Πανεπιστημιο Θεσσαλιας
Πολυτεχνικη Σχολη
Τμημα Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Υλοποίηση Interpolation Search Tree (IST) σε NoSQL Συστήματα Βάσεων Δεδομένων

Μεταπτυχιακη Εργασια

του

ΓΕΩΡΓΙΟΥ ΣΗΚΩΤΙΔΗ

Επιβλέπων: Βασιλακόπουλος Μιχαήλ
Αναπληρωτής Καθηγητής

Βόλος, Ιούνιος 2018

Πανεπιστήμιο Θεσσαλίας
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Υλοποίηση Interpolation Search Tree (IST) σε NoSQL Συστήματα Βάσεων Δεδομένων

Μεταπτυχιακη Εργασια

του

ΓΕΩΡΓΙΟΥ ΣΗΚΩΤΙΔΗ

Επιβλέπων: Βασιλακόπουλος Μιχαήλ
Αναπληρωτής Καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή στις ..... Ιουνίου 2018.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Βασιλακόπουλος Μιχαήλ	Τσομπανοπούλου Παναγιώτα	Δασκαλοπούλου Ασπασία
Αναπληρωτής Καθηγητής	Αναπληρώτρια Καθηγήτρια	Επίκουρη Καθηγήτρια

Βόλος, Ιούνιος 2018

Πανεπιστήμιο Θεσσαλίας
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

(Υπογραφή)

.....
Γεώργιος Σηκωτίδης

Περίληψη

Τα τελευταία χρόνια υπήρξε μια ραγδαία αύξηση των ηλεκτρονικών συσκευών που είναι συνδεδεμένες στο Διαδίκτυο. Κατά κύριο λόγο, αυτές οι συσκευές αποτελούνται από κινητά τηλέφωνα, αλλά όλο και περισσότερο προστίθενται και παραδοσιακές οικιακές συσκευές μέσω της εξέλιξης των λεγόμενων έξυπνων σπιτιών. Όλα αυτά δημιουργούν έναν τεράστιο όγκο πληροφοριών ο οποίος πρέπει να αποθηκευτεί σε κατάλληλες βάσεις δεδομένων. Επίσης, όταν απαιτείται εύρεση της πληροφορίας, θα πρέπει να εκτελείται μια αποδοτική και τάχιστη διαδικασία αναζήτησης έτσι ώστε ο τελικός χρήστης να μπορεί να απολαμβάνει την ποιότητα των υπηρεσιών. Η παρούσα πτυχιακή εργασία πραγματεύεται την υλοποιηση ενός δέντρου αναζήτησης με παρεμβολή σε NoSQL συστήματα βάσεων δεδομένων. Στη αναζήτηση με παρεμβολή (interpolation search) η εύρεση της εγγραφής στον (ταξινομημένο) πίνακα γίνεται υπολογίζοντας την πιθανή θέση της εγγραφής σε σχέση με τις τιμές των στοιχείων στα άκρα της περιοχής αναζήτησης. Αν ο πίνακας έχει στοιχεία ομοιόμορφα κατανεμημένα θα απαιτηθούν O(loglogn) συγκρίσεις. Αρχικά, στο κεφάλαιο 1 γίνεται μια εισαγωγή στα big data και στο cloud computing και παρατίθεται τα βασικά χαρακτηριστικά τους. Στο κεφάλαιο 2, παρουσιάζεται το Apache Hadoop μαζί με το Hadoop Distributed File System καθώς και το προγραμματιστικό μοντέλο MapReduce. Στο κεφάλαιο 3, παραουσιάζεται η NoSQL βάση δεδομένων Apache HBase και η γενικότερη δομή της αρχιτεκτονικής της. Στο κεφάλαιο 4 εκπονείται η υλοποίηση της δομής στην Hbase. Παρουσιάζονται σχετικές εργασίες, γίνεται ανάλυση και περιγραφή του προβλήματος και περιγράφεται αναλυτικά όλη η διαδικασία της δημιουγίας της ζητούμενης δομής αναζήτησης με παρεμβολή. Στο κεφάλαιο 5 παρουσιάζονται με πίνακες και διαγράμματα όλα τα πειράματα που διεξήχθηκαν στα πλαίσια της πτυχιακής εργασίας.

Λέξεις Κλειδιά

Apache Hadoop, HBase, NoSQL, MapReduce, Big Data, Cloud Computing, Αναζήτηση παρεμβολής

Abstract

In recent years there has been a rapid increase in devices connected to the Internet. Primarily, these devices are made up of mobile phones, but traditional home appliances are increasingly being added through the development of so-called smart homes. All of this creates a huge amount of information to be stored in appropriate databases. Also, when searching for information, an efficient and faster search process must be performed so that the end user can enjoy the quality of the services. This diploma thesis deals with the implementation of an interpolation search tree (IST) in NoSQL database systems. Using interpolation search to find a record in a (sorted) table is done by estimating the potential position of the record according to the values of the elements at the edges of the region search. If the table has elements uniform distributed, O(log log n) comparisons will be required. Initially, chapter 1 introduces big data and cloud computing and presents their key features. Chapter 2 introduces Apache Hadoop along with the Hadoop Distributed File System as well as the MAPReduce programming model. Chapter 3 presents the Apache Hbase NoSQL database and the general structure of its architecture. Chapter 4 implements the structure of the IST in HBase. Chapter 3 presents related research papers of the literature, analyzes and describes the problem and describes in detail the whole process of creating a search structure using interpolation. Chapter 5 presents all the tests performed in the framework of the thessis with tables and diagrams.

Keywords

Apache Hadoop, HBase, NoSQL, MapReduce, Big Data, Cloud Computing, Interpolation search

στους γονείς μου

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα αναπληρωτή καθηγητή κ. Βασιλακόπουλο Μιχαήλ για την ευκαιρία που μου έδωσε να ασχοληθώ με το συγκεκριμένο θέμα της διπλωματικής, παρά τις όποιες δυσκολίες αντιμετώπισα κατά την διάρκεια της εκπόνησης.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

Η κοινωνία γίνεται ολοένα και περισσότερο εξαρτημένη από έξυπνες συσκευές διασυνδεδένες στο Διαδίκτυο. Αυτό έχει σαν αποτέλεσμα την δημιουργία τεράστιων ποσοτήτων δεδομένων και ως εκ τούτου την ανάγκη αποθήκευσής αλλά και επεξεργασίας τους με τον βέλτιστο δυνατό τρόπο.

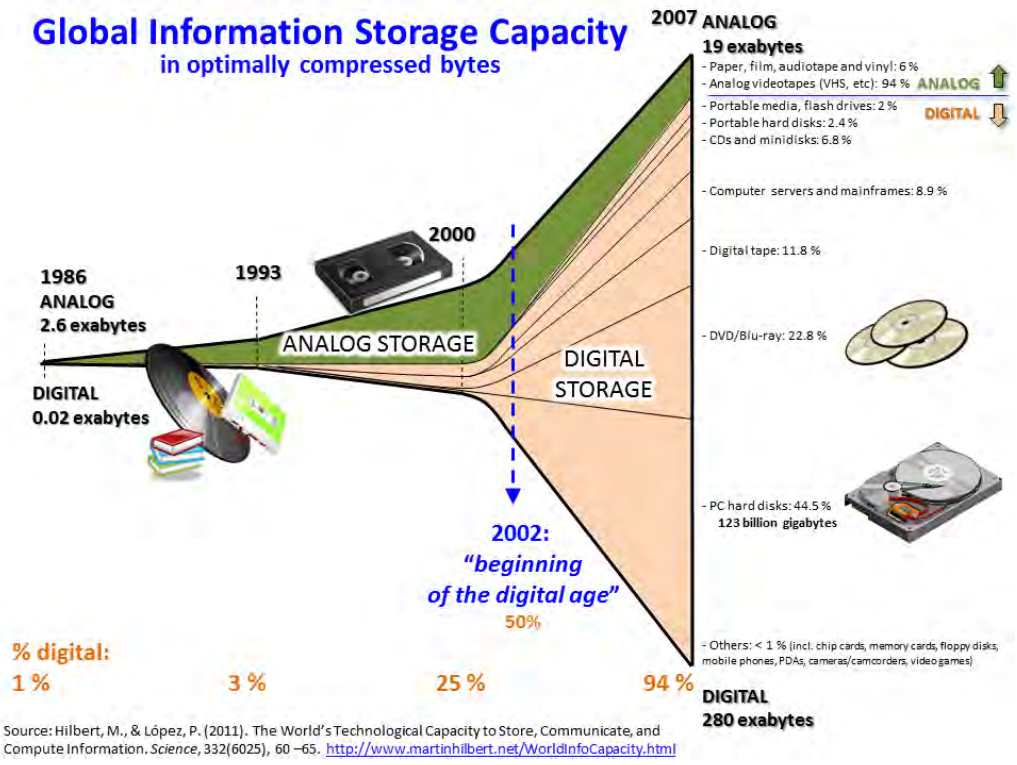
1.1 Big Data

Ένας ακριβής ορισμός του όρου Big Data είναι δύσκολο να διατυπωθεί, διότι κάθε εταιρία, επιχείρηση ή οργανισμός τον χρησιμοποιεί με διαφορετικό τρόπο. Θα μπορούσαμε να πούμε ότι ο όρος Big Data χρησιμοποιείται κυρίως για να περιγράψει την εκθετική αύξηση και διαθεσιμότητα δομημένων ή και αδόμητων μορφών δεδομένων καθώς επίσης για την περιγραφή μη παραδοσιακών στρατηγικών και τεχνολογιών που απαιτούνται για τη συγκέντρωση, οργάνωση, διεκπεραίωση και συλλογή πληροφοριών από μεγάλα σύνολα δεδομένων. Το σχήμα 1.1 δείχνει αυτή την χαρακτηριστική εκθετική αύξηση των δεδομένων τις τελευταίες δεκαετίες. Ως μεγάλο σύνολο δεδομένων νοείται ένα σύνολο που είναι πολύ μεγάλο για να επεξεργαστεί ή να αποθηκευτεί με παραδοσιακά εργαλεία ή σε έναν μόνο υπολογιστή. Τα Big Data αναφέρονται κυρίως στις τρεις παρακάτω κατηγορίες δεδομένων [12]:

- Παραδοσιακά δεδομένα εταιριών, τα οποία περιλαμβάνουν κυριώς πληροφορίες πελατών που προέρχονται από συστήματα διαχείρισης πελατειακών σχέσεων (Customer Relationship Management, CRM) και δεδομένα από συστήματα ενδοεπιχειρησιακού σχεδιασμού (Enterprise Resource Planning, ERP).

- Δεδομένα παραγόμενα από μηχανές/αισθητήρες, τα οποία προέρχονται κυρίως από καταγραφές ιστού (weblogs), έξυπνους μετρητές (smart meters) και συστήματα συναλλαγών.

- Δεδομένα κοινωνικών μέσων, τα οποία προέρχονται κυρίως από σχόλια ανατροφοδότησης χρηστών (customer feedback) και γενικότερα από δημοφιλή sites κοινωνικών μέσων δικτύωσης όπως το Twitter και το Facebook.



Σχήμα 1.1: Ανάπτυξη της παγκόσμιας χωρητικότητας αποθήκευσης πληροφοριών, [15]

1.1.1 Χαρακτηριστικά των Big Data

Αρκετά χρόνια πριν, το πρόβλημα της επεκτασιμότητας (scalability) στην αποθήκευση δεδομένων ήταν ένα από τα κύρια τεχνικά θέματα προς αντιμετώπιση. Παρ όλα αυτά, με την είσοδο των νέων τεχνολογίων για τα Bid Data και τις καινούριες μεθόδους διαχείρισης δεδομένων αποθήκευσης, έχει σχεδόν πλήρως αντιμετωπιστεί. Επιπλέον, παράγονται συνεχώς δεδομένα, όχι μόνο με τη χρήση του διαδικτύου αλλά και από εταιρείες που παράγουν μεγάλα ποσά πληροφοριών που προέρχονται από αισθητήρες, υπολογιστές και αυτοματοποιημένες διαδικασίες. Το φαινόμενο αυτό επιταχύνθηκε τα τελευταία χρόνια ακόμη περισσότερο χάρη στην αύξηση των συνδεδεμένων συσκευών (κυρίως κινητών τηλεφώνων) καθώς και στην παγκόσμια απήχηση των κοινωνικών μέσων δικτύωσης. Εταιρίες παγκόσμιου βεληνεκούς όπως η Google, η Facebook, η Amazon και η Twitter ήταν οι πρώτες που αντιμετώπισαν την συνεχόμενη αύξηση των δεδομένων τους, βρίσκοντας κυρίως ad-hoc λυσεις. Οι λύσεις αυτές, με την πάροδο του χρόνου διαμοιράστηκαν στην παγκόσμια κοινότητα αποτελώντας το αρχικό σημείο για την περαιτέρω εξάπλωση των Big Data τεχνολογιών και σε μικρότερες επιχειρήσεις και οργανισμούς [14].

Παρακάτω, τα κύρια χαρακτηριστικά των Big Data:

- **Όγκος (Volume)**: Το πρώτο χαρακτηριστικό αναφέρεται στην ποσότητα των δεδομένων που αναλύονται και διαχειρίζονται προκειμένου να επιτευχθούν τα επιθυμητά αποτελέσματα. Δεδομένα που δημιουργούνται από μηχανές (machine-generated), παράγονται σε υπερβολικά μεγάλες ποσότητες. Για παράδειγμα, ένα κινητήρας jet μπορεί

να δημιουργεί 10TB δεδομένων σε 30 λεπτά και με χιλιάδες πτήσεις καθημερινά, ο όγκος των δεδομένων φτάνει στα όρια των Petabytes.

- **Ταχύτητα (Velocity)**: Αυτό το χαρακτηριστικό αναφέρεται στους αυξανόμενους ρυθμούς παραγωγής δεδομένων, καθώς όλο και περισσότερα δεδομένα δημιουργούνται και πρέπει να συλλέγονται σε μικρότερα χρονικά διαστήματα. Τα δεδομένα των κοινωνικών μέσων δικτύωσης (παρόλο που δεν είναι τόσο μαζικά όσο τα δεδεμένα που παράγονται από μηχανές) αποτελούν αντιπροσωπευτικό παράδειγμα αυτού του χαρακτηριστικού.

- **Ποικιλία (Variety)**: Το τρίτο χαρακτηριστικό των Big Data αντιπροσωπεύει τον τύπο των δεδομένων που αποθηκεύονται, αναλύονται και χρησιμοποιούνται. Αυτός ποικίλλει και μπορεί να αποτελείται από συντεταγμένες τοποθεσίας, αρχεία βίντεο, δεδομένα που αποστέλλονται από προγράμματα περιήγησης, προσομοιώσεις κλπ. Η πρόκληση είναι πώς να ταξινομηθούν όλα αυτά τα δεδομένα ώστε να μπορούν να διαβαστούν από όλους τους χρήστες που έχουν πρόσβαση και να μην δημιουργούνται διφορούμενα αποτελέσματα.

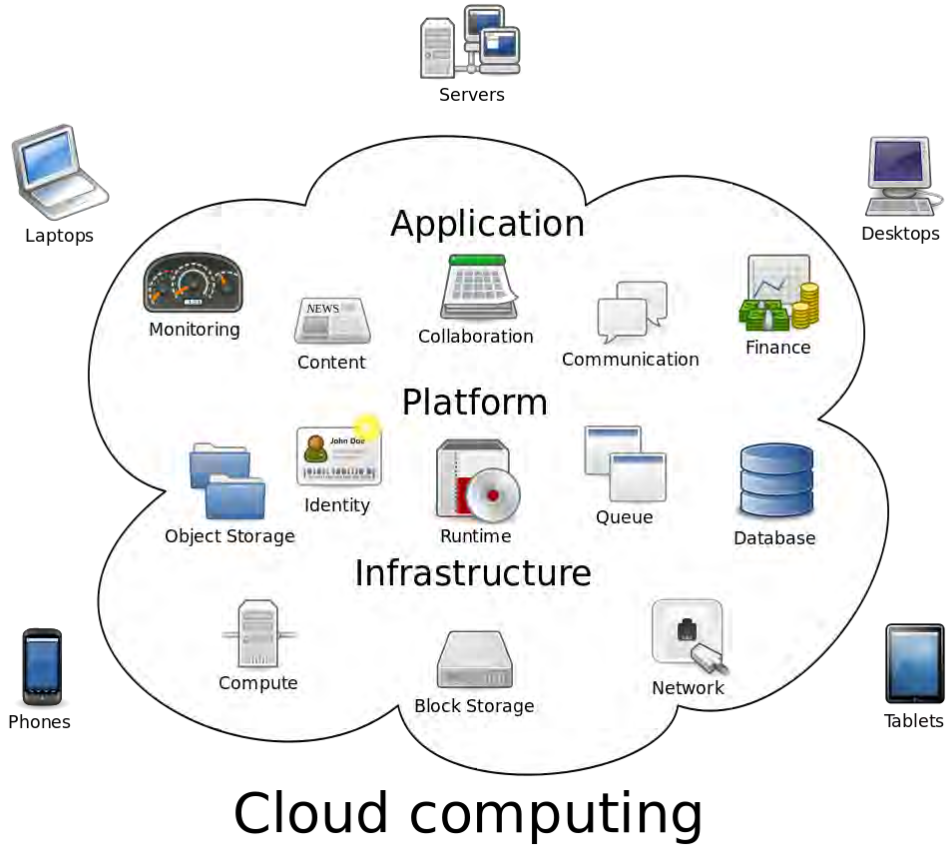
- **Αξία (Value)**: Η αξία αφορά την ποιότητα των αποθηκευμένων δεδομένων και την περαιτέρω χρήση τους. Για παράδειγμα, ενώ μεγάλες ποσότητες δεδομένων αποθηκεύονται από τις εγγραφές κλήσεων κινητών τηλεφώνων, το ερώτημα είναι αν όλα μαζί μπορούν να έχουν οποιαδήποτε εμπορική αξία.

Μια πρόσφατη μελέτη διαπίστωσε ότι το 90% των στελεχών εταιριών πιστεύει ότι τα δεδομένα γίνονται ο τέταρτος παράγοντας παραγωγής για επιχειρήσεις που είναι απαραίτητες, όπως η γη, η εργασία και το κεφάλαιο. Συνεπώς, ο όρος «λήψη αποφάσεων βάσει δεδομένων» (data-driven decision making) περιγράφει τη διαδικασία συλλογής και ανάλυσης δεδομένων για την καθοδήγηση ή τη βελτίωση των αποφάσεων (Big Data challenges). Αυτό περιλαμβάνει την ανάλυση μη-συναλλακτικών (non-transactional) και αδόμητων δεδομένων όπως ιδέες προϊόντων ή κριτικές που δημιουργούνται από τους καταναλωτές. Στην πραγματικότητα, ειδικοί επί του θέματος (data specialists) διερευνούν τα Big Data που συλλέγονται, για παράδειγμα, από τα κοινωνικά μέσα δικτύωσης για να διεξαγάγουν έρευνες, δοκιμάζοντας μια συγκεκριμένη υπόθεση έτσι ώστε να μπορούν να καθορίσουν την αξία, την εγκυρότητα και τη σκοπιμότητα αυτών των ιδεών και να προετοιμάσουν τα σχέδια για την εκτέλεση τους. Οι ειδικοί χρησιμοποιούν διάφορα όργανα «ακρόασης» για να διεξαγάγουν ανάλυση κειμένων και μέσω αυτών των εργαλείων να μπορούν να μετρήσουν ορισμένες πτυχές ενδιαφέροντος για τα προϊόντα τους και να λάβουν τις αναγκαίες ενέργειες για τη διόρθωση ή τη βελτίωση.

