εργασία, θα μπορούσαν να εισαχθούν μετρητές για θέματα χρόνου ώστε να έχουμε μία πλήρη καταγραφή των παραμέτρων χρόνου και εργασιών σε αντιπαράθεση. Αυτή η νέα συνθήκη θα επέτρεπε να κατανοηθεί η διάρκεια του χρόνου που απαιτείται για την ολοκλήρωση της κάθε επιμέρους διεργασίας (παραδείγματος χάρη η αποθήκευση των δεδομένων σε βάση δεδομένων, η εξόρυξη από το αρχείο, η δημιουργία δέντρων).

Βιβλιογραφία

1. Beye Network "Big Data vs Data Warehouses" Διαδικτυακό άρθρο: http://www.b-eye-network.com/view/17017 **[accessed 17/12/2014]**

2. Data Informed " Hadoop vs. Data Warehouse: Comparing Apples and Oranges?" **[online]**: http://data-informed.com/hadoop-vs-data-warehouse-comparing-apples-oranges/ **[accessed 17/12/2014]**

3. **[online],** Teradata White Paper "Hadoop and the Data Warehouse When to Use Which"

Διαδικτυακό άρθρο: http://www.teradata.com/Resources/White-Papers/Hadoop-and-the-Data-Warehouse-When-to-Use-Which/ **[accessed 17/12/2014]**

4. **[online],** TechTarget "Big data and data warehousing: Where's the relationship headed?" Διαδικτυακό Handbook: http://searchdatamanagement.techtarget.com/ehandbook/Big-data-and-data-warehousing-Wheres-the-relationship-headed **[accessed 17/12/2014]**

5. CoreServlets Tutorials, **[online],** "Hadoop Tutorial Developing Big-Data Applications with Apache Hadoop" Διαδικτυακό tutorial: http://www.coreservlets.com/hadoop-tutorial/#Tutorial-Intro **[accessed 17/12/2014]**

6. S Wadkar, M Siddalingaiah, J Venner "Pro Apache Hadoop" Apress 2nd Edition Pages 11 - 20.

7. S Wadkar, M Siddalingaiah, J Venner "Pro Apache Hadoop" Apress 2nd Edition Pages 21 - 31.

8. S Wadkar, M Siddalingaiah, J Venner "Pro Apache Hadoop" Apress 2nd Edition Pages 73 - 107.

9. R Paravastu, R Scarlat, B Chandrasekaran, “Adaptive Load Balancing in MapReduce Using Flubber”, Duke University **[online]** : https://www.cs.duke.edu/courses/fall12/cps216/Project/Project/projects/Adaptive_load_balancer/adaptive-load-balancing.pdf **[accessed 17/12/2014]**

10. PostgreSQL Official Web Site : http://www.postgresql.org/about/ **[accessed 17/12/2014]**

11. PostgresSQL Wiki and installation tutorial **[online],**: http://wiki.gentoo.org/wiki/PostgreSQL/QuickStart **[accessed 17/12/2014]**

12. SourceForge " HadoopDB Quick Start Guide" **[online],**: http://hadoopdb.sourceforge.net/guide/ **[accessed 17/12/2014]**

13. Hortonworks official site : http://hortonworks.com/ **[accessed 17/12/2014]**

14. **[online],** Hortonworks documentation on hadoop : http://hortonworks.com/hdp/docs/ **[accessed 17/12/2014]**

15. Developer blog **[online],** "Getting started with Hadoop using Hortonworks Sandbox" Διαδικτυακό tutorial : https://developer.rackspace.com/blog/getting-started-with-hadoop-using-hortonworks-sandbox/ **[accessed 17/12/2014]**

16. eData Analyst **[online],** " Hands-on Hadoop Tutorial with Hortonworks SandBox VM part 1: the Boss Edition" Διαδικτυακό tutorial: http://datacenter.opentray.com/2013/08/hands-on-hadoop-tutorial-with-hortonworks-sandbox-vm-part-1-the-boss-edition/ **[accessed 17/12/2014]**