1.2 Cloud Computing

Το Cloud Computing μπορεί να οριστεί ως η χρήση της τεχνολογίας υπολογιστών που εκμεταλλεύεται την επεξεργαστική ισχύ πολλών διασυνδεδεμένων υπολογιστών σ' ένα δίκτυο, ενώ ταυτόχρονα αποκρύπτεται τη δομή που βρίσκεται πίσω από αυτό [15]. Η προέλευση του

όρου cloud (σύννεφο) προέρχεται μάλλον από την φύση του πλαισίου αυτής της τεχνολογίας, το σύστημα δηλάδή λειτουργεί για τους χρήστες, αλλά οι τελευταίοι δεν έχουν καμία ιδέα για την πολυπλοκότητα του συστήματος. Αυτό που δεν συνειδητοποιούν οι χρήστες είναι ότι υπάρχει ένας τεράστιος όγκος δεδομένων σε παγκόσμιο επίπεδο που επεξεργάζεται σε πραγματικό χρόνο για να μπορούν να λειτουργούν τις εφαρμογές τους. Το σχήμα 1.2 δείχνει την γενική μορφή του cloud, καθώς και των συσκευών και υπηρεσίων που συνδέονται και επικοινωνούν μεταξύ τους.



Σχήμα 1.2: Το cloud computing και οι διασυνδέσεις των διαφόρων στοιχείων του, [14]

1.2.1 Χαρακτηριστικά του Cloud Computing

Το Cloud Computing παρουσιάζει τα ακόλουθα βασικά χαρακτηριστικά [12]:

- Η **ευελιξία** (agility) των οργανισμών μπορεί να βελτιωθεί, καθώς το Cloud Computing μπορεί να δώσει στους χρήστες την δυνανότητα της επαναφοράς, προσθήκης ή επέκτασης των υπολογιστικών πόρων της εκάστοτε τεχνολογικής υποδομής.

- **Χαμηλό κόστος** χρήσης και συντήρησης. Κάποιος που χρειάζεται να εκτελέσει πολύπλοκες υπολογιστικές διαδικασίες σε ισχυρούς υπολογιστικούς πόρους για μη τακτά

χρονικά διαστήματα ή μόνο μια φορά, δεν θα χρειαστεί να αγοράσει ανάλογο εξοπλισμό, αλλά θα μπορεί να νοικιάζει και να χρησιμοποιεί κάποιο σύστημα Cloud.

- Η **ανεξαρτησία** τοποθεσίας και συσκευής (Device and location independence) περιγράφει την δυνανότητα των χρηστών να έχουν πρόσβαση σε συστήματα χρησιμοποιώντας ένα πρόγραμμα περιήγησης ιστού, ανεξάρτητα από την τοποθεσία τους ή τη συσκευή που χρησιμοποιούν (π.χ. PC, κινητό τηλέφωνο).

- Η **συντήρηση** των εφαρμογών του Cloud Computing είναι ευκολότερη, επειδή δεν χρειάζεται να εγκατασταθεί στον υπολογιστή του κάθε χρήστη και μπορεί να διαχειριστεί από διαφορετικά μέρη.

- Εύκολη **παρακολούθηση** της απόδοσης και επίδοσης των συστημάτων Cloud από ειδικούς τεχνικούς πληροφορικής μέσα από ειδικά συστήματα ανίχνευσης σφαλμάτων.

- Η **παραγωγικότητα** μπορεί να αυξηθεί αφού πολλοί χρήστες μπορούν να δουλέψουν ταυτόχρονα στα ίδια δεδομένα.

- Ένα καλά σχεδιασμένο σύστημα Cloud Computing μπορεί να προσφέρει υψηλή **αξιοπιστία** στις υπηρεσίες οποιασδήποτε επιχείρησης ή οργανισμού.

- Δύο ακόμη σημαντικές έννοιες του Cloud Computing, η **επεκτασιμότητα** (scalability) και η **ελαστικότητα** (elasticity), δίνουν την δυνατότητα στους χρήστες για δυναμική διαχείριση των υπολογιστικών πόρων ανάλογα με τις ανάγκες τους σε συγκεκριμένες χρονικές στιγμές.

- Αυτοεξυπηρέτηση υπηρεσιών κατ απαίτηση (self-service on demand). Ένας χρήστης μπορεί μονομερώς να χρησιμοποιεί υπολογιστικού πόρους, όπως servers και δίκτυο, χωρίς να απαιτείται ανθρώπινη αλληλεπίδραση με κάποιον πάροχο υπηρεσιών.

1.2.2 Μοντέλα υπηρεσιών (Service models)

Οι υπηρεσίες cloud μπορούν να χωριστούν σε τρεις κατηγορίες, ανάλογα με το επίπεδο υπηρεσιών και πόρων που παρέχουν στους χρήστες. Αυτές είναι: το μοντέλο λογισμικού σαν υπηρεσία (Software-as-a-Service - SaaS), το μοντέλο πλατφόρμας σαν υπηρεσία (Platform-as-a-Service - Paas) και το μοντέλο υποδομής σαν υπηρεσία (Infrastructure-as-a-Service - IaaS). Παρακάτω παρουσιάζονται συνοπτικά τα σημαντικότερα χαρακτηριστικά αυτών των μοντέλων [15].

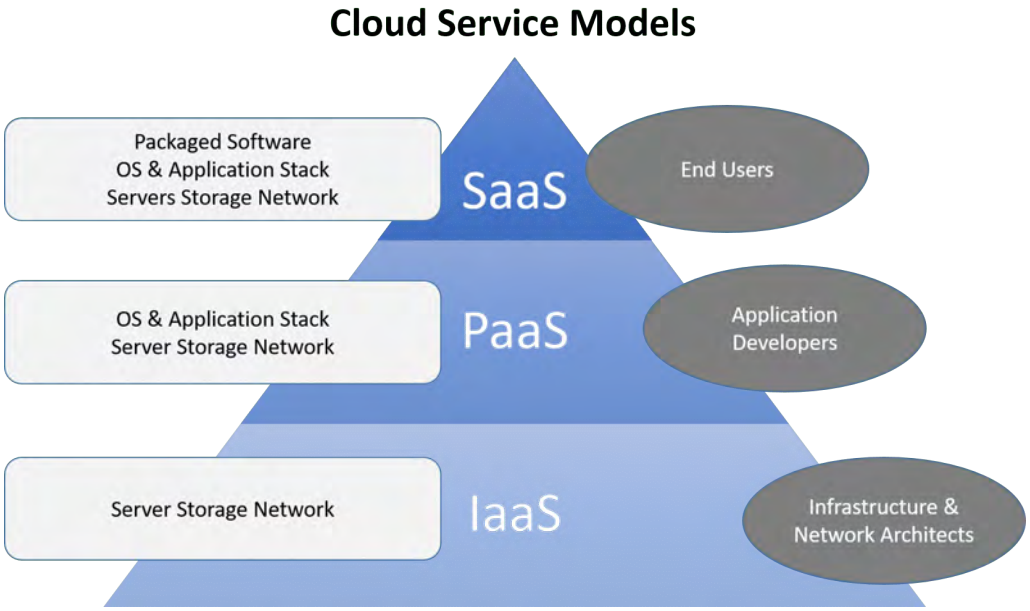
- **Λογισμικό σαν υπηρεσία (Software-as-a-Service - SaaS)**: Η δυνατότητα που έχει ο πελάτης να χρησιμοποιεί τις εφαρμογές ενός παρόχου που εκτελούνται σε μια υποδομή cloud. Οι εφαρμογές είναι προσβάσιμες από απομακρυσμένες συσκευές του πελάτη και μέσω διαφόρων εφαρμογών όπως π.χ. ενός προγράμματος περιήγησης ιστού (browser). Ο πελάτης δεν μπορεί να ελέγχει ή να διαχειρίζεται τις υποδομές του cloud,

όπως τις ρυθμίσεις δικτύου, τους servers, τους αποθηκευτικούς χώρους κ.α.. Παραδείγματα SaaS είναι τα Google Apps(Gmail, Drive, Calendar ...), το Dropbox και το Office 365.

- **Πλατφόρμα σαν υπηρεσία (Platform-as-a-Service - PaaS)**: Η δυνατότητα που παρέχεται στον πελάτη ειναι η ανάπτυξη, μέσα στην υποδομή του cloud, διαφόρων εφαρμογών και συστημάτων με την χρήση γλωσσών προγραμματισμού, βιβλιοθηκών και υπηρεσιών που υπάρχουν στο σύστημα. Έτσι, ο πελάτης έχει τον έλεγχο οποιασδήποτε εφαρμογής επιλέξει από το cloud χωρίς να χρειάζεται να την εγκαταστήσει στο σύστημά του και να την συντηρεί. Η ευθύνη είναι αποκλειστικά του παρόχου. Παραδείγματα PaaS είναι τα Google App Engine, Salesforce1 Platform και Appistry.

- **Υποδομή σαν υπηρεσία (Infrastructure-as-a-Service - IaaS)**: Η δυνατότητα που παρέχεται στον πελάτη είναι να νοικιάζει δυναμικά υπολογιστικούς πόρους (αποθήκευση, δίκυτα, επεξεργαστές κ.α.), στους οποίους μπορεί να προσθέσει τις δικές του εφαρμογές ή λειτουργικό σύστημα, δημιουργώντας τις δικές του εικονικές μηχανές (virtual machines). Κάποια παραδείγματα IaaS είναι τα Amazon Web Services, RackSpace, Windows Azure, Google Cloud Platform και Okeanos GR.NET

Στο σχήμα 1.3 παρουσιάζονται τα διάφορα μοντέλα υπηρεσιών cloud, στους χρήστες στους οποίους απευθύνονται καθώς και τις υπηρεσίες που προσφέρει το καθένα.



Σχήμα 1.3: Μοντέλα υπηρεσιών cloud και οι χρήστες που απευθύνονται, [2]

Κεφάλαιο 2

Apache Hadoop

Το Apache Hadoop είναι μια πλατφόρμα λογισμικού ανοιχτού κώδικα για κατανεμημένη αποθήκευση και εξεργασία πολύ μεγάλων συνόλων δεδομένων σε συστάδες (cluster) υπολογιστών [13]. Οι υπηρεσίες Hadoop παρέχουν αποθήκευση, επεξεργασία, διαχείριση και ασφάλεια δεδομένων. Το Hadoop καθιστά δυνατή την εκτέλεση εφαρμογών σε συστήματα με χιλιάδες κόμβους υπολογιστών καθώς και τη διαχείριση χιλιάδων terabytes δεδομένων. Διαθέτει δικό του σύστημα διαχείρισης αρχείων, το Hadoop Distributed File System (HDFS), το οποίο διευκολύνει την ταχεία μεταφορά δεδομένων μεταξύ των κόμβων και επιτρέπει στο σύστημα να συνεχίσει να λειτουργεί σε περίπτωση κάποιας αστοχίας στους κόμβους. Αυτή η προσέγγιση μειώνει τον κίνδυνο κάποιας αποτυχίας στο σύστημα ή κάποιας απροσδόκητης απώλειας δεδομένων, ακόμη και αν ένας σημαντικός αριθμός κόμβων καταστεί ανενεργός. Κατά συνέπεια, το Hadoop γρήγορα αναδείχθηκε ως βασικός πυλώνας για την επεξεργασία διαφόρων τομέων Big Data, όπως η επιστημονική ανάλυση, ο σχεδιασμός επιχειρήσεων και πωλήσεων, η επεξεργασία τεράστιων όγκων δεδομένων για δίκτυα αισθητήρων κ.α..

Το Hadoop προήλθε από το σύστημα αρχείων της Google (Google File System), το οποίο δημοσιεύθηκε τον Οκτώβριο του 2003, [5]. Η δημοσίευση αυτή οδήγησε και σ΄ ένα ακόμα ερευνητικό έγγραφο από την Google, σχετικό με το προγραμματιστικό μοντέλο MapReduce[3]. Η ανάπτυξη ξεκίνησε από το έργο Apache Nutch, το οποίο οδήγησε στην πρώτη έκδοση του Hadoop από τον Owen O'Malley τον Απρίλιο του 2006 (έκδοση 0.1.0). Για ιστορικούς λόγους, επίσης, το Hadoop πήρε το όνομά του από έναν από τους ελέφαντες του ιδρυτή.

2.1 Βασικά στοιχεία του συστήματος Hadoop

Ο πυρήνας του Apache Hadoop αποτελείται, όπως έχει ήδη αναφερθεί, από ένα σύστημα αποθήκευσης, γνωστό ως Hadoop Distributed File System (HDFS), και ένα σύστημα παράλληλης επεξεργασίας, το μοντέλο προγραμματισμού MapReduce. Το Hadoop χωρίζει τα αρχεία σε μεγάλα μπλοκ και τα διανέμει σε διάφορους υπολογιστικούς κόμβους, έτσι ώστε στη συνέχεια να επεξεργαστεί τα δεδομένα παράλληλα. Στα βασικά στοιχεία του Hadoop συγκαταλέγονται οι παρακάτω τεχνολογίες [13]:

- Hadoop Common - Περιέχει βιβλιοθήκες και βοηθητικά προγράμματα που απαιτούνται από άλλα υποσυστήμτα του Hadoop

- Hadoop Distributed File System (HDFS) - Ένα κατανεμημένο σύστημα αρχείων που αποθηκεύει δεδομένα σε μηχανές, παρέχοντας πολύ υψηλό συνολικό εύρος ζώνης σε όλη την συστάδα υπολογιστών (cluster).

- Hadoop YARN - Πλατφόρμα υπεύθυνη για τη διαχείριση των υπολογιστικών πόρων στα clusters

- Hadoop MapReduce - Εφαρμογή του προγραμματισμού μοντέλου MapReduce για επεξεργασία δεδομένων μεγάλης κλίμακας.

Επιπρόσθετα, στο ευρύτερο σύστημα του Hadoop, υπάρχει ένα σύνολο πακέτων λογισμικού που συνυπάρχουν μαζί, όπως τα:

- Apache Pig - Μία γλώσσα υψηλού επιπέδου ερωτημάτων, δεδομένων ροής (Pig Latin) και εκτέλεσης σε παράλληλα υπολογιστικά συστήματα. Αναλύει μεγάλα σύνολα δεδομένων και πετυχαίνει και τη συγχώνευσή τους.

- Apache Hive - Το Facebook χρησιμοποιεί τη γλώσσα Hive. Είναι μία γλώσσα υψηλότερου επίπεδου ερωτημάτων, χτισμένη πάνω στο MapReduce. Είναι η υποδομή αποθήκης δεδομένων που παρέχει εργαλεία για την περιληπτική παρουσίαση δεδομένων, την ad-hoc αναζήτηση (έρευνα – εντοπισμό – ανάκτηση δεδομένων με μεγάλη ταχύτητα) και την ανάλυση μεγάλων συνόλων δεδομένων, που είναι αποθηκευμένα σε αρχεία Hadoop. Παρέχει μηχανισμό, ο οποίος αλλάζει τα δεδομένα σε δομημένα δεδομένα και, επίσης, παρέχει μια απλή γλώσσα ερωτημάτων, την HiveQL, η οποία βασίζεται στην SQL, όμως, επιτρέπει και map/reduce συναρτήσεις.

- Apache HBase - Μία κατανεμημένη βάση δεδομένων, που υποστηρίζει την αποθήκευση δομημένων δεδομένων για μεγάλους πίνακες.

- Apache ZooKeeper - Μία υψηλής απόδοσης υπηρεσία για κατανεμημένες εφαρμογές. Μια κεντρική υπηρεσία για τη διατήρηση πληροφοριών διαμόρφωσης, ονομασιών, παροχής κατανεμημένου συγχρονισμού και παροχής ομάδων.

- Apache Flume - Είναι μία κατανεμημένη και αξιόπιστη υπηρεσία για τη μετακίνηση μεγάλων ποσοτήτων δεδομένων.

- Apache Sqoop - SQL στο Hadoop (Sqoop) είναι ένα εργαλείο μιας απλής γραμμής εντολών, που εισάγει μεμονωμένους πίνακες ή ολόκληρη βάση δεδομένων στο HDFS, δημιουργεί Java κλάσεις, για να μπορεί ο χρήστης να επέμβει στα δεδομένα του. Τέλος, παρέχει τη δυνατότητα στον χρήστη να εισάγει βάσεις δεδομένων SQL στο σύστημα αποθήκευσης του Hive.

- Apache Oozie - Είναι μία υπηρεσία που διαχειρίζεται εργασίες επεξεργασίας δεδομένων και συντονίζει τις εξαρτήσεις μεταξύ των εργασιών που εκτελούνται στο Hadoop (και των HDFS, Pig, MapReduce).

Το Hadoop είναι γραμμένο κυρίως στη γλώσσα προγραμματισμού Java, με κάποια μέρη σε C και κάποια βοηθητικά shell scripts. Παρ όλα αυτά, οποιαδήποτε γλώσσα προγραμματισμού μπορεί να χρησιμοποιηθεί στο προγραμματιστικό μοντέλο MapReduce του Hadoop για την εφαρμογή του map και του reduce των τμημάτων του προγράμματος του χρήστη.

2.1.1 MapReduce

Το MapReduce είναι ένα προγραμματιστικό μοντέλο που αναπτύχθηκε, αρχικά από την Google, για να υποστηρίξει κατανεμημένο υπολογισμό σε μεγάλα σύνολα δεδομένων σε συστοιχίες (clusters) υπολογιστών, [9]. Τα προγράμματα MapReduce μπορούν να είναι γραμμένα σε διάφορες γλώσσες, όπως Java, Ruby, Python και C++.

Μια επεξεργασία δεδομένων με το μοντέλο MapReduce αποτελείται από δύο φάσεις, από την φάση map (χαρτογράφηση) και την φάση reduce (μείωση). Η κάθε φάση έχει στην είσοδο και στην έξοδό του ένα ζευγάρι κλειδιού-τιμής αντίστοιχα, ο τύπος των οποίων μπορεί να επιλεγεί από τον προγραμματιστή. Στη φάση map γίνεται ανάγνωση των δεδομένων εισόδου, τα οποία αναλύονται και επεξεργάζονται, ώστε να διατηρηθούν οι χρήσιμες πληροφορίες μόνο, οι οποίες αποθηκεύονται στη μορφή κλειδιού-τιμής (key-value). Στη συνέχεια, αυτά τα ζεύγη εισέρχονται στη φάση reduce, όπου γίνεται περαιτέρω επεξεργασία τους και διατηρείται μόνο η απαραίτητη πληροφορία. Παρακάτω, παρουσιάζεται ένα απλό παράδειγμα όπως περιγράφεται στο [13]. Το πρόγραμμα επεξεργάζεται καιρικά δεδομένα και υπολογίζει τη μέγιστη θερμοκρασία που παρατηρήθηκε για κάθε έτος. Τα δεδομένα εισόδου, που φαίνονται στο σχήμα 2.1, περνάνε αρχικά από την φάση map. Το κλειδί εισόδου για τη συνάρτηση map είναι ο αριθμός της κάθε γραμμής στο αρχείο που γίνεται η ανάγνωση, αλλά επειδή δεν αποτελεί σημαντικό στοιχείο, αγνοείται. Οι τιμές εισόδου είναι οι γραμμές του αρχείου. Η συνάρτηση map επεξεργάζεται την κάθε γραμμή του αρχείου και εξάγει την χρονιά και την παρατηρήσιμη θερμοκρασία. Η χρονιά αποτελεί για τη συνάρτηση map το κλειδί (key) εξόδου και η θερμοκρασία την τιμή (value) εξόδου. Η συνάρτηση map παράγει δηλαδή τα παρακάτω ζεύγη:

(1950, 0)
(1950, 22)
(1950, -11)
(1949, 111)
(1949, 78)

Πριν αποσταλούν στην φάση reduce, το MapReduce επεξεργάζεται τις τιμές αυτές, τις ομαδοποιεί και τις ταξινομεί ανα κλειδί. Επομένως, η είσοδος της συνάρτησης reduce είναι τα παρακάτω ζεύγη κλειδιού-συνόλου τιμών όπου το κλειδί είναι η χρονιά και οι τιμές μια λίστα θερμοκρασιών.

```
(0, 0067011990999991950051507004...9999999N9+00001+99999999999...)
(106, 0043011990999991950051512004...9999999N9+00221+99999999999...)
(212, 0043011990999991950051518004...9999999N9-00111+99999999999...)
(318, 0043012650999991949032412004...0500001N9+01111+99999999999...)
(424, 0043012650999991949032418004...0500001N9+00781+99999999999...)
```

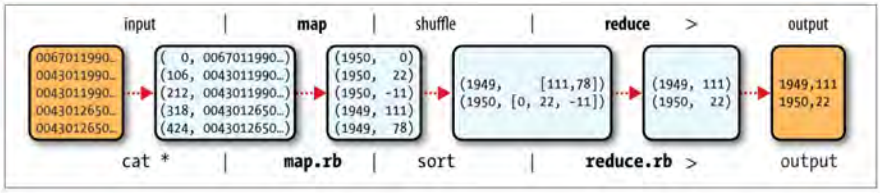
Σχήμα 2.1: Παράδειγμα δεδομένων εισόδου για το MapReduce, [13]

(1949, [111, 78])
(1950, [0, 22, -11])

Τέλος, η συνάρτηση reduce αναλύει τις τιμές εισόδου της, βρίσκει τη μέγιστη τιμή θερμοκρασίας για κάθε κλειδί και εξάγει ένα ζεύγος κλειδιού-τιμής (key-value), όπου το κλειδί είναι η χρονιά και η τιμή η μέγιστη θερμοκρασία την αντίστοιχη χρονιά.

(1949, 111)
(1950, 22)

Στο σχήμα 2.2 φαίνεται η λογική ροή των δεδομένων στο MapReduce



Σχήμα 2.2: Η λογική ροή των δεδομένων στο MapReduce, [13]

2.1.2 Hadoop Distributed File System (HDFS)

Όταν ένα σύνολο δεδομένων ξεπερνά τη χωρητικότητα αποθήκευσης ενός και μόνο υπολογιστή, καθίσταται αναγκαίο να κατακερματιστεί σε ένα σύνολο από υπολογιστές. Τα συστήματα αρχείων, που διαχειρίζονται την αποθήκευση σε ένα δίκτυο υπολογιστών, ονομάζονται κατανεμημένα συστήματα αρχείων. Δεδομένου ότι αυτά βασίζονται σε κάποιο δίκτυο δεδομένων, υπεισέρχονται όλες οι επιπλοκές του προγραμματισμού μέσω δικτύου, καθιστώντας έτσι τα κατανεμημένα συστήματα αρχείων πιο περίπλοκα απ' ό,τι τα συστήματα αρχείων σε έναν μόνο σκληρό δίσκο. Για παράδειγμα, μία από τις μεγαλύτερες προκλήσεις είναι το κατανεμημένο σύστημα αρχείων να αντιμετωπίζει βλάβες σε υπολογιστικούς κόμβους, χωρίς να υποστεί απώλεια δεδομένων.

Το κατανεμημένο σύστημα αρχείων του Hadoop (HDFS) είναι ένα σύστημα σχεδιασμένο να μπορεί τρέχει σε τυπικούς υπολογιστές, έχει πολλές ομοιότητες, αλλά και σημαντικές δια-

φορές με άλλα κατανεμημένα συστήματα αρχείων που ήδη υπάρχουν. Το HDFS είναι σχεδιασμένο για την αποθήκευση πολύ μεγάλων αρχείων, με υποστήριξη για πρότυπα προσπέλασης σε δεδομένα συνεχούς ροής (streaming data access patterns), σε συστάδες τυπικών υπολογιστών (clusters). Στο πλαίσιο του Hadoop κάτι τέτοιο σημαίνει ότι τα αρχεία μπορεί να έχουν μέγεθος που είναι εκατοντάδες Megabytes, Gigabytes ή ακόμα και Terabytes. Υπάρχουν σήμερα cluster Hadoop σε λειτουργία, που αποθηκεύουν Petabytes δεδομένων. Το HDFS είναι χτισμένο γύρω από την ιδέα ότι το πιο αποτελεσματικό πρότυπο επεξεργασίας δεδομένων είναι αυτό της «εγγραφής-μια φορά, ανάγνωσης-πολλές φορές» (write-once, read-multiple times). Τυπικά, ένα σύνολο δεδομένων παράγεται ή αντιγράφεται από προϋπάρχουσα πηγή, και, στη συνέχεια, πραγματοποιούνται διάφορες αναλύσεις σε αυτό το σύνολο δεδομένων. Κάθε ανάλυση περιλαμβάνει ένα μεγάλο μέρος, αν όχι όλο, από το σύνολο των δεδομένων. Επίσης, το Hadoop δεν απαιτεί ακριβό ή εξαιρετικά αξιόπιστο υλικό. Έχει σχεδιαστεί, ώστε να εκτελείται σε clusters που αποτελούνται από τυπικούς υπολογιστές, το υλικό των οποίων μπορεί να προμηθευτεί κάποιος από περισσότερους προμηθευτές. Αυτό, βέβαια, καθιστά την πιθανότητα αποτυχίας ενός κόμβου στο cluster υψηλή, ιδιαίτερα για clusters μεγάλου μεγέθους. Το HDFS, παρ' όλα αυτά, έχει σχεδιαστεί για να δουλεύει αξιόπιστα και χωρίς εμφανείς διακοπές λειτουργίας προς τον χρήστη σε περίπτωση τέτοιων προβλημάτων.

Η αρχιτεκτονική του HDFS βασίζεται στη σχέση master-slave μεταξύ των υπολογιστικών συστημάτων. Ο master του HDFS αναλαμβάνει το ρόλο του NameNode, ένας κόμβος του δικτύου που διαχειρίζεται το σύστημα αρχείων και καθορίζει την πρόσβαση των υπολοιπων υπολογιστών σε αυτά. Οι slaves του HDFS αναλαμβάνουν τον ρόλο των datanodes οι οποίοι διαχειρίζονται τον αποθηκευτικό χώρο που αντιστοιχεί στον καθένα. Στο σχήμα 2.3 απεικονίζεται η αρχιτεκτονική του HDFS.

2.1.3 Εγκατάσταση του Apache Hadoop

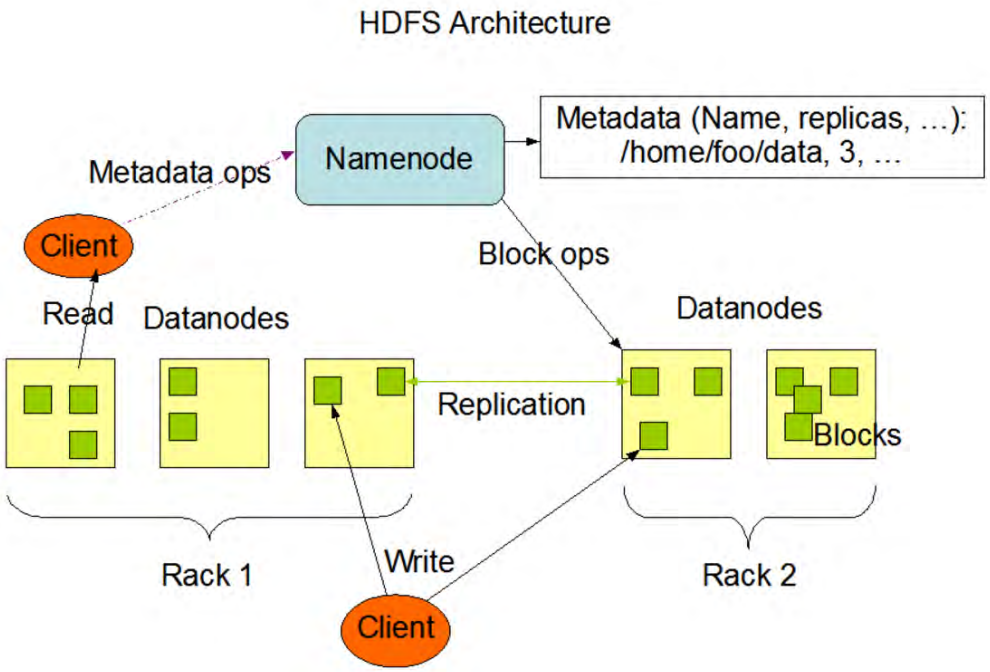
Η εγκατάσταση του Apache Hadoop έγινε σε υπολογιστή που είχε εγκατεστημένο το λειτουργικό σύστημα Ubuntu 14.04. Παρατίθενται οι βασικές οδηγίες εγκατάστασης σε pseudo-distributed mode. Η έκδοση του Apache Hadoop που χρησιμοποιήθηκε είναι η 2.7.1. Αρχικά πρέπει να κατεβάσουμε το installation package που επιθυμούμε από ενα repository και να το κανουμε extract:

$ wget http://mirrors.sonic.net/apache/hadoop/common/hadoop-2.7.1/hadoop-2.7.1.tar.gz

$ tar xvzf hadoop-2.7.1.tar.gz

Στην συνέχεια μεταφέρουμε τον φάκελο σε κάποιον κατάλογο του συστήματος που επιθυμούμε (κυρίως για λόγους οργάνωσης, εδώ στο usr/local/hadoop)

$ sudo mv * /usr/local/hadoop



Σχήμα 2.3: Η αρχιτεκτονική του HDFS, [1]

Στο επόμενο βήμα πρέπει να βάλουμε τις κατάλληλες ρυθμίσεις στα configuration files του Hadoop:

1. ∼/.bashrc
2. /usr/local/hadoop/etc/hadoop/hadoop-env.sh
3. /usr/local/hadoop/etc/hadoop/core-site.xml
4. /usr/local/hadoop/etc/hadoop/mapred-site.xml.template
5. /usr/local/hadoop/etc/hadoop/hdfs-site.xml

1. ∼/.bashrc

Πριν τροποποιήσουμε αυτό το αρχείο, θα πρέπει να προσδιορίσουμε το path του συστήματος στο οποίο ειναι εγκατεστημένη η JAVA. Αυτό γίνεται γράφοντας στο terminal:

$ echo $JAVA_HOME

οπότε στην δική μας περίπτωση έχουμε ένα αποτέλεσμα σαν κι αυτό: /usr/lib/jvm/java-7-openjdk-amd64. Στην συνέχεια ανοίγουμε το αρχείο /.bashrc και προσθέτουμε τα παρακάτω στο τέλος:

$ vim ∼/.bashrc

```
.....
#HADOOP VARIABLES START
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
#HADOOP VARIABLES END
.....
```

Αποθηκεύουμε το αρχείο και στη συνέχεια εκτελούμε από το terminal την εντολή:

$ source ∼/.bashrc

2. /usr/local/hadoop/etc/hadoop/hadoop-env.sh

Ανοίγουμε το αρχείο hadoop-env.sh κι ενημερώνουμε την τιμή του JAVA_HOME:

$ vim conf/hadoop-env.sh

```
.....
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
.....
```

3. /usr/local/hadoop/etc/hadoop/core-site.xml

Ανοίγουμε το αρχείο core-site.xml και γράφουμε τα ακόλουθα ανάμεσα στα tags του configuration:

$ vim /usr/local/hadoop/etc/hadoop/core-site.xml

```
<configuration>
 <property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
```

```
  <description>A base for other temporary directories.</description>
 </property>
 <property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:54310</value>
 </property>
</configuration>
```

4. /usr/local/hadoop/etc/hadoop/mapred-site.xml

Ανοίγουμε το αρχείο mapred-site.xml και γράφουμε τα ακόλουθα ανάμεσα στα tags του configuration:

$ vim /usr/local/hadoop/etc/hadoop/mapred-site.xml

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

5. /usr/local/hadoop/etc/hadoop/hdfs-site.xml

Το αρχείο hdfs-site.xml προσδιορίζει τους φακέλους που θα χρησιμοποιηθούν από τα στοιχεία των namenode και datanode. Γι αυτό τον λόγο πρέπει πρώτα να δημιουργήσουμε τους αντίστοιχους φακέλους.

$ sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode

$ sudo mkdir -p /usr/local/hadoop_store/hdfs/datanode

$ sudo chown -R hduser:hadoop /usr/local/hadoop_store

Στην συνέχεια ανοίγουμε το αρχείο hdfs-site.xml και προσθέτουμε τα ακόλουθα ανάμεσα στα tags του configuration:

$ vim /usr/local/hadoop/etc/hadoop/hdfs-site.xml

```
<configuration>
```

```
<property>
 <name>dfs.replication</name>
 <value>1</value>
 <description>Default block replication.
 The actual number of replications can be specified when the
 file is created. The default is used if replication is not
 specified in create time.
 </description>
</property>
<property>
  <name>dfs.namenode.name.dir</name>
  <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
</property>
</configuration>
```

Την πρώτη φορά που θα χρησιμοποιήσουμε το Hadoop πρέπει να κάνουμε format τον namenode του HDFS:

$ hadoop namenode -format

Ξεκινάμε το Hadoop και το HDFS (στον κατάλογο sbin του Hadoop):

$ ./start-all.sh

Μπορούμε να δούμε τις java διεργασίες που τρέχουν δίνοντας στο terminal την εντολή jps:

$ jps

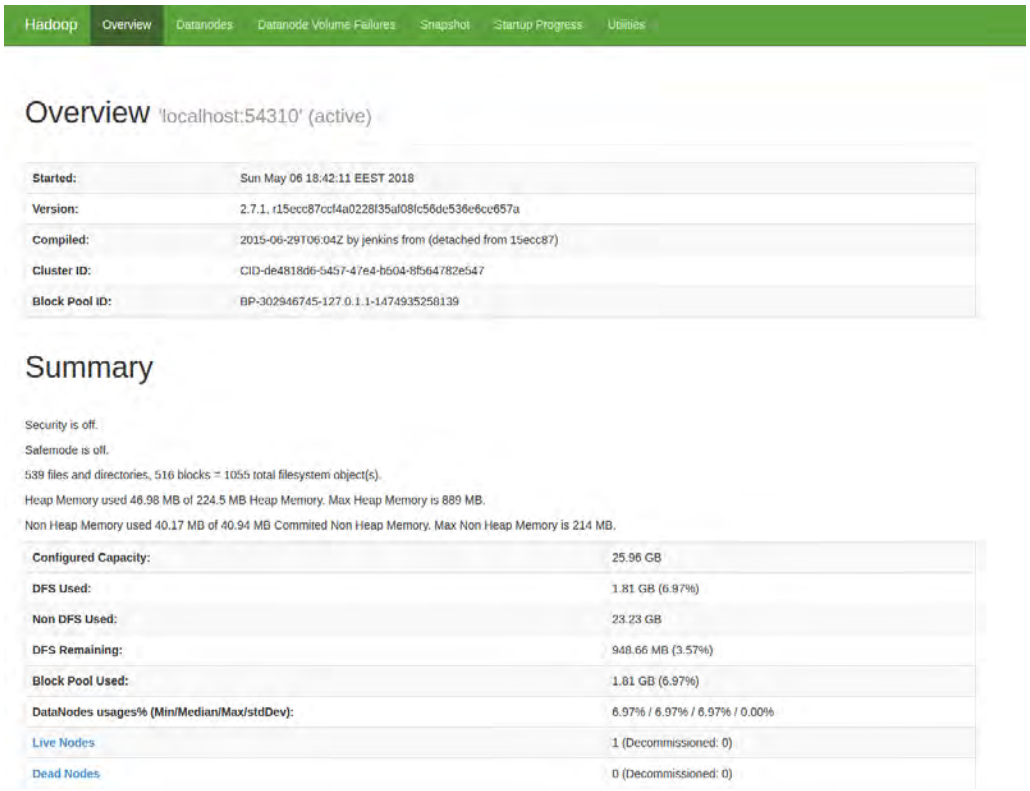
οπότε και θα πάρουμε ένα αποτέλεσμα σαν κι αυτό:

17050 ResourceManager
17204 NodeManager
17449 Jps
16473 NameNode
16654 DataNode
16882 SecondaryNameNode

Το σύστημα έχει ξεκινήσει και στο url http://localhost:50070 μπορεί κανείς να βλέπει την

κατάσταση του cluster όπως φαίνται και στο σχήμα 2.4. Για να σταματήσουμε το Hadoop, δίνουμε στο terminal την εντολή:

$ ./stop-all.sh



Σχήμα 2.4: Συνολική εικόνα της κατάστασης του Hadoop μέσω browser

Κεφάλαιο 3

Apache HBase

Ο αριθμός των εφαρμογών που αναπτύσσονται για να δουλεύουν με μεγάλα ποσά δεδομένων αυξήθηκε ραγδαία τα τελευταία χρόνια. Για να υποστηριχθεί αυτή η νέα σειρά εφαρμογών, έχουν αναπτυχθεί αρκετά νέα συστήματα διαχείρισης δεδομένων. Πολλά από αυτά τα νέα συστήματα είναι ανοιχτού κώδικα που αναπτύσσονται σε αρκετές μεγάλες εταιρείες. Η Apache HBase [4] είναι ένα τέτοιο σύστημα. Είναι μια κατανεμημένη βάση δεδομένων ανοιχτού κώδικα, βασισμένη στο Google Bigtable [5], και όλο και πιο δημοφιλής επιλογή βάσης δεδομένων για εφαρμογές που χρειάζονται γρήγορη τυχαία πρόσβαση σε μεγάλα ποσά δεδομένων. Η HBase είναι μια NoSQL βάση δεδομένων που χρησιμοποιεί το Hadoop και δουλεύει πάνω στο κατανεμημένο σύστημα αρχείων HDFS. Πριν συνεχίσουμε, όμως, ας δούμε τι είναι τα NoSQL συστήματα βάσεων δεδομένων, καθώς και τα βασικά χαρακτηριστικά τους.