17. eData Analyst, **[online],** " Hands-on Hadoop Tutorial with Hortonworks SandBox VM part 1: the Boss Edition" Διαδικτυακό tutorial: http://datacenter.opentray.com/2013/08/hands-on-hadoop-tutorial-with-hortonworks-sandbox-vm-part-2-the-boss-edition/ **[accessed 17/12/2014]**

18. Hortonworks "Hadoop Installation on Windows" **[online]**: http://hortonworks.com/wp-content/uploads/unversioned/pdfs/InstallingHortonworksSandbox2onWindowsusingVB.pdf **[accessed 17/12/2014]**

19. Επίσημη σελίδα geofabrik **[online]** : http://www.geofabrik.de/data/ **[accessed 17/12/2014]**

20. Openstreet Wik osm αρχεία **[online]** : https://wiki.openstreetmap.org/wiki/Planet.osm **[accessed 17/12/2014]**

21. Kipral A. Venkatesh, K Neelamegam : “Using MapReduce and Load Balancing on the Cloud”, IBM Articles, July 2010

22. Online Hadoop Tutorial Creation of Multi Nodes : http://bigdatahandler.com/hadoop-hdfs/hadoop-multi-node-cluster-setup/

23. Virtual Box επίσημη ιστοσελίδα "Shared Folders management" **[online]** : https://www.virtualbox.org/manual/ch04.html **[accessed 17/12/2014]**

Εικόνες

Εικόνα 1.1: Jeff Kelly **[online], «**Big Data: Hadoop, Business Analytics and Beyond», Feb 2014. http://wikibon.org/wiki/v/Big_Data:_Hadoop,_Business_Analytics_and_Beyond **[accessed 17/12/2014]**

Εικόνα 1.2: Jeff Kelly **[online], «**Big Data: Hadoop, Business Analytics and Beyond», Feb 2014. http://wikibon.org/wiki/v/Big_Data:_Hadoop,_Business_Analytics_and_Beyond **[accessed 17/12/2014]**

Εικόνα 2.1: Unknown, **[online],** «Apache Hadoop HDFS Architecture» May 2013. http://www.edureka.co/blog/apache-hadoop-hdfs-architecture/ **[accessed 17/12/2014]**

Εικόνα 2.2: Karishma Surana **[online],** «Hadoop HDFS data read and write operations», Jul 2014. http://nixustechnologies.com/2014/07/hadoop-hdfs-data-read-and-write-operations/ **[accessed 17/12/2014]**

Εικόνα 2.3: Tejas Patil **[online],** http://stackoverflow.com/questions/9363761/how-does-mapreduce-framework-implement-the-sort-phase **[accessed 17/12/2014]**

Εικόνα 2.4 : Yale University HadoopDB Team «HadoopDB Quick Start Guide» 2009 **[online],** http://hadoopdb.sourceforge.net/guide/ **[accessed 17/12/2014]**

Εικόνα 2.5: Unknown, «Alternative Techniques to MapReduce», **[online],** http://largescaledataanalysis.wikispaces.asu.edu/Alternative+Techniques+to+MapReduce **[accessed 17/12/2014]**

Εικόνα 2.6 : Apache Software Foundation, «Introducing Apache Hadoop to Developers» **[online],** http://hortonworks.com/hadoop-tutorial/introducing-apache-hadoop-developers/ **[accessed 17/12/2014]**

Εικόνα 2.7: Apache Software Foundation, «Introducing Apache Hadoop to Developers» **[online],** http://hortonworks.com/hadoop-tutorial/introducing-apache-hadoop-developers/ **[accessed 17/12/2014]**

Εικόνα 3.1**:** Unknown, «Local_Install_HDPSandboxWebInterface» **[online], Apr 2014.**http://bicortex.com/bicortex/wp-content/post_content//2014/04/HDP_Local_Install_HDPSandboxWebInterface.png **[accessed 17/12/2014]**

Εικόνα 3.7 : «Big Data Handler», **[online],** http://i1.wp.com/bigdatahandler.com/wp-content/uploads/2013/10/m1.png?resize=281%2C292 **[accessed 17/12/2014]**

Εικόνα Εξωφύλλου: Kevin T. Smith **[online],** «Big Data Security: The Evolution of Hadoop's Security Model» **[accessed 17/12/2014]**