3.1 NoSQL συστήματα βάσεων δεδομένων

Τα NoSQL συστήματα βάσεων δεδομένων (Not-only SQL) αποτελούν μια ομάδα μη σχεσιακών συστημάτων διαχείρισης δεδομένων, όπου οι βάσεις δεδομένων δεν βασίζονται κυρίως σε πίνακες και δεν χρησιμοποιούν SQL για τον χειρισμό των δεδομένων. Τα συστήματα διαχείρισης βάσεων δεδομένων NoSQL είναι χρήσιμα όταν είναι απαραίτητη η εργασία με πολύ μεγάλα ποσά δεδομένων, όπου σ αυτές τις περιπτώσεις, η φύση των δεδομένων δεν απαιτεί ένα σχεσιακό μοντέλο. Τα παραδοσιακά συστήματα είναι αποτελεσματικά για μικρούς αριθμούς εγγραφών δεδομένων, και εμφανίζουν αδυναμίες όταν τα δεδομένα αυξάνονται πολύ σε μέγεθος. Για παράδειγμα, η αναζήτηση πληροφορίας που είναι αποθηκευμένη σε πολλαπλούς πίνακες θα έχει σαν αποτέλεσμα την εκτέλεση διαδικασιών I/O που περιέχουν περιττά στοιχεία, καθώς η αναζήτηση θα μας επιστρέφει ολόκληρες γραμμές.

Τα νέα συστήματα αποθήκευσης δεδομένων δεν χρησιμοποιούν την SQL για διαχείριση δεδομένων, αλλά ένα διαφορετικό και απλούστερο περιβάλλον, το NoSQL. Με τη χρήση διαφόρων εργαλείων, ερωτήματα SQL μπορούν να μετατραπούν και να χρησιμοποιηθούν σε NoSQL αποθήκες δεδομένων. Η διαφορά αυτών των δύο μεθόδων βρίσκεται, κυρίως στο επίπεδο της αρχιτεκτονικής. Η NoSQL περιέχει πολλές διαστάσεις με τις οποίες μπορούμε να διακρίνουμε τα πλεονεκτήματα των μη σχεσιακών συστημάτων διαχείρισης βάσεων δεδομένων,

έναντι των αντίστοιχων σχεσιακών:

- Ελαστική επεκτασιμότητα (elastic scalability) - Στο παρελθόν, οι υπηρεσίες διαχείρισης βάσεων δεδομένων (Database administrator - DBA) έπρεπε να εξαρτώνται από την επεκτασιμότητα κάθε φορά που υπήρχε ανάγκη επέκτασης των δομών της βάσης. Αυτό σήμαινε την αγορά μεγαλύτερων εξυπηρετητών για την αντιμετώπιση του αυξανόμενου φορτίου δεδομένων. Οι βάσεις δεδομένων NoSQL προσφέρουν πολύ πιο εύκολη επιλογή εξαγωγής (οι βάσεις δεδομένων διανέμονται σε πολλούς προϋπάρχοντες κεντρικούς υπολογιστές). Με την αύξηση των απαιτήσεων διαθεσιμότητας και των ποσοστών συναλλαγής, η επεκτασιμότητα σε εικονικά περιβάλλοντα προσφέρει μια πιο οικονομική εναλλακτική λύση. Δεν είναι πολύ εύκολο να υπάρξει επέκταση και αναβάθμιση των RDBMS συστημάτων που είναι εγκατεστημένα σε συστάδες υπολογιστών, αλλά με NoSQL βάσεις δεδομένων, η επέκτασιμότητα είναι προγραμματισμένη από την δημιουργία της βάσης, ώστε να μπορούν να επεκταθούν για να συμπληρώσουν νέους κόμβους.

- Χρήση στα big data - Ενώ η χωρητικότητα των RDBMSs αυξήθηκε ώστε να ταιριάζει με τις απαιτήσεις των νέων όγκων δεδομένων (big data), παρ΄ όλα αυτά υπάρχει περιορισμός στον όγκο των δεδομένων που μπορεί να διαχειριστεί ένα μόνο RDBMS. Αντ άυτού, πολλοί άνθρωποι, εταιρίες ή οργανισμοί στρέφονται σε συστήματα NoSQL, όπως το Hadoop, για να χειριστούν τους όγκους των big data, καθώς αυτά ξεπερνούν τις δυνατότητες των πιο σημαντικών RDBMS.

- Μειωμένη εξάρτηση από ειδικούς διαχειριστές βάσεων δεδομένων - Ένα σημαντικό μειονέκτημα των RDBMS υψηλών προδιαγραφών είναι ότι η συντήρηση τους είναι δυνατή μόνο με τη χρήση εκπαιδευμένων διαχειριστών βάσεων δεδομένων (DBAs), οι οποίοι ασφαλώς δεν αποτελούν κάποια φθηνή λύση. Συγκεκριμένα, εμπλέκονται στο σχεδιασμό, την εγκατάσταση και την απόδοση των RDBMS, πράγμα που τους καθιστά σχεδόν απαραίτητους. Από την άλλη πλευρά, οι βάσεις δεδομένων NoSQL έχουν σχεδιαστεί για να απαιτούν λιγότερες απαιτήσεις στην διαχείριση, με χαρακτηριστικά όπως η διανομή δεδομένων, η αυτόματη επιδιόρθωση και τα απλοποιημένα μοντέλα δεδομένων. Ενώ κι σ αυτήν την περίπτωση πρέπει να υπάρχει ένας υπεύθυνος για τη διαχείριση των συστημάτων, οι εταιρίες που εφαρμόζουν αυτά τα συστήματα μπορούν να βασίζονται κυρίως στις απομακρυσμένες υπηρεσίες DBA, οι οποίες είναι φθηνότερες και λειτουργούν εξίσου καλά, αντί να επιβαρύνουν το κόστος διατήρησης και κατάρτισης, ενός διαχειριστή βάσεων δεδομένων (DBA).

- Χαμηλό κόστος - Οι NoSQL βάσεις δεδομένων έχουν σχεδιαστεί για να μπορούν να χρησιμοποιούνται σε συστάδες υπολογιστών που αποτελούνται από σχετικά φθηνά υπολογιστικά συστήματα. Οι σχεσιακές βάσεις δεδομένων, από την άλλη πλευρά, απαιτούν ακριβά συστήματα αποθήκευσης, πράγμα που σημαίνει μεγαλύτερο κόστος ανά όγκο αποθήκευσης δεδομένων.

- Χρήση ευέλικτων μοντέλων δεδομένων - Στα RDBMS μπορεί να δημιουργήθεί εύκολα κάποια σύγχυση ή γενικότερα προβλήματα, όταν προκύπτουν ανάγκες για αλλαγές

στην ευρύτερη διαχείριση της βάσης, ειδικά για τα μεγάλες εγκαταστάσεις βάσεων δεδομένων. Η ελάσσονος σημασίας αλλαγή πρέπει να παρακολουθείται προσεκτικά και ενδέχεται να συνεπάγεται κάποιο χρόνο διακοπής ή μείωση των επιπέδων υπηρεσιών. Η NoSQL δεν έχει τέτοιους περιορισμούς στα μοντέλα δεδομένων και ακόμη και οι πιο δύσχρηστες βάσεις δεδομένων NoSQL που βασίζονται στη δομή BigTable εξακολουθούν να επιτρέπουν σχετική ευελιξία, όπως η προσθήκη νέων στηλών χωρίς σημαντικές αναλύσεις.

Υπάρχουν τεσσερις τύποι NoSQL βάσεων δεδομένων, καθένας από τους οποίους με τα δικά του συγκεκριμένα χαρακτηριστικά:

- Βάση δεδομένων με βάση το σχήμα αποθήκευσης κλειδί-τιμή (key-value) - Αποτελούν τις πιο απλές περιπτώσεις, στις οποίες κάθε στοιχείο στην βάση αποθηκεύεται ως κάποιο όνομα χαρακτηριστικού (κλειδί - key) μαζί με την τιμή του (value). Παραδείγματα τέτοιων βάσεων είναι οι Riak, Voldemort και Redis.

- Βάση δεδομένων γράφων - Βασισμένες στην θεωρία γράφων, αυτές οι βάσεις δεδομένων σχεδιάζονται για βάσεις των οποίων τα δεδομένα αναπαριστώνται καλύτερα με την μορφή γράφων. Παραδείγματα τέτοιων βάσεων αποτελούν τα Neo4j και Titan.

- Στηλο-προσανατολισμένες βάσεις δεδομένων - Σ΄ αυτή την περίπτωση αντί για την αποθήκευση δεδομένων σε γραμμές, οι βάσεις είναι σχεδιασμένες για την αποθήκευση των δεδομένων ως τμήματα των στηλών. Ενώ αυτή η απλή περιγραφή ακούγεται σαν το αντίστροφο μιας παραδοσιακής βάσης δεδομένων, οι στηλο-προσανατολισμένες βάσεις δεδομένων προσφέρουν πολύ υψηλές επιδόσεις και εξαιρετικά κλιμακούμενη (scalable) αρχιτεκτονική. Παραδείγματα αποτελούν οι: HBase, BigTable και HyperTable.

- Βάση δεδομένων εγγράφων - Αποτελεί επέκταση της βασικής ιδέας κλειδιού-τιμής, όπου τα έγγραφα αποτελούν το δεύτερο σκέλος του ζευγαριού αντικαθιστόντας το πεδίο της τιμής. Κάθε κλειδί αντιστοιχεί σ΄ ένα έγγραφο. Τέτοιες γνωστές βάσεις δεδομένων είναι οι MongoDB και CouchDB.

Ο παρακάτω πίνακας περιγράφει συνοπτικά τα χαρακτηριστικά του κάθε τύπου NoSQL βάσης δεδομένων:

Πίνακας 3.1: Χαρακτηριστικά διαφόρων τύπως NoSQL βάσεων δεδομένων

Μοντέλο Δεδομένων	Απόδοση	Επεκτασιμότητα	Ευελιξία	Πολυπλοκότητα
Key-value store	Υψηλή	Υψηλή	Υψηλή	Χαμηλή
Column Store	Υψηλή	Υψηλή	Μέτρια	Χαμηλή
Document Store	Υψηλή	Υψηλή	Υψηλή	Χαμηλή
Graph Database	Μεταβλητή	Μεταβλητή	Υψηλή	Υψηλή

3.2 Αρχιτεκτονική της HBase

Η HBase διαφέρει αρκετά από τις παραδοσιακές σχεσιακές βάσεις δεδομενων όπως η MySQL, η PostgreSQL, η Oracle κ.α. κυρίως στην δομή της αρχιτεκτονικής τους καθώς και στα χαρακτηριστικά που παρέχει στις εφαρμογές που το χρησιμοποιούν. Στις στηλο-προσανατολισμένες (column-oriented) βάσεις δεδομένων τα δεδομένα ομαδοποιύνται σε στήλες και αποθηκεύονται στον δίσκο με αυτόν τρόπο, πράγμα που διαφέρει από τους παραδοσιακούς γραμμο-προσανατολισμένους (row-oriented) πίνακες όπου τα δεδομένα αποθηκεύονται σε ολόκληρες συνεχείς γραμμές, [4]. Ο λόγος που αποθηκεύουμε τα δεδομένα σε στήλες είναι το γεγονός ότι δεν είναι απαραίτητες όλες οι τιμές των δεδομένων, κάτι το οποίο είναι σύνηθες σε βεις δεδομένων που προοορίζονται για αναλυτικούς σκοπούς. Το κύριο πλεονέκτημα αυτής της τεχνικής είναι το μειωμένο I/O καθώς τα μη απαραίτητα δεδομεα αγνοούνται στην εγγραφή και κατ επέκταση και στην ανάγνωση. Επίσης, μιας και τα δεδομένα των στηλών μοιάζουν μεταξύ τους ως προς τη μορφή, η στηλο-προσανατολισμένη αποθήκευση δεδομένων είναι κατάλληλη για την εφαρμογή διαφόρων μεθόδων συμπίεσης δεδομέμων.

Η HBase ακολουθεί το στηλο-προσανατολισμένο μοντέλο αποθήκευσης δεδομένων, αλλά δεν λειτουργεί ακριβώς όπως οι στηλο-προσανατολισμένες βάσεις δεδομένων. Οι βάσεις αυτές ειδικεύονται κυρίως στην ανάλυση δεδομένων σε πραγματικό χρόνο, ενώ η HBase ειδικεύεται επιπλέον, στο να παρέχει πρόσβαση με τη μορφή κλειδιού-τιμής (key-value) σε καποια δεδομένα ή σε καποιο εύρος δεδομένων. Ο ευκολότερος τρόπος για να κανατοήσει καποιος το μοντέλο δεδομένων της HBase είναι μέσω της περιγραφής ενός τυπικού πίνακα που αποτελείται από γραμμές και στήλες. Παρακάτω παρουσιάζονται κάποιοι βασικοί όροι των πινάκων HBase:

- Πίνακας (Table) - Η HBase οργανώνει τα δεδομένα της σε πίνακες, τα ονόματα των οποίων πρέπει να αποτελούνται από αλφαριθμητικά (strings).

- Γραμμή (Row) - Μέσα σε έναν πίνακα, τα δεδομένα αποθηκεύονται σε γραμμές. Οι γραμμές προσδιορίζονται με μοναδικό τρόπο από το κλειδί της κάθε γραμμής. Τα κλειδιά των γραμμών δεν έχουν τύπο δεδομένων και αντιμετωπίζονται πάντα ως byte arrays.

- Οικογένεια στήλης (Column Family) - Τα δεδομένα σε μια γραμμή ομαδοποιούνται σε οικογένειες στηλών. Οι οικογένειες στηλών επηρεάζουν τη φυσική διάταξη των δεδομένων που είναι αποθηκευμένα στην HBase. Για το λόγο αυτό, πρέπει να οριστούν αρχικά, επειδή δεν ειναι εύκολη η μετέπειτα τροποποιήσή τους. Κάθε γραμμή σε έναν πίνακα έχει τις ίδιες οικογένειες στηλών, παρόλο που μια γραμμή δεν χρειάζεται να αποθηκεύει δεδομένα σε όλα τα πεδία των οικογένειων στηλών της.

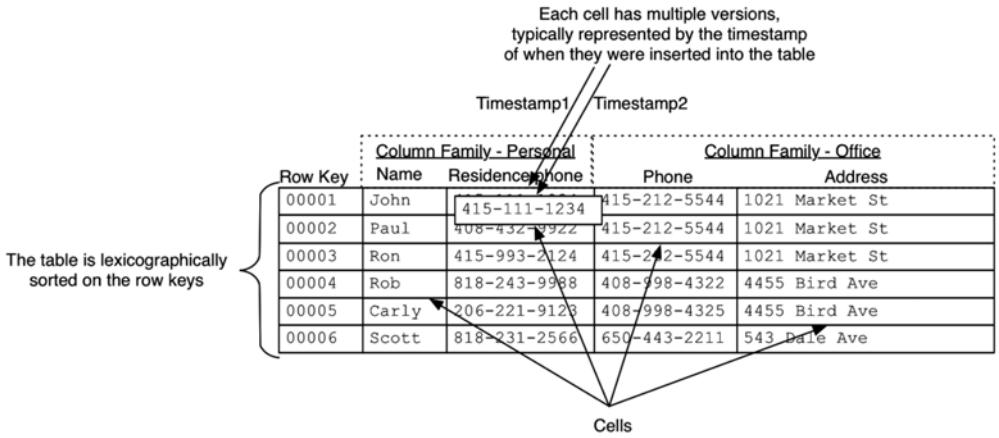
- Αναγνωριστικό στήλης (Column Qualifier) - Τα δεδομένα εντός μια οικογένειας στηλών προσδιορίζονται μέσω του αναγνωρστικού στήλης και πρέπει να οριστών κατα την αρχικοποίηση του πίνακα. Δεν είναι υποχρεωτικό να υπάρχει συνέπεια μεταξύ των αναγνωριστικών στηλών αναμεσα σε διαφορετικές γραμμές του πίνακα.

- Κελί (Cell) - Ο συνδυασμός ενός κλειδιού γραμμής, μιας οικογένειας στηλών κι ενός αναγνωριστικού στήλης, προσδιορίζει με μοναδικό τρόπο ένα κελί. Η τιμή ενός κελιού

(cell's value) αποτελεί ουσιαστικά τον συνδυασμό των δεδομένων από αυτά τα τρία στοιχεία.

- Χρονικό σημάδι (Timestamp) - Οι τιμές μέσα σ΄ ένα κελί προσδιορίζονται από έναν αριθμό έκδοσης, ο οποίος από προεπιλογή (default) είναι το χρονικό σημάδι (timestamp) κατά το οποίο γράφτηκε στο κελί.

Ένα παράδειγμα ενός πίνακα στην HBase φαίνεται στο σχήμα 3.1, όπου ο πίνακας αποτελείται από δύο οικογένειες στηλών (Personal και Office), κάθε μια από τις οποίες αποτελείται από δύο στήλες. Η οντότητα που περιέχει τα δεδομένα είναι το κελί. Οι γραμμές είναι ταξινομημένες σύμφωνα με το κλειδί της γραμμής, [8].



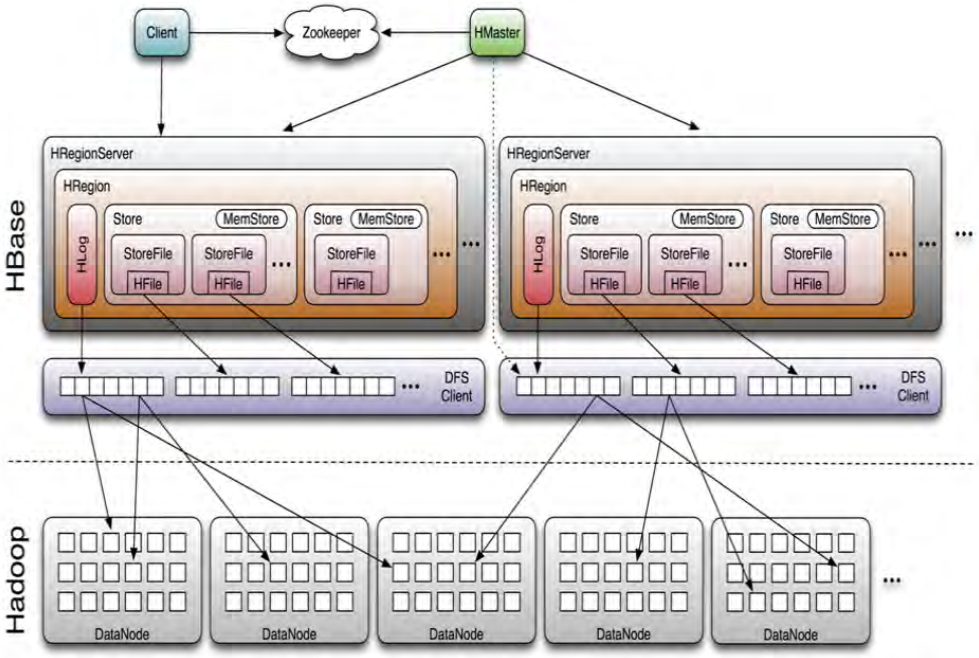
Σχήμα 3.1: Πίνακας στην HBase, [8]

Η αρχιτεκτονική της HBase παρουσιάζεται στο σχήμα 3.2. Αποτελείται από εξυπηρετητές (servers) δημιουργώντας σχέσεις εξάρτησης τύπου αφέντη - σκλάβου (master - slave). Τυπικά, μια συστάδα (cluster) HBase αποτελείται από έναν κόμβο master που λέγεται HMaster και πολλαπλούς κόμβους - εξυπηρετητές περιοχής (regions servers) που λέγονται HRegionServers. Κάθε HRegionServer αποτελείται, επίσης, από πολλούς Regions. Οι Regions είναι ουσιαστικά οι περιοχές στις οποίες θα αποθηκεύονται οι πίνακες της HBase. Όταν ένα πίνακας γίνεται υπερβολικά μεγάλος κατανέμεται σε διάφορους Regions. Σε κάθε HRegionServer φιλοξενείται περίπου ίσος αριθμός από Regions. Ο HMaster κόμβος της HBase είναι υπεύθυνος για:

- την εκτέλεση της συνολικής διαχείρισης
- διαχείριση και παρακολούθηση της συστάδας (cluster)
- την ορθή κατανομή των Regions στους διάφορους Region Servers
- τον έλεγχο και την εξισορρόπηση φόρτου των υπολογιστικών πόρων καθώς και την αντιμετώπιση πιθανών αποτυχιών του συστήματος

Από την άλλη πλευρά, οι HRegionServers είναι υπεύθυνοι για:

- την φιλοξενία και διαχείριση των Regions
- τον αυτόματο διαχωρισμό των Regions, όπου κρίνεται απαραίτητο
- τον χειρισμό των αιτήσεων διαβάσματος ή εγγραφής δεδομένων (read/write requests)
- την επικοινωνία με τους πελάτες (clients)



Σχήμα 3.2: Αρχιτεκτονική της HBase, [4]

3.3 Εγκατάσταση της HBase

Η έκδοση της HBase που χρησιμοποιήθηκε στην εργασία για την υλοποίηση της δομής είναι η 1.2.3 και η εγκατάσταση πραγματοποιήθηκε σε pseudo-distributed mode. Αρχικά πρέπει να καταβάσουμε το installation package που επιθυμούμε από ενα repository και να το κανουμε extract:

$ wget http://www-us.apache.org/dist/hbase/stable/hbase-1.2.3-bin.tar.gz

$ tar xvzf hbase-1.2.3-bin.tar.gz

Στην συνέχεια μεταφέρουμε τον φάκελο σε κάποιον κατάλογο του συστήματος που επιθυμούμε (κυρίως για λόγους οργάνωσης, εδώ στο usr/local/hbase)

$ sudo mv * /usr/local/hbase

Στο επόμενο βήμα πρέπει να βάλουμε τις κατάλληλες ρυθμίσεις στα configuration files της HBase:

1. ∼/.bashrc
2. /usr/local/hbase/conf/hbase-env.sh
3. /usr/local/hbase/conf/hbase-site.xml

1. ∼/.bashrc

Ανοίγουμε το αρχείο /.bashrc και προσθέτουμε τα παρακάτω στο τέλος:

$ vim ∼/.bashrc

.....
export HBASE_HOME=/usr/local/hbase
export PATH=$PATH:$HBASE_HOME/bin
.....

Αποθηκεύουμε το αρχείο και στη συνέχεια εκτελούμε από το terminal την εντολή:

$ source ∼/.bashrc

2. /usr/local/hbase/conf/hbase-env.sh

Ανοίγουμε το αρχείο hbase-env.sh και προσθέτουμε τα παρακάτω στο τέλος:

$ vim hbase-env.sh

.....
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export HBASE_REGIONSERVERS=$HBASE_HOME/conf/regionservers
export HBASE_MANAGES_ZK=true
.....

3. /usr/local/hbase/conf/hbase-site.xml

Ανοίγουμε το αρχείο hbase-site.xml και γράφουμε τα ακόλουθα ανάμεσα στα tags του configuration:

```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://localhost:54310/hbase</value>
```

```
  </property>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>
  <property>
    <name>hbase.zookeeper.property.clientPort</name>
    <value>2222</value>
  </property>
  <property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/home/hduser/zookeeper</value>
  </property>
</configuration>
```

Για να ξεκινήσουμε την HBase (αφού έχουμε ξεκινήσει το Hadoop) μεταφερόμαστε στον κατάλογο bin της HBase και γράφουμε στο terminal:

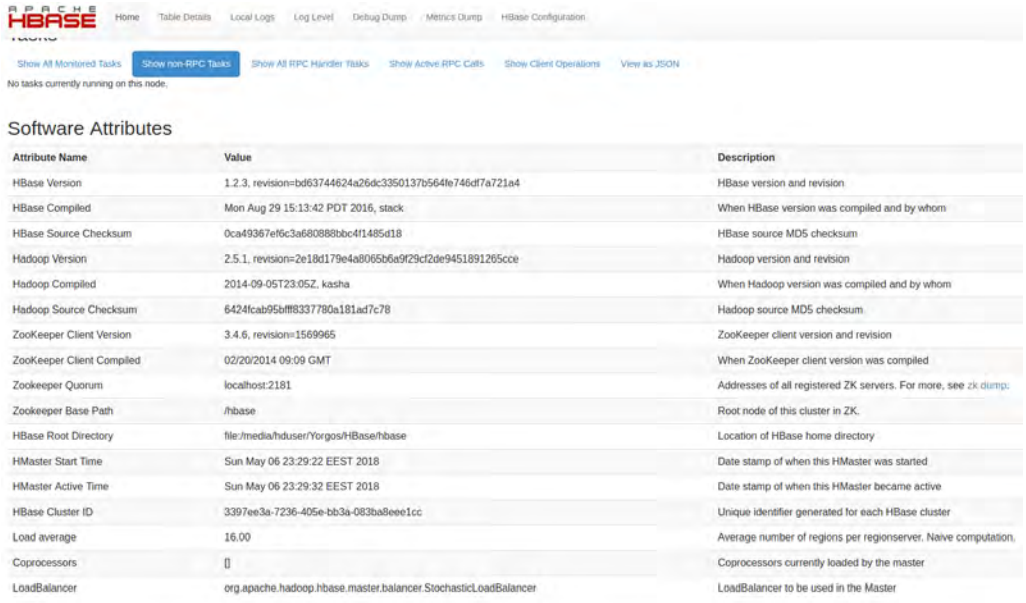
$ ./start-hbase.sh

Μπορούμε να ελεγξουμε εάν ξεκίνησε σωστά η HBase δίνοντας ξανά στο terminal την εντολή jps, οπότε και θα πάρουμε ένα αποτέλεσμα σαν κι αυτό:

17050 ResourceManager
17204 NodeManager
26710 **HMaster**
17449 Jps
16473 NameNode
16654 DataNode
16882 SecondaryNameNode

Το σύστημα έχει ξεκινήσει και στο url http://localhost:16010 μπορεί κανείς να βλέπει την κατάσταση της HBase όπως φαίνται και στο σχήμα 3.3. Για να σταματήσουμε την HBase, δίνουμε στο terminal την εντολή:

$ ./stop-hbase.sh



Σχήμα 3.3: Συνολική εικόνα της κατάστασης της HBase μέσω browser

Κεφάλαιο 4

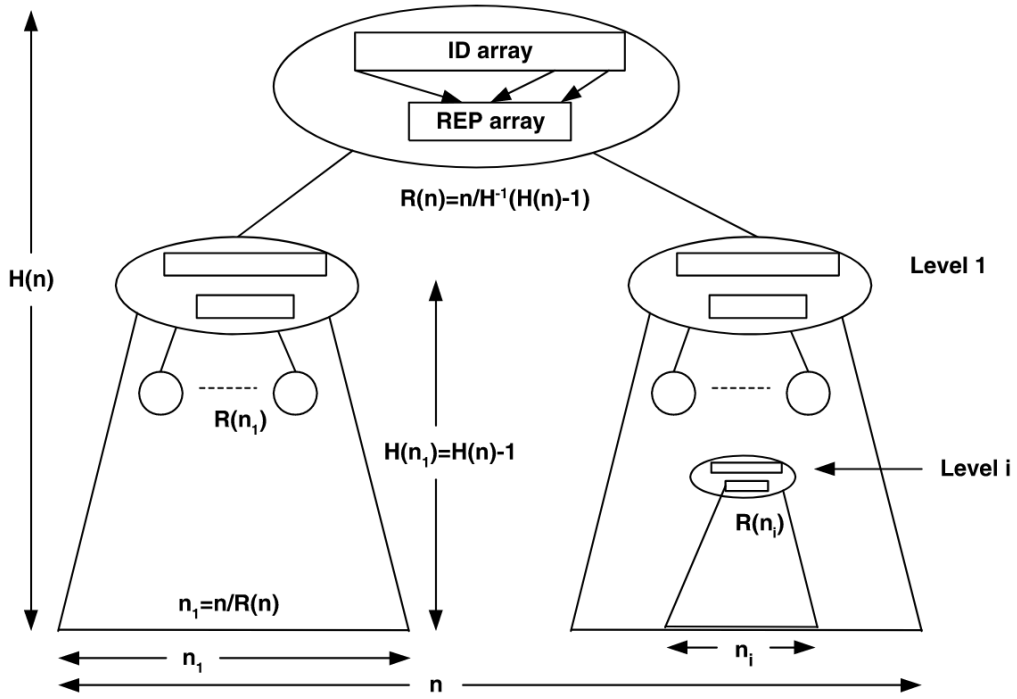
Υλοποίηση Interpolation Search Tree στην HBase

4.1 Σχετικές εργασίες

Στις εργασίες [7, 11, 10] παρουσιάζονται περιπτώσεις αναζήτησης με παρεμβολή σε μεγάλης κλίματας και κατανεμημένες υποδομές χιλιάδων κόμβων. Πιο συγκεκριμένα, στην ερευνητική εργασία [6], παρουσιάζεται η δημιουργία ενός B-δέντρου (ISB-tree), το οποίο χρησιμοποιεί αναζήτηση με παρεμβολή για την λειτουργία των διεργασιών της. Η δομή του δεντρου αναπτύσσεται σε δύο επίπεδα. Στο πρώτο, περιγράφεται η δημιουργία ενός στατικού δέντρου αναζήτησης με παρεμβολή [6], ενώ το δεύτερο επίπεδο αποτελείται από ένα *δάσος κάδων* (forest of buckets), καθένα από τα οποία υλοποιεί μια παραλλαγή του κλασικού B-δέντρου, το Lazy B-tree. Το Lazy B-tree καταφέρνει αναζήτηση με πολυπλοκότητα $O(\log_B n)$, όπου B είναι το μέγεθος των μπλοκς (blocksize) και n είναι το πλήθος των αποθηκευμένων στοιχείων, καθώς και διαδικασία ενημέρωσης σε $O(1)$. Η παρούσα πτυχιακή εργασία εστιάζεται κυρίως στο πρώτο επίπεδο υλοποίησης.

Ένα στατικό δέντρο αναζήτησης με παρεμβολή (Static Interpolation Search Tree - SIST) αποθηκεύει τα στοιχεία στα φύλλα του και μπορεί να χαρακτηριστεί από 3 συναρτήσεις H(n), R(n) και I(n). Η H(n) υποδεικνύει το ύψος του δέντρου, η R(n) αναφέρεται στον βαθμό της ρίζας του δέντρου και η I(n) σχετίζεται με την κατανομή των δεδομένων μέσα στο δέντρο. Έστω ότι ο κόμβος της ρίζας ενός SIST αντιστοιχεί σ΄ ένα ταξινομημένο αρχείο P μεγέθους n. Κάθε παιδί (κόμβος σε βάθος 1) θα αντιστοιχεί σ΄ ένα μέρος του αρχείου P, μεγέθους $n_1 = n/R(n)$ και ύψους $H(n_1) = H(n/R(n)) = H(n) - 1$. Κάθε εσωτερικός κόμβος n του δέντρου, σε βάθος i, σχετίζεται μ΄ έναν πίνακα REP[1...$R(n_i)$] ο οποίος περιέχει αντιπροσώπους των στοιχείων, έναν για κάθε ένα υποδέντρο. Επίσης, σχετίζεται και μ΄ έναν πίνακα ID[1...I(n_i)], ο οποίος αποτελείται από δείγματα στοιχείων του υποδέντρου με βάση την συνάρτηση αντίστροφης κατανομής. Έστω ότι l και u είναι η μικρότερη και η μεγαλύτερη τιμή αντίστοιχα σε κάθε υποδέντρο, τότε ο ρόλος του πίνακα ID είναι να κατανέμει το διάστημα [l,u] σε I(n_i) ίσα μέρη, κάθε ένα με μήκος $\frac{u-l}{I(n_i)}$. Ο ρόλος του πίνακα REP είναι να κατανέμει το συσχετιζόμενο υποαρχείο του P σε R ίσα υποαρχεία, κάθε ένα με μέγεθος $\frac{n_i}{R(n_i)}$.

Χρησιμοποιώντας τον πίνακα ID μπορούμε να κάνουμε interpolate στον πίνακα REP για να υπολογίσουμε το υποδέντρο στο οποίο θα συνεχιστεί η διαδικασία της αναζήτησης. Πιο συγκεκριμένα, για έναν πίνακα ID[1...I(n_i)] που σχετίζεται με έναν κόμβο v, μπορεί εύκολα να υπολογιστεί ότι $ID[i] = j$ εάν $REP[j] < l + i(u-l)/I(n_i) \leq REP[j+1]$. Με άλλα λόγια, ο πίνακας ID μας βοηθάει να κάνουμε interpolate τον πίνακα REP, υπολογίζοντας έτσι γρηγορότερα το κατάλληλο σημείο στο οποίο πρέπει να ξεκινήσουμε την αναζήτηση. Για παράδειγμα, έστω x το στοιχείο αναζήτησης. Ξεκινώντας από την ρίζα, σε κάθε κόμβο v του SIST σε βάθος i, υπολογίζουμε τον δείκτη $j = ID[\lfloor (I(n_i)(x-l)/(u-l)) \rfloor]$, έτσι ώστε να κάνουμε interpolate στο συσχετιζόμενο με τον κόμβο v, πίνακα REP, και διαδοχικά να κάνουμε αναζήτηση στον πίνκα REP από το στοιχείο REP[j+1] μέχρις ότου βρεθεί κατάλληλο παιδί για την συνέχεια της αναζήτησης, δηλαδή μέχρι να βρεθεί κάποιος δείκτης t έτσι ώστε $REP[j+1+t] < x \leq REP[j+2+t]$.

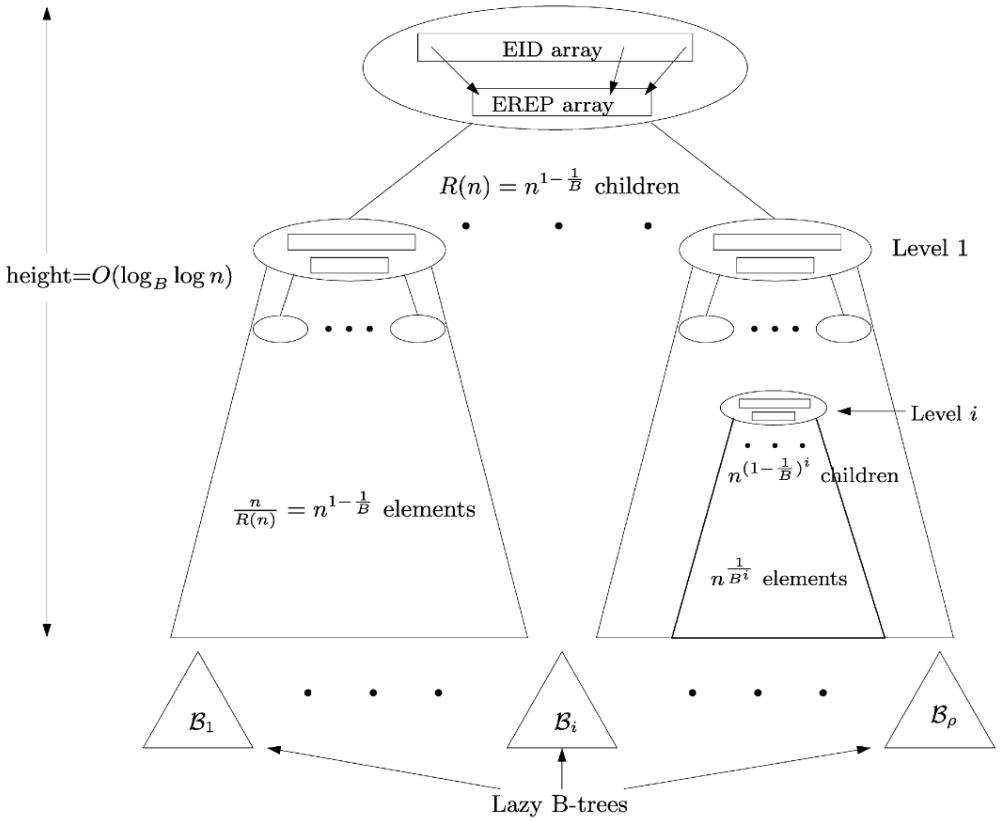


Σχήμα 4.1: Static Interpolation Search Tree - SIST, [6]

4.2 Ανάλυση - περιγραφή προβλήματος

Σύμφωνα με το [6], για την δημιουργία του πρώτου επιπέδου της δομή (SIST), ορίζονται κάποιες παράμετροι που θα μας βοηθήσουν στην ανάπτυξη του αλγορίθμου για την υλοποίηση της δομής του δέντρου αναζήτησης με παρεμβολή.

- $R(n) = n^{\delta}$, ο αριθμός των παιδιών ενός κόμβου (στην πράξη και για απλοποίηση της υλοποίησης θα το θεωρούμε ίσο με n καθώς το δ θα είναι πολύ κοντά στην μονάδα), όπου $\delta = 1 - \frac{1}{B}$, με B το blocksize.



Σχήμα 4.2: Το ISB-tree με n στοιχεία, [6]

- $I(n) = \frac{n}{(\log_2 \log_2 n)(1+\epsilon)}$, ο συνολικός αριθμός των στοιχείων του πίνακα ID, όπου $\epsilon > 0$

- $n = \frac{n_0}{B}$, όπου n_0 ο συνολικός αριθμός των αποθηκευμένων δεδομένων

Ουσιαστικά, η επιθυμητή δομή αποτελείται από έναν πίνακα με n_0 δεδομένα, ο οποίος χωρίζεται σε n κόμβους-buckets, έχοντας επίσης δύο ξεχωριστούς βοηθητικούς πίνακες, τους REP και ID, οι οποίοι επιτρέπουν την διαδικασία της παρεμβολής, μειώνοντας έτσι τον χρόνο αναζήτησής. Η αναζήτηση ξεκινάει από τον πίνακα ID, ο οποίος θα μας υποδείξει ενα συγκεκριμένο index του πίνακα REP, ο οποίος με την σειρά του μας υποδεικνύει τον συγκεκριμένο κόμβο-bucket του πίνακα στον οποίο πρεπει να ξεκινήσουμε την αναζήτηση, μειώνοντας έτσι τον συνολικό χρόνο αναζήτησης. Ο αλγόριθμος 1 παρουσιάζει την διαδικασία δημιουργίας των δύο πινάκων REP και ID και ο αλγόριθμος 2 δείχνει την διαδικασία αναζήτησης που ακολουθείται κάνοντας χρήση των δύο πινάκων, βρίσκοντας τελικά το στοιχείο αναζήτησης με την μέθοδο της παρεμβολής.