ΠΑΡΑΡΤΗΜΑ

Η κλάση **AggregateJob :**

```
package hadoop.geo;

//import ProjectionMapper;

import java.util.Scanner;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

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

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

import org.apache.hadoop.mapreduce.lib.reduce.LongSumReducer;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;


import parsing.geo.Staff;


public class AggregateJob extends Configured implements Tool {


        // Before we run our job, we need some driver code to wire up the mapper and

        // reducer, which we do using the AggregateJob class, shown Listing Three

        // (AggregateJob.java).
```

```
@Override
public int run(String[] args) throws Exception {

        // The Job instance specifies various things about the job: a name for
        // display purposes, the mapper and reducer classes,
        //  and the job output types, which have to match the mapper and
        // reducer output types
        Job job = new Job(getConf());

        // The call to setJarByClass() is needed because MapReduce is a
        // distributed system, and Hadoop needs to know which JAR file to ship
        // to the nodes in the cluster running the map and reduce tasks.
        job.setJarByClass(getClass());

        job.setJobName(getClass().getSimpleName());

        // We also need to tell the job what input data to process and where to
        // place the output. We do this via the static APIs on FileInputFormat
        // and FileOutputFormat, using positional command-line arguments to
        // specify the file paths.
        FileInputFormat.addInputPath(job, new Path(args[2]));
        FileOutputFormat.setOutputPath(job, new Path("/asset/"+args[3]));

        job.setMapperClass(ProjectionMapper.class);
        job.setCombinerClass(LongSumReducer.class);
        job.setReducerClass(LongSumReducer.class);
```

```
		job.setOutputKeyClass(Text.class);
		job.setOutputValueClass(LongWritable.class);

		//here method launches the job and waits for it to complete.
		return job.waitForCompletion(true) ? 0 : 1;
	}

	public static String fPlaceFile;

	public static void main(String[] args) throws Exception {
		String Line;
		Scanner input = new Scanner(System.in);
		do {
			System.out.println("Please insert Hadoop jar command : The format should be: \n hadoop jar nameofjar file name of class inputfile outputfile");
			Line = input.nextLine();
		if(Line.compareTo("hadoop jar HadoopFinal.jar hadoop.geo.AggregateJob greece-latest.osm.bz2 results") ==0 ) {
			System.out.println("Jar Command Correct, please press enter to start Hadoop MapReduce Process");
			input.nextLine();
			break;
		}
		else {
			System.out.println("Wrong input format please try again!!");
		}
```

```
                } while(true);
                System.out.println("Running AggregateJob to parse file.....");
                // fPlaceFile = "assets/"+args[0]; //e.g. placesGreece.xml
                fPlaceFile = Line;
                // Do all the transactions here
                Staff goBinary = new Staff();
                goBinary.parseXmlFile();

                // goBinary.printSQLResults();
                // Hadoop
                int rc = ToolRunner.run(new AggregateJob(), args);
                System.exit(rc);
        }
}
```

Η κλάση **LongSumReducer :**

```
//The packets required from hadoop
package hadoop.geo;
import java.io.IOException;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Reducer;

import parsing.geo.LoadBalancingTree;
```

```
public class LongSumReducer<KEY> extends Reducer<KEY, LongWritable, KEY, LongWritable>
{

	private LongWritable result = new LongWritable();


	public void reduce(KEY key, Iterable<LongWritable> values, Context context)
				throws IOException, InterruptedException {
		long sum = 0;
		for (LongWritable val : values) {
				// load balancing Singleton
				// Pattern is used for load balancing tree
				LoadBalancingTree.getInstance().insert(val.get());


				//Calculating the number of couples to used by mappers
				sum += val.get();
		}


		//Printing the nodes in order here.
		LoadBalancingTree.getInstance().inorderBinaryTreeForReducer();


		//The sum result regarding the output file
		result.set(sum);
		context.write(key, result);
	}


}
```