Algorithm 1 REP and ID arrays creation

```
1: u ← upper value of the bucket
2: l ← lower value of the bucket
3: V[ ] ← array with the stored elements
4: for j = 0 → numberOfBuckets do          ▷ min value of each bucket of the V array
5:     REP[j]=V[j * blockSize]
6: for k = 0 → I(n) do
7:     for z = 0 → numberOfBuckets do
8:         if (l + k * (u − l)/I(n)) > REP[z]) && ((l + k * (u − l)/I(n)) <= REP[z + 1]) then
9:             ID[k] = z
```

Algorithm 2 Search procedure

```
1: u ← upper value of the bucket
2: l ← lower value of the bucket
3: V[ ] ← array with the stored elements
4: el ← the element we search for
5: procedure CALCULATION OF MAIN PARAMETERS
6:     idIndex = ⌊I(n) * (el − l)/(u − l)      ▷ compute the index of ID array
7:     repIndex = ID[idIndex]                  ▷ index for the REP array
8:     bucketId = REP[repIndex]
9: for i = bucketId → numberOfStoredElements do
10:    if el = V[i] then
11:        elementfound
```

4.2.1 Δημιουργία δεδομένων

Ένας σημαντικός παράγοντας της υλοποίησης της δομής είναι τα δεδομένα που θα χρησιμοποιηθούν για τις διαδικασίες των πειραμάτων αναζήτησης. Για τον σκοπό αυτό δημιουργήθηκε πρόγραμμα σε C++, το οποίο δημιουργεί τυχαία δεδομένα σε αρχεία txt, έτσι ώστε στην συνέχεια να μεταφερθούν στο σύστημα HDFS και στην HBase με διαδιασία που θα περιγραφεί παρακάτω. Τα δεδομένα έχουν την μορφή του πίνακα 4.1, όπου κάθε εγγραφή περιγράφεται από τα ακόλουθα πεδία:

- ID
- Όνομα (Name)
- Επώνυμο (Surname)
- Όδός (Street)
- Αριθμός (Street number)

- Πόλη (City)
- Χώρα (Country)

Πίνακας 4.1: Πίνακας δεδομένων

ID	Name	Surname	Street	StreetNumber	City	Country
500000001	George	Sikotidis	Dodekanisou	7	Drama	Greece
500000002	Thanasis	Papakonstantinou	Andromedas	15	Larissa	Greece
500000003	Argiris	Mpakirtzis	Myrovolou	23	Kavala	Greece
500000004	Manos	Xatzidakis	Oneirwn	14	Athens	Greece
500000005	Belfort	Jordan	Wall	47	New York	USA

Το πρόγραμμα δημιουργεί τα ID σε αύξοντα ρυθμό και για κάθε ένα πεδίο, εκτός του StreetNumber (δημιουργείται απλά ένας τυχαίος αριθμός), ένα τυχαίο αλφαριθμητικό 10 χαρακτήρων προσομοιώνοντας έτσι το αντίστοιχο όνομα του πεδίου και διευκολύντας την δημιουργία αρκέτα μεγάλων αρχείων, απαραίτητα για τα πειράματα αναζήτησης. Το txt αρχείο εξόδου έχει την μορφή του σχήματος 4.3. Ο κώδικας παρατίθεται στο παράρτημα.

```
1  500000001,George,Sikotidis,Dodekanisou,7,Drama,Greece
2  500000002,VAOmRWhREe,IgnNWlGFGm,TbsUtLQTTQ,4,hyUxTIdsYy,YJiaWgngDj
3  500000003,rKZlQNCYnR,GvBHoDeBpV,MrULjjsjBC,41,KvCFQnPQkd,EEQnrmoYTD
4  500000004,VqwwuIUHPW,roHodbinxi,BRvGqaxMzY,22,ldLMeNxZmd,iDvNYGvQkW
5  500000005,zoWUJysEeG,zILHSGakAP,EaMuzCEPYE,49,FBUVwbweTB,XPGACdHNwM
6  500000006,mFeeLiVXvT,GRiONtVKsc,HmahzpnZdl,48,KgCkejdmVA,kWpbJhgTvM
7  500000007,nWyIfhMJDW,BPDNOEwzMz,ockkmXWGjE,16,uFDWcJAmPG,LxLIltqgPV
8  500000008,NIIXtairaD,MWNMVIuRYM,bwOOEtRjXx,4,qRlJIhJnot,xPgsMFMilx
9  500000009,mwmxZdkiYa,vfbokeJdZb,tcezyuyLKl,8,vFvMqXFmeZ,jMoyxbEiwn
10 500000010,ynECjJjhSN,AuTPqQlHht,JUULjXIwNK,50,QKlppKsBOY,xWAZmgBRTK
```

Σχήμα 4.3: Το csv αρχείο με τυχαία δεδομένα

4.2.2 Μεταφορά δεδομένων στο σύστημα HDFS και στην HBase

Μετά την δημιουργία των δεδομένων, θα πρέπει να γίνει η μεταφορά τους στο σύστημα HDFS και στην συνέχεια στην HBase με την δημιουργία των αντίστοιχων πινάκων. Έστω ότι στον κατάλογο /bin του φακέλου εγκατάστασης του Hadoop βρίσκεται το αρχείο data.txt με τα δεδομένα που δημιουργήσαμε στην προηγούμενη ενότητα, τότε για την μεταφορά του στον φάκελο /user/input του HDFS θα πρέπει να δωθεί στο terminal η ακόλουθη εντολή:

$ ./hadoop fs -copyFromLocal data.txt /user/input/data.txt

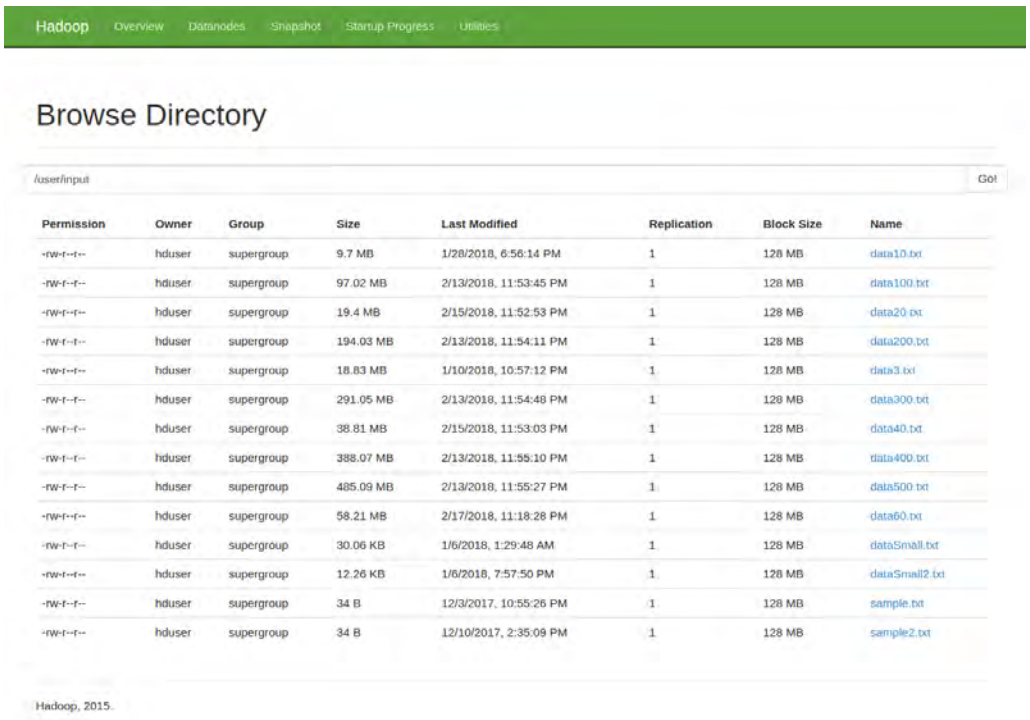
Για να ελέγξουμε ότι δημιουργήθηκε το αρχείο, καθώς και για να δούμε όλα τα αρχεία του καταλόγου input, δίνουμε την εντολή:

$ ./hadoop fs -ls /user/input

Σ΄ αυτή την περίπτωση το terminal θα επιστρέψει όλα τα αρχεία με κάποιες βασικές πληροφορίες όπως φαίνεται στο σχήμα 4.4. Το ίδιο αποτέλεσμα μπορούμε να δούμε και μέσα από τον browser, αφού μπούμε στο http://localhost:50070 και περιηγηθούμε στον αντίστοιχο φάκελο όπως φαίνεται και στο σχήμα 4.5

```
Found 15 items
-rw-r--r--   1 hduser supergroup    3391184 2018-04-27 00:34 /user/input/data.txt
-rw-r--r--   1 hduser supergroup   10173296 2018-01-28 18:56 /user/input/data10.txt
-rw-r--r--   1 hduser supergroup  101729542 2018-02-13 23:53 /user/input/data100.txt
-rw-r--r--   1 hduser supergroup   20345562 2018-02-15 23:52 /user/input/data20.txt
-rw-r--r--   1 hduser supergroup  203460394 2018-02-13 23:54 /user/input/data200.txt
-rw-r--r--   1 hduser supergroup   19745994 2018-01-10 22:57 /user/input/data3.txt
-rw-r--r--   1 hduser supergroup  305189120 2018-02-13 23:54 /user/input/data300.txt
-rw-r--r--   1 hduser supergroup   40691754 2018-02-15 23:53 /user/input/data40.txt
-rw-r--r--   1 hduser supergroup  406920059 2018-02-13 23:55 /user/input/data400.txt
-rw-r--r--   1 hduser supergroup  508650512 2018-02-13 23:55 /user/input/data500.txt
-rw-r--r--   1 hduser supergroup   61037600 2018-02-17 23:18 /user/input/data60.txt
-rw-r--r--   1 hduser supergroup      30785 2018-01-06 01:29 /user/input/dataSmall.txt
-rw-r--r--   1 hduser supergroup      12556 2018-01-06 19:57 /user/input/dataSmall2.txt
-rw-r--r--   1 hduser supergroup         34 2017-12-03 22:55 /user/input/sample.txt
-rw-r--r--   1 hduser supergroup         34 2017-12-10 14:35 /user/input/sample2.txt
```

Σχήμα 4.4: Εμφάνιση όλων των αρχείων που βρίσκονται στον φάκελο input του HDFS



Σχήμα 4.5: Εμφάνιση μέσω browser όλων των αρχείων που βρίσκονται στον φάκελο input του HDFS

Το επόμενο βήμα είναι να μεταφέρουμε τα δεδομένα στην HBase δημιουργώντας αρχικά τον αντίστοιχο πίνακα. Για την δημιουγία του πίνακα θα πρέπει να δώσουμε στο shell της HBase την ακόλουθη εντολή:

create 'data','name','surname','street','streetNumber','city','country'

όπου ορίζουμε το όνομα του πίνακα (data), καθώς και τα ονόματα των στηλών τα οποία θα πρέπει να αντιστοιχίζονται μ΄ εκείνα του txt αρχείου εισαγωγής. Το shell θα επιστρέψει αντίστοιχο μήνυμα επιτυχίας όπως φαίνεται στο σχήμα 4.6. Εάν θέλουμε να δούμε όλους του

```
hbase(main):001:0> create 'data','name','surname','street','streetNumber','city'
,'country'
0 row(s) in 1.8230 seconds
```

Σχήμα 4.6: Δημιουργία του πίνακα data στην HBase

πίνακες που υπάρχουν μέσα στην HBase, μπορούμε απλά να δώσουμε την εντολή list στο shell της HBase, οπότε θα επιστραφεί στον χρήστη μια λίστα όπως π.χ. αυτή του σχήματος 4.7. Διαφορετικά, μπορούμε να δούμε όλους τους πίνακες που έχουμε δημιουργήσει στην

```
hbase(main):002:0> list
TABLE
IDhbase
REPhbase
data
data10
data100
data20
data200
data300
data40
data400
data500
data60
data80
13 row(s) in 0.0300 seconds

=> ["IDhbase", "REPhbase", "data", "data10", "data100", "data20", "data200", "data300", "data40", "data40
0", "data500", "data60", "data80"]
```

Σχήμα 4.7: Λίστα με όλους τους πίνακες που έχουμε δημιουργήσει στην HBase

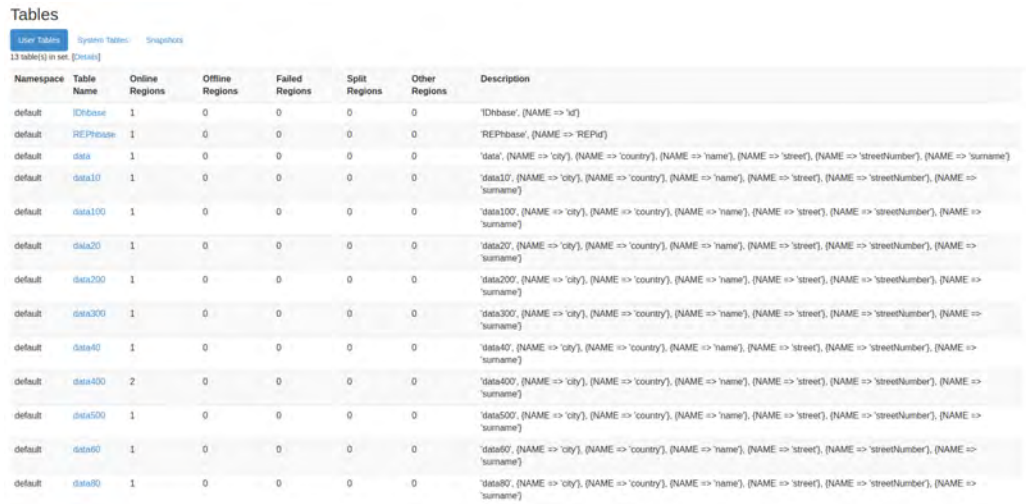
HBase, μαζί με περισσότερες πληροφορίες, μέσα από οποιονδήποτε browser στην αντίστοιχη διεύθυνση και πόρτα (http://localhost:16010/master-status) όπως φαίνεται στο σχήμα 4.8. Για την μεταφορά των δεδομένων από το αρχείο data.txt στον αντίστοιχο πίνακα data της HBase πρέπει να δώσουμε την παρακάτω εντολή:

$ ./hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=”,” -Dimporttsv.columns=HBASE_ROW_KEY,name,surname,street,streetNumber,city, country data hdfs://localhost:54310/user/input/data.txt

Εάν εκτελεστεί επιτυχώς θα επιστραφεί αντίστοιχο μήνυμα όπως αυτό της εικόνας 4.9, όπου αναφέρονται με λεπτομέρειες όλες οι διαδικασίες MapReduce που πραγματοποιήθηκαν από το σύστημα για την ορθή υλοποίηση της εντολής. Στην συνέχεια, μπορούμε να δούμε τα δεδομένα του πίνακα δίνοντας στο shell της HBase την εντολή:

scan ’data’

το αποτέλεσμα της οποίας φαίνεται στο σχήμα 4.10.



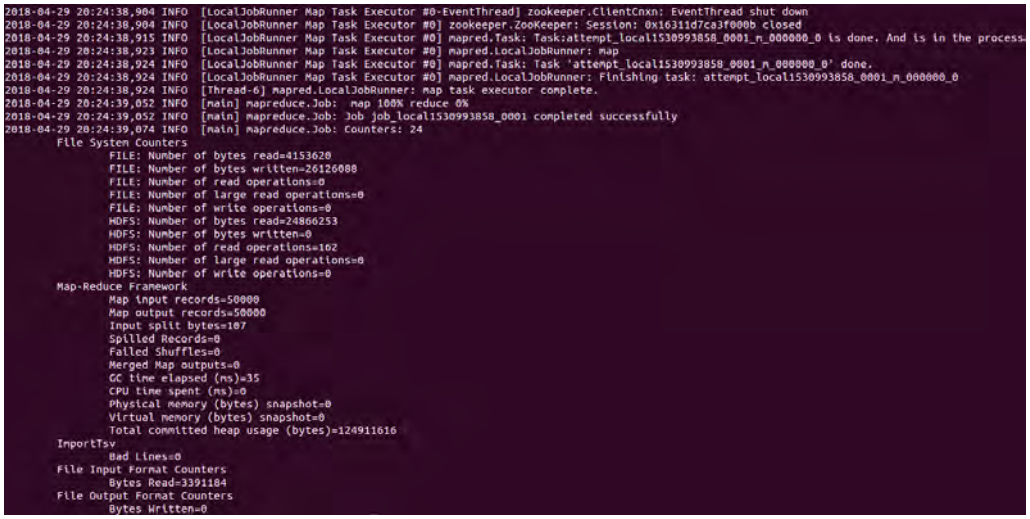
Tables

User Tables | System Tables | Snapshots

13 table(s) in set. [Details]

Namespace	Table Name	Online Regions	Offline Regions	Failed Regions	Split Regions	Other Regions	Description
default	IDhbase	1	0	0	0	0	'IDhbase', {NAME => 'id'}
default	REPhbase	1	0	0	0	0	'REPhbase', {NAME => 'REPid'}
default	data	1	0	0	0	0	'data', {NAME => 'city'}, {NAME => 'country'}, {NAME => 'name'}, {NAME => 'street'}, {NAME => 'streetNumber'}, {NAME => 'surname'}
default	data10	1	0	0	0	0	'data10', {NAME => 'city'}, {NAME => 'country'}, {NAME => 'name'}, {NAME => 'street'}, {NAME => 'streetNumber'}, {NAME => 'surname'}
default	data100	1	0	0	0	0	'data100', {NAME => 'city'}, {NAME => 'country'}, {NAME => 'name'}, {NAME => 'street'}, {NAME => 'streetNumber'}, {NAME => 'surname'}
default	data20	1	0	0	0	0	'data20', {NAME => 'city'}, {NAME => 'country'}, {NAME => 'name'}, {NAME => 'street'}, {NAME => 'streetNumber'}, {NAME => 'surname'}
default	data200	1	0	0	0	0	'data200', {NAME => 'city'}, {NAME => 'country'}, {NAME => 'name'}, {NAME => 'street'}, {NAME => 'streetNumber'}, {NAME => 'surname'}
default	data300	1	0	0	0	0	'data300', {NAME => 'city'}, {NAME => 'country'}, {NAME => 'name'}, {NAME => 'street'}, {NAME => 'streetNumber'}, {NAME => 'surname'}
default	data40	1	0	0	0	0	'data40', {NAME => 'city'}, {NAME => 'country'}, {NAME => 'name'}, {NAME => 'street'}, {NAME => 'streetNumber'}, {NAME => 'surname'}
default	data400	2	0	0	0	0	'data400', {NAME => 'city'}, {NAME => 'country'}, {NAME => 'name'}, {NAME => 'street'}, {NAME => 'streetNumber'}, {NAME => 'surname'}
default	data500	1	0	0	0	0	'data500', {NAME => 'city'}, {NAME => 'country'}, {NAME => 'name'}, {NAME => 'street'}, {NAME => 'streetNumber'}, {NAME => 'surname'}
default	data60	1	0	0	0	0	'data60', {NAME => 'city'}, {NAME => 'country'}, {NAME => 'name'}, {NAME => 'street'}, {NAME => 'streetNumber'}, {NAME => 'surname'}
default	data80	1	0	0	0	0	'data80', {NAME => 'city'}, {NAME => 'country'}, {NAME => 'name'}, {NAME => 'street'}, {NAME => 'streetNumber'}, {NAME => 'surname'}

Σχήμα 4.8: Λίστα με όλους τους πίνακες που έχουμε δημιουργήσει στην HBase μέσα από browser



```
2018-04-29 20:24:38,904 INFO  [LocalJobRunner Map Task Executor #0-EventThread] zookeeper.ClientCnxn: EventThread shut down
2018-04-29 20:24:38,904 INFO  [LocalJobRunner Map Task Executor #0] zookeeper.ZooKeeper: Session: 0x16311d7ca3f000b closed
2018-04-29 20:24:38,915 INFO  [LocalJobRunner Map Task Executor #0] mapred.Task: Task:attempt_local1530993858_0001_m_000000_0 is done. And is in the process
2018-04-29 20:24:38,923 INFO  [LocalJobRunner Map Task Executor #0] mapred.LocalJobRunner: map
2018-04-29 20:24:38,924 INFO  [LocalJobRunner Map Task Executor #0] mapred.Task: Task 'attempt_local1530993858_0001_m_000000_0' done.
2018-04-29 20:24:38,924 INFO  [LocalJobRunner Map Task Executor #0] mapred.LocalJobRunner: Finishing task: attempt_local1530993858_0001_m_000000_0
2018-04-29 20:24:38,924 INFO  [Thread-6] mapred.LocalJobRunner: map task executor complete.
2018-04-29 20:24:39,052 INFO  [main] mapreduce.Job:  map 100% reduce 0%
2018-04-29 20:24:39,052 INFO  [main] mapreduce.Job: Job job_local1530993858_0001 completed successfully
2018-04-29 20:24:39,074 INFO  [main] mapreduce.Job: Counters: 24
        File System Counters
                FILE: Number of bytes read=4153620
                FILE: Number of bytes written=26126088
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=24866253
                HDFS: Number of bytes written=0
                HDFS: Number of read operations=102
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=0
        Map-Reduce Framework
                Map input records=50000
                Map output records=50000
                Input split bytes=107
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=35
                CPU time spent (ms)=0
                Physical memory (bytes) snapshot=0
                Virtual memory (bytes) snapshot=0
                Total committed heap usage (bytes)=124911616
        ImportTsv
                Bad Lines=0
        File Input Format Counters
                Bytes Read=3391184
        File Output Format Counters
                Bytes Written=0
```

Σχήμα 4.9: Επιτυχές μήνυμα για την εισαγωγή δεδομένω μέσω MapReduce μεθόδων στην HBase

4.2.3 Δημιουργία του IST

Η δομή του IST αποτελείται από δύο πίνακες αποθηκευμένους στην HBase, τον IDhbase και τον REPhbase. Ο πίνακας IDhbase περιέχει δείκτες που δείχνουν στον πίνακα REPhbase και ο τελευταίος περιέχει δείκτες που δείχνουν στο dataset της αναζήτησής μας. Αυτό φαίνεται στα σχήματα 4.11 και 4.12 αντίστοιχα.

Η διαδικασία αναζήτησης ξεκινάει από τον IDhbase, όπου προσδιορίζεται αρχικά ένας δείκτης, ο $idIndex$, ο οποίος ισούται με:

$$idIndex = \lfloor I(n) * (el - l)/(u - l) \tag{4.1}$$

όπου:

```
hbase(main):001:0> scan 'data'
ROW                                   COLUMN+CELL
 500000001                            column=city:, timestamp=1525022665786, value=Drama
 500000001                            column=country:, timestamp=1525022665786, value=Greece
 500000001                            column=name:, timestamp=1525022665786, value=George
 500000001                            column=street:, timestamp=1525022665786, value=Dodekanisou
 500000001                            column=streetNumber:, timestamp=1525022665786, value=7
 500000001                            column=surname:, timestamp=1525022665786, value=Sikotidis
 500000002                            column=city:, timestamp=1525022665786, value=hyUxTIdsYy
 500000002                            column=country:, timestamp=1525022665786, value=YJiaWgngDj
 500000002                            column=name:, timestamp=1525022665786, value=VAOmRWhREe
 500000002                            column=street:, timestamp=1525022665786, value=TbsUtLQTTQ
 500000002                            column=streetNumber:, timestamp=1525022665786, value=4
 500000002                            column=surname:, timestamp=1525022665786, value=IgnNWlGFGm
 500000003                            column=city:, timestamp=1525022665786, value=KvCFQnPQkd
 500000003                            column=country:, timestamp=1525022665786, value=EEQnrmoYTD
 500000003                            column=name:, timestamp=1525022665786, value=rKZlQNCYnR
 500000003                            column=street:, timestamp=1525022665786, value=MrULjjsjBC
 500000003                            column=streetNumber:, timestamp=1525022665786, value=41
 500000003                            column=surname:, timestamp=1525022665786, value=GvBHoDeBpV
```

Σχήμα 4.10: Τα πρώτα τρία στοιχεία του πίνακα data της HBase

```
20                                    column=id:id, timestamp=1525974331859, value=63
21                                    column=id:id, timestamp=1525974331867, value=66
22                                    column=id:id, timestamp=1525974331873, value=69
23                                    column=id:id, timestamp=1525974331878, value=72
24                                    column=id:id, timestamp=1525974331880, value=76
25                                    column=id:id, timestamp=1525974331882, value=79
26                                    column=id:id, timestamp=1525974331885, value=82
27                                    column=id:id, timestamp=1525974331888, value=85
28                                    column=id:id, timestamp=1525974331890, value=88
29                                    column=id:id, timestamp=1525974331892, value=92
```

Σχήμα 4.11: Ο πίνακας IDhbase της HBase

```
60                                    column=REPid:REPid, timestamp=1525974336812, value=500061441
61                                    column=REPid:REPid, timestamp=1525974336814, value=500062465
62                                    column=REPid:REPid, timestamp=1525974336816, value=500063489
63                                    column=REPid:REPid, timestamp=1525974336819, value=500064513
64                                    column=REPid:REPid, timestamp=1525974336821, value=500065537
65                                    column=REPid:REPid, timestamp=1525974336823, value=500066561
66                                    column=REPid:REPid, timestamp=1525974336826, value=500067585
67                                    column=REPid:REPid, timestamp=1525974336828, value=500068609
68                                    column=REPid:REPid, timestamp=1525974336831, value=500069633
69                                    column=REPid:REPid, timestamp=1525974336833, value=500070657
```

Σχήμα 4.12: Ο πίνακας REPhbase της HBase

$I(n)$: ο συνολικός αριθμός των στοιχείων του πίνακα IDhbase
el: το στοιχείο που ψάχνουμε
l: το μικρότερο στοιχείο από το dataset
u: το μεγαλύτερο στοιχείο από το dataset

Έτσι βρίσκουμε το στοιχείο $IDhbase[idIndex] = repIndex$, το οποίο θα μας οδηγήσει στο τελικό $bucketID = REPhbase[repIndex]$, απ΄ όπου και μπορούμε πλέον να ξεκινήσουμε την διαδιακασία της αναζήτησης, έχοντας τελικά κάνει παρεμβολή στο αρχικό dataset.

Κεφάλαιο 5

Περιγραφή των πειραμάτων

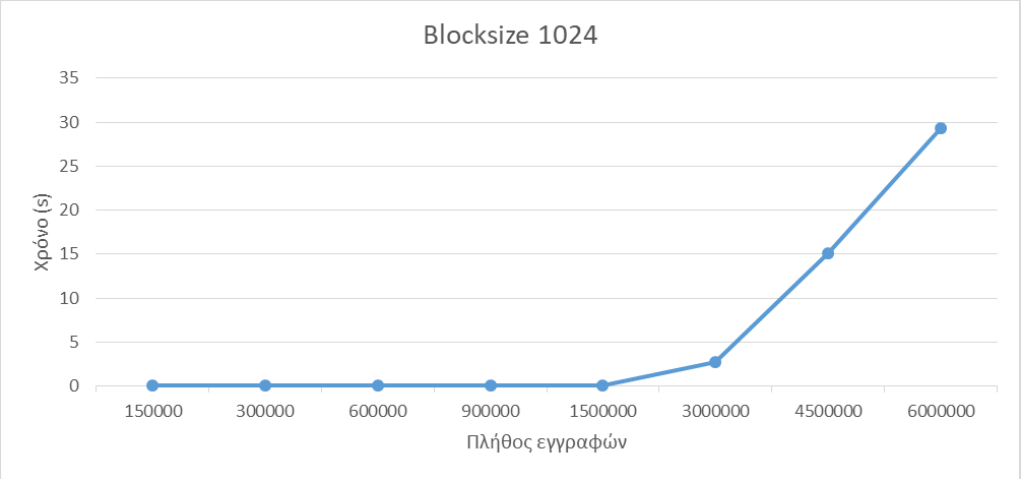
5.1 Πειράματα και χρόνοι αναζήτησης

Στο κεφάλαιο αυτό γίνεται περιγραφή των πειραμάτων που πραγματοποιήθηκαν καθώς και των αποτελεσμάτων που λαμβάνονται από αυτά. Ο κύριος στόχος των μετρήσεων είναι ο υπολογισμός του μέσου χρόνου αναζήτησης ενός στοιχείου, σε datasets (πίνακες) διαφόρων μεγεθών. Στον πίνακα 5.1 φαίνοται οι πίνακες της HBase που χρησιμοποιήθηκαν μαζί με τα βασικά χαρακτηριστικά τους. Το μέγεθος των πινάκων στην HBase είναι μεγαλύτερο σε σχέση με τα txt αρχεία, καθώς για την αποθήκευση των δεδομένων στην βάση απαιτούνται κι άλλα στοιχεία για κάθε εγγραφή όπως π.χ. τα ονόματα των στηλών, τα timestamps κ.α.. Η αναζήτηση στους πίνακες έγινε με βάση το ID της κάθε εγγραφής. Βασική παράμετρος για την διεξαγωγή των πειραμάτων είναι το Blocksize. Οι διάφορες τιμές του Blocksize για τις οποίες έγινε έλεγχος στους χρόνους αναζήτησης είναι 1024, 2048, 4096, 8192, 16384, 32768 και 65536. Για την ορθή μέτρηση των χρόνων, πήραμε μέσω μιας random συνάρτησης 500 διαφορετικές περιπτώσεις τιμών ID από τις εγγραφές των πινάκων, εκτελώντας ισάριθμα πειράματα. Στα σχήματα 5.1 έως 5.7 παρουσιάζονται τα διαγράμματα των χρόνων αναζήτησης για κάθε μία διαφορετική περίπτωση του Blocksize και στα σχήματα 5.8 έως 5.16 είναι τα ίδια διαγράμματα αλλά με λογαριθμική κλίμακα. Ο πίνακας 5.2 παρουσιάζει συγκεντρωτικά όλους τους χρόνους που μετρήθηκαν.

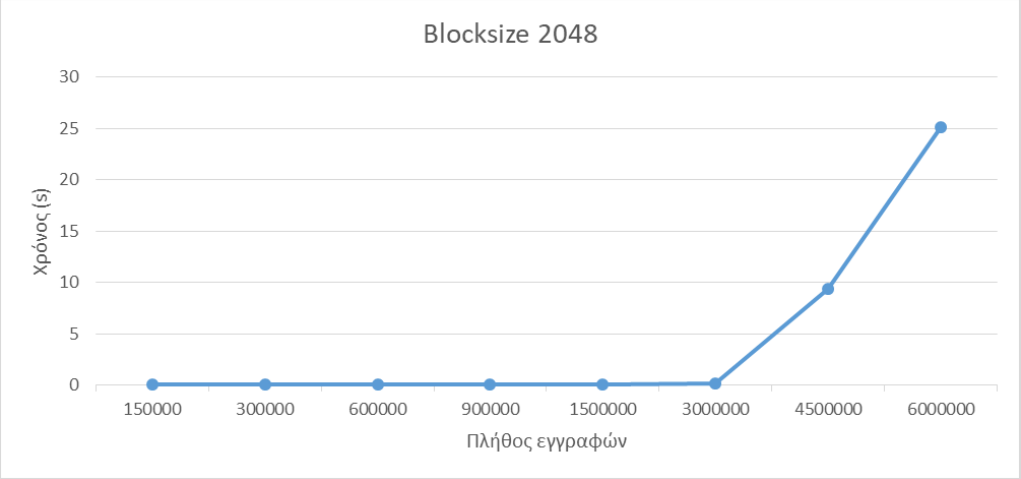
	Πλήθος εγγραφών	**Μέγεθος στον δίσκο (txt)**	**Μέγεθος στην HBase**
Πίνακας 1	150.000	10 MB	46 MB
Πίνακας 2	300.000	20 MB	92 MB
Πίνακας 3	600.000	40 MB	184 MB
Πίνακας 4	900.000	60 MB	276 MB
Πίνακας 5	1.500.000	100 MB	460 MB
Πίνακας 6	3.000.000	200 MB	920 MB
Πίνακας 7	4.500.000	300 MB	1.4 GB
Πίνακας 8	6.000.000	400 MB	1.8 GB

Πίνακας 5.1: Τα μεγέθη των πινάκων που χρησιμοποιήθηκαν στα πειράματα

Από τις μετρήσεις μπορούμε να παρατηρήσουμε μια λογική αύξηση του χρόνου καθώς αυξάνεται το πλήθος των εγγραφών. Παρ΄ όλα αυτά, το Blocksize παίζει καθοριστικό ρόλο. Ιδιαίτερα στην περίπτωση όπου το Blocksize ισούται με 8192 οι χρόνοι αναζήτησης μεταξύ του μικρότερου και του μεγαλύτερου μεγέθους πίνακα των εγγραφών παρουσιάζουν πάρα πολύ μικρή διαφορά. Από εκεί και πάνω παρατηρείται ξανά μια αύξηση του χρόνου καθώς μεγαλάνει το μέγεθος του dataset. Στις περιπτώσεις των 1024 και 2048, ο χρόνος αυξάνεται εκθετικά όταν το πλήθος των εγγραφών ξεπερνά τα 3.000.000 περίπου.



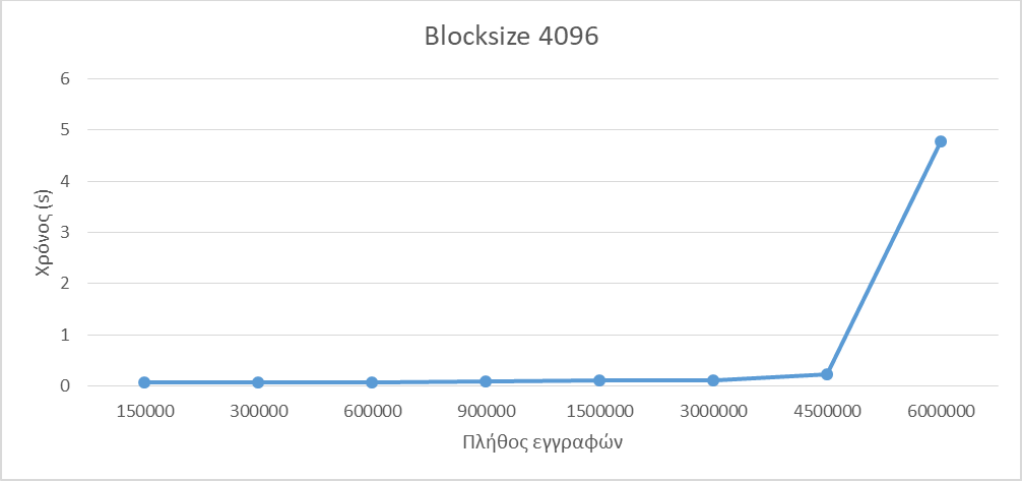
Σχήμα 5.1: Μέσος χρόνος αναζήτησης για Blocksize 1024



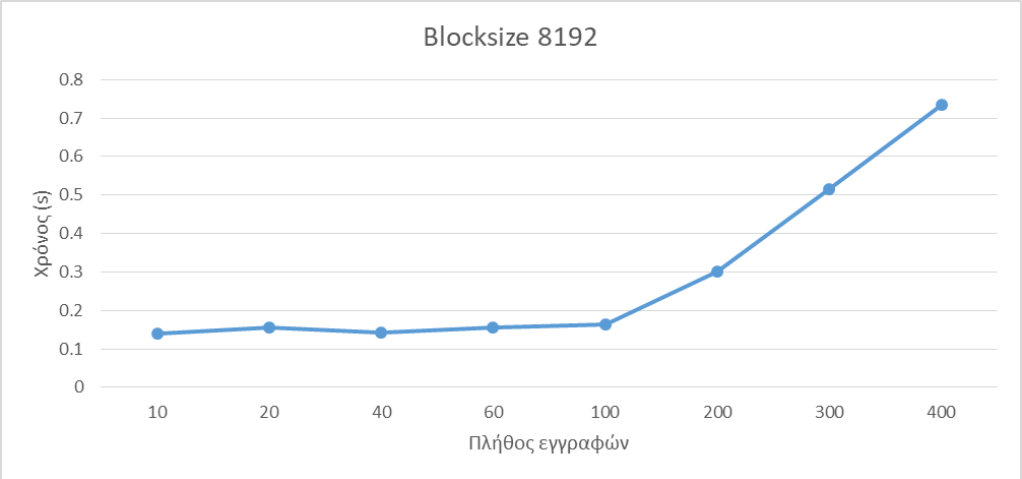
Σχήμα 5.2: Μέσος χρόνος αναζήτησης για Blocksize 2048

5.2 Χρόνοι δημιουργίας των πινάκων IDhbase και REPhbase

Όσον αφορά τους χρόνους δημιουργίας των πινάκων IDhbase και REPhbase, κυμαίνονται σχεδόν πάντα στα ίδια επίπεδα, γύρω στα 5 με 7 seconds. Μόνο στις περιπτώσεις όπου το