Η κλάση **ProjectionMapper :**

```
//The hadoop and java import packages

package hadoop.geo;


import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;


public class ProjectionMapper extends

                Mapper<LongWritable, Text, Text, LongWritable> {

        private Text word = new Text();

        private LongWritable count = new LongWritable();


        @Override

        protected void map(LongWritable key, Text value, Context context)

                        throws IOException, InterruptedException {


                // we split the word before inserting to the sql                        //database

                //Split with any white space chars

                String[] split = value.toString().split("\\s+");//

                word.set(split[0]);

                if (split.length >= 1) {

                        try {

// setting the count according to the ASCII of the first char of //the word for each city in the file
```

```
                                    count.set((long)(split[0].charAt(0)));                                    //
count.set((long)(word.charAt(0)));
                                    context.write(word, count);
                            } catch (NumberFormatException e) {
                                    // if it cannot parse it will be caught in //exception
                            }
                    }
            }
}
```

Η κλάση **LoadBalancing :**

```
package parsing.geo;

//Load Balancing of the tree class
public class LoadBalancing {

        public LoadBalancing left, right;

        private String name;
        private String lat;
        private String lon;

        private long value;

//constructors of the tree...
```

```
public LoadBalancing(long value) {
        this.value = value;
        this.left = null;
        this.right = null;
}

public long getValue()
{
        return value;
}

public LoadBalancing(String name, String lat, String lon) {
        this.name = name;
        this.lat = lat;
        this.lon = lon;
        this.left = null;
        this.right = null;
}
//set - get values
public String getName() {
        return name;
}

public String getLat() {
        return lat;
}
```

```
	public String getLon() {
		return lon;
	}

}
```

Η κλάση **LoadBalancingTree :**

```
package parsing.geo;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;

public class LoadBalancingTree {
   LoadBalancing root;

   private static volatile LoadBalancingTree instance = null;

	public static LoadBalancingTree getInstance() {
		if (instance == null) {
			synchronized (LoadBalancingTree.class) {
				// Double check
```

```
                                    if (instance == null) {
                                            instance = new LoadBalancingTree();
                                    }
                            }
                    }
                    return instance;
            }


    public LoadBalancingTree()
    {
        root = null;
    }


    //check if the position is empty and has the Singleton
   //Requirements to be created.
   protected LoadBalancing insertNodebyLongValue(LoadBalancing NodeR, LoadBalancing NewNode) {
        if (NodeR == null)
        {
            NodeR = NewNode;
        }
        else if (NewNode.getValue() < NodeR.getValue())
        {
            NodeR.left = insertNodebyLongValue(NodeR.left, NewNode);
        }
        else
```

```
      {
          NodeR.right = insertNodebyLongValue(NodeR.right, NewNode);
      }
      return NodeR;
  }
   //insert values
  public void insert(long value)
  {
              LoadBalancing Node = new LoadBalancing(value);
     root = insertNodebyLongValue(root,Node);
  }


  protected LoadBalancing insertNode(LoadBalancing NodeR, LoadBalancing NewNode) {
     if (NodeR == null)
     {
        NodeR = NewNode;
     }  //Load balancing takes place here by comparing                    //the    different    binary
nodes
     else if (((NewNode.getName()).compareTo(NodeR.getName()))  < 0)
     {
        NodeR.left = insertNode(NodeR.left, NewNode);
     }
     else
     {
        NodeR.right = insertNode(NodeR.right, NewNode);
     }
```

```
    return NodeR;
}

public void insert(String name,String lat,String lon)
{
            LoadBalancing Node = new LoadBalancing(name, lat, lon);
    root = insertNode(root,Node);
}

          /**
           * Parsing the value here and writing the result to the .tsv file. That file
           * will later be parsed by Hadoop.
           */
protected void inorder(LoadBalancing NodeR) {
     if (NodeR != null) {
         inorder(NodeR.left);
         mWriter.println(NodeR.getName()+"\t"+NodeR.getLat()+"\t"+NodeR.getLon());
         inorder(NodeR.right);
     }
 }

          /**
           * here the results.tsv file is created
           * that will write the results there
           */
 private PrintWriter mWriter;
```

```
 public void inorderBinaryTree() {
//Creation of file
        File file = new File("assets/result.tsv");
                if(file.exists())
                {
                        file.delete();
                }

                try {
                        mWriter = new PrintWriter("assets/result.tsv", "UTF-8");
                        inorder(root);
                        mWriter.close();
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                } catch (UnsupportedEncodingException e) {
                        e.printStackTrace();
                }

 }

 protected void inorderForReducer(LoadBalancing NodeR) {
    if (NodeR != null) {
         inorderForReducer(NodeR.left);
        System.out.println(NodeR.getValue());
        inorderForReducer(NodeR.right);
    }
```

```
    }

        public void inorderBinaryTreeForReducer() {

                inorderForReducer(root);

        }

}
```

Η κλάση **Staff :**

```
package parsing.geo;


import hadoop.geo.AggregateJob;
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
//database connection
public class Staff {

    public void insertToDatabase(String name,String lat,String lon) {

      Connection connection = null ;
      try {
           Class.forName("org.postgresql.Driver");
           //Database url connection
          connection = DriverManager.getConnection("jdbc:postgresql://localhost:5432/postgres","postgres", "123456");
           connection = DriverManager.getConnection("jdbc:postgresql://localhost:5432/postgres","Manos", "123456");
```

```
        System.out.println("Opened database successfully");
//insert into places table statement
        String insertCommand = "INSERT INTO places VALUES(?,?,?)";
        PreparedStatement pst = connection.prepareStatement(insertCommand);


        pst.setString(1,name);
        pst.setString(2,lat);
        pst.setString(3,lon);


        pst.executeUpdate();


     } catch ( Exception e ) {
      System.err.println( e.getClass().getName()+": "+ e.getMessage() );


     }
    finally
    {
          try {
                    connection.close();
              } catch (SQLException e) {
                    e.printStackTrace();
              }
    }
```

```
      System.out.println("Table created successfully");


   }


    public void printSQLResults() {
      String name, lat, lon;
      Connection connection = null;
      Statement stmt = null;
      try {
          Class.forName("org.postgresql.Driver");


          connection                                                    =
DriverManager.getConnection("jdbc:postgresql://localhost:5432/postgres","Mano
s", "123456");
          connection.setAutoCommit(false);
          System.out.println("Opened database successfully");


          stmt = connection.createStatement();
//Retrieving data and inserting them into the Tree and then into //the file
          ResultSet rs = stmt.executeQuery( "SELECT * FROM places ;" );


          //new code
           PrintWriter mWriter;
```

```
mWriter = new PrintWriter("assets/result.tsv", "UTF-8");


while ( rs.next() ) {
    name = rs.getString("name");
    lon = rs.getString("long");
    lat = rs.getString("lat");


                //Printing DB values.
            System.out.println("name = " + name);
                System.out.println("lon = " + lon);
                System.out.println("lat = " + lat);
                System.out.println();


    //writing values to the file which will be read by Hadoop later.
    mWriter.println(name+"\t"+lon+"\t"+lon);


    //According to the article regarding load balancing Singleton Pattern is used for load balancing tree
    //LoadBalancingTree.getInstance().insert(name,lat,lon);
}


//new code
                mWriter.close();
```

```
        rs.close();
        stmt.close();
        connection.close();

        } catch ( Exception e ) {
              System.err.println( e.getClass().getName()+": "+ e.getMessage() );

        }

        System.out.println("Job Finished Succesfully");

    }

    public void parseXmlFile() throws ParserConfigurationException,SAXException, IOException {
                String name,lat,lon;

                DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
                DocumentBuilder builder = factory.newDocumentBuilder();

                Document document = builder.parse(new File(AggregateJob.fPlaceFile));
```

```
            NodeList nodeList = document.getDocumentElement().getChildNodes();
            for (int i = 0; i < nodeList.getLength(); i++) {
                Node node = nodeList.item(i);

                if (node.getNodeType() == Node.ELEMENT_NODE) {
                    Element elem = (Element) node;
                    name = elem.getElementsByTagName("name").item(0).getChildNodes().item(0).getNodeValue();
                    lat = elem.getElementsByTagName("lat").item(0).getChildNodes().item(0).getNodeValue();
                    lon = elem.getElementsByTagName("long").item(0).getChildNodes().item(0).getNodeValue();

                    insertToDatabase(name, lat, lon);
                }
            }
    }

}
```