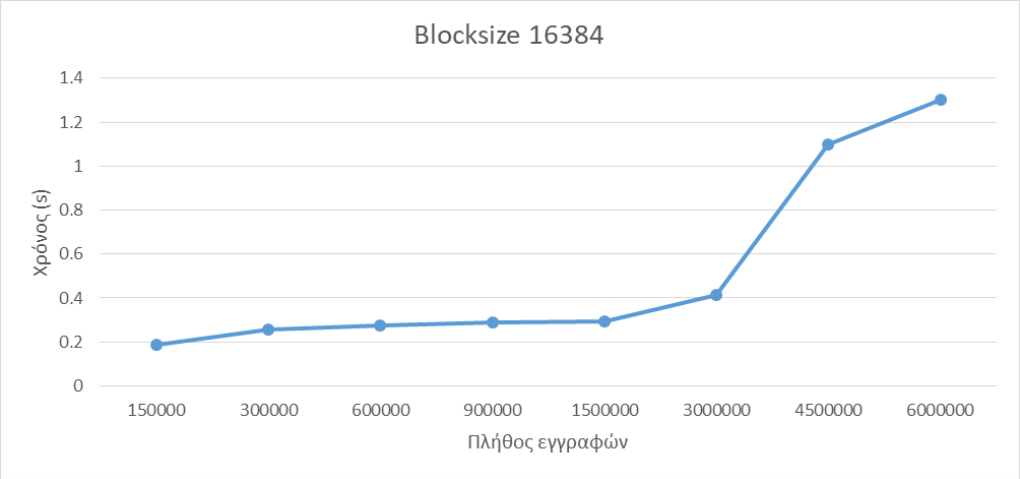


Σχήμα 5.3: Μέσος χρόνος αναζήτησης για Blocksize 4096

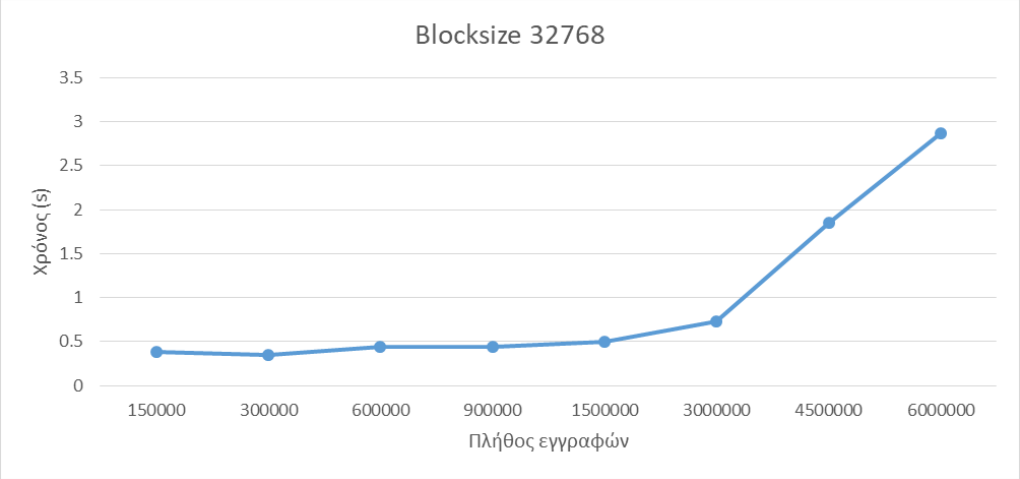


Σχήμα 5.4: Μέσος χρόνος αναζήτησης για Blocksize 8192

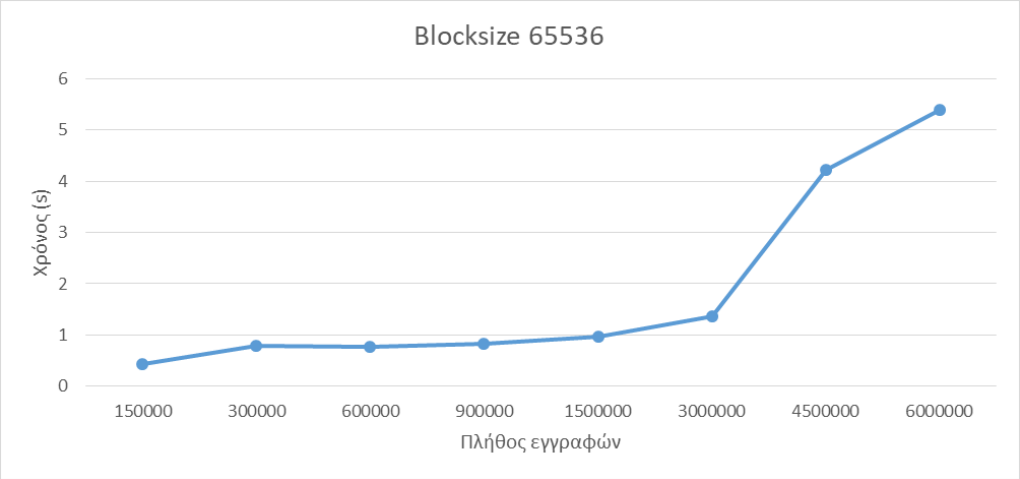
Blocksize είναι ίσο με 1024 και 2048 και το πλήθος των εγγραφών είναι πάνω από 3.000.000 παρατηρείται μια διαφορά σε σχέσή με όλες τις υπόλοιπες περιπτώσεις.



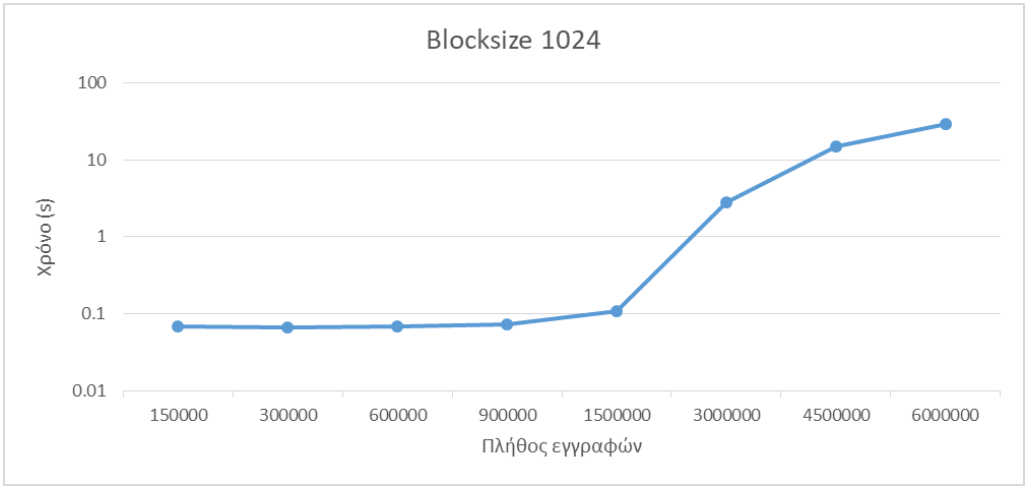
Σχήμα 5.5: Μέσος χρόνος αναζήτησης για Blocksize 16384



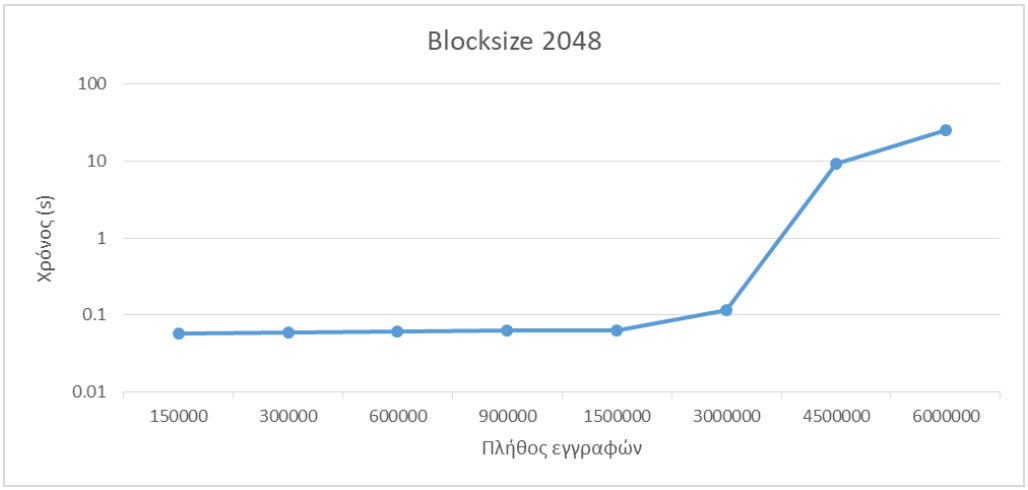
Σχήμα 5.6: Μέσος χρόνος αναζήτησης για Blocksize 32768



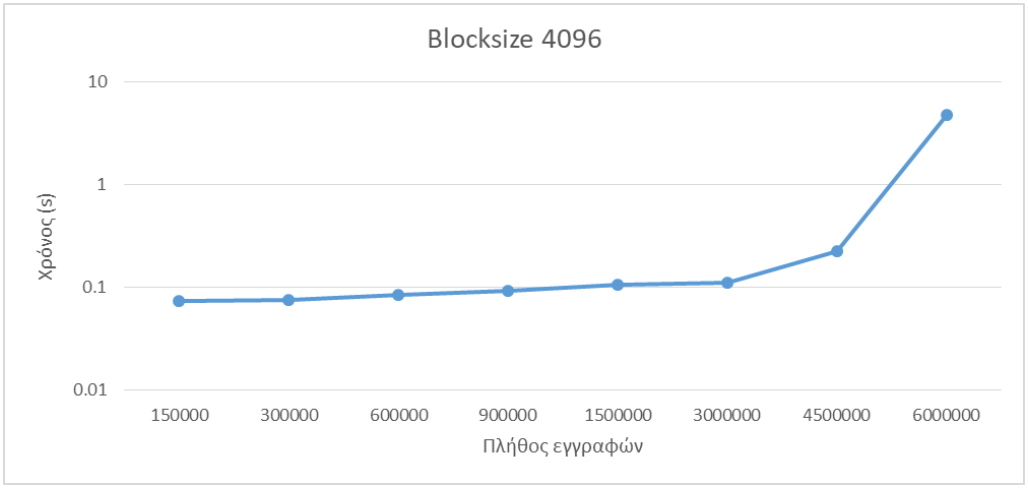
Σχήμα 5.7: Μέσος χρόνος αναζήτησης για Blocksize 65536



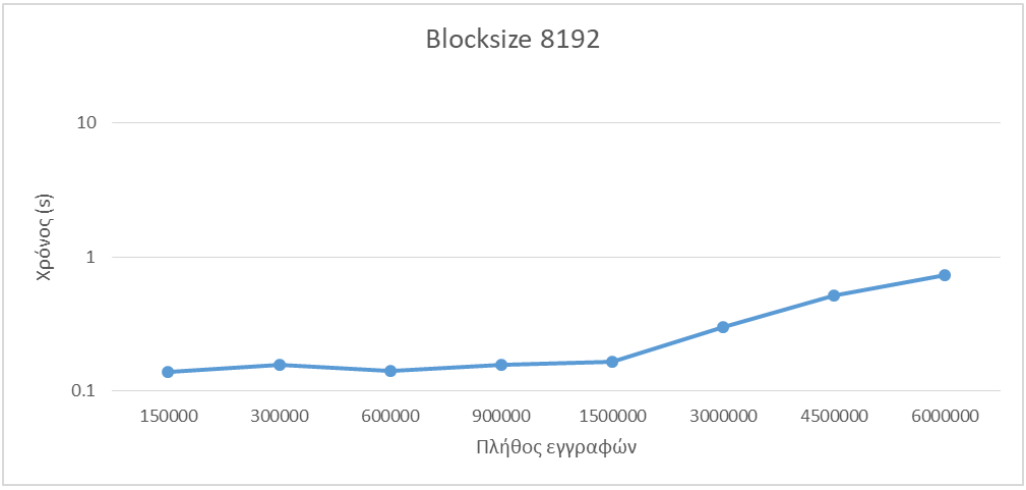
Σχήμα 5.8: Μέσος χρόνος αναζήτησης για Blocksize 1024 - Λογαριθμική κλίμακα



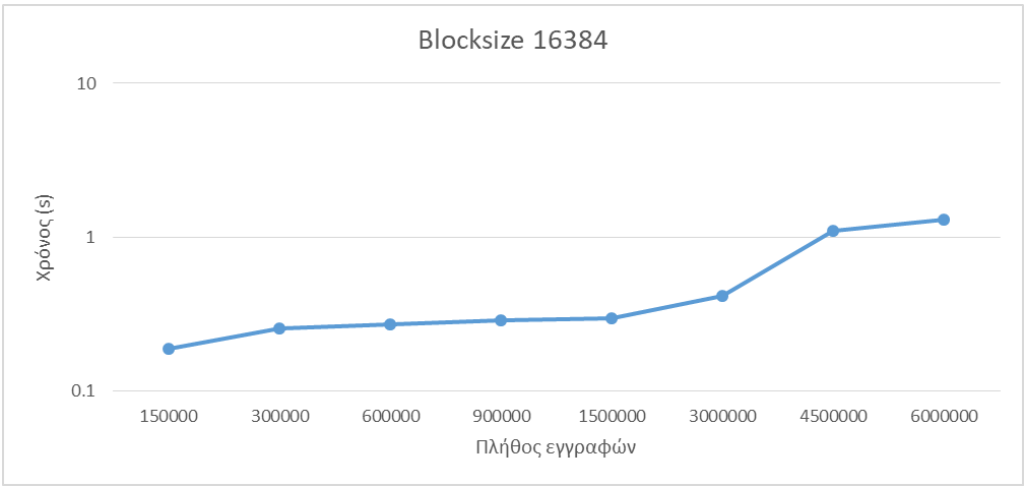
Σχήμα 5.9: Μέσος χρόνος αναζήτησης για Blocksize 2048 - Λογαριθμική κλίμακα



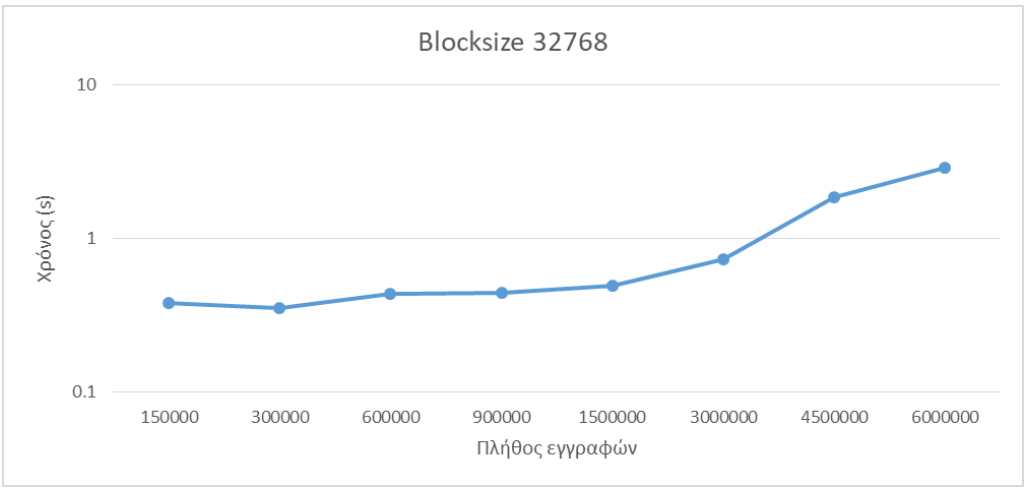
Σχήμα 5.10: Μέσος χρόνος αναζήτησης για Blocksize 4096 - Λογαριθμική κλίμακα



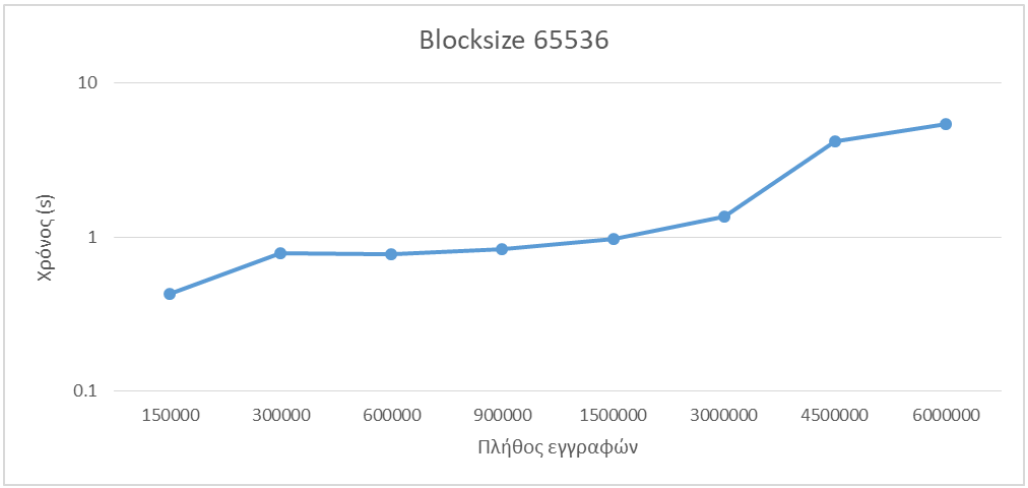
Σχήμα 5.11: Μέσος χρόνος αναζήτησης για Blocksize 8192 - Λογαριθμική κλίμακα



Σχήμα 5.12: Μέσος χρόνος αναζήτησης για Blocksize 16384 - Λογαριθμική κλίμακα



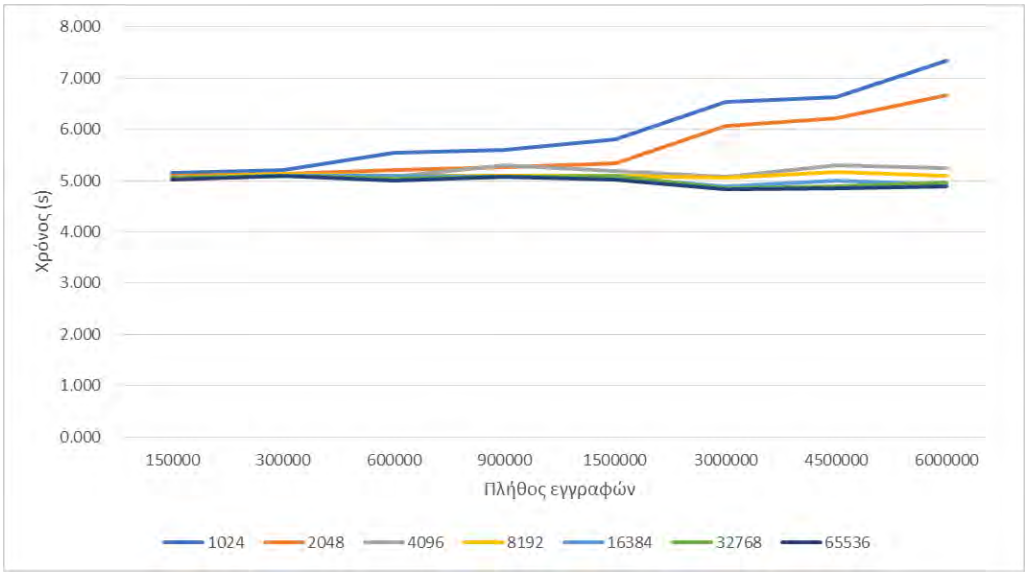
Σχήμα 5.13: Μέσος χρόνος αναζήτησης για Blocksize 32768 - Λογαριθμική κλίμακα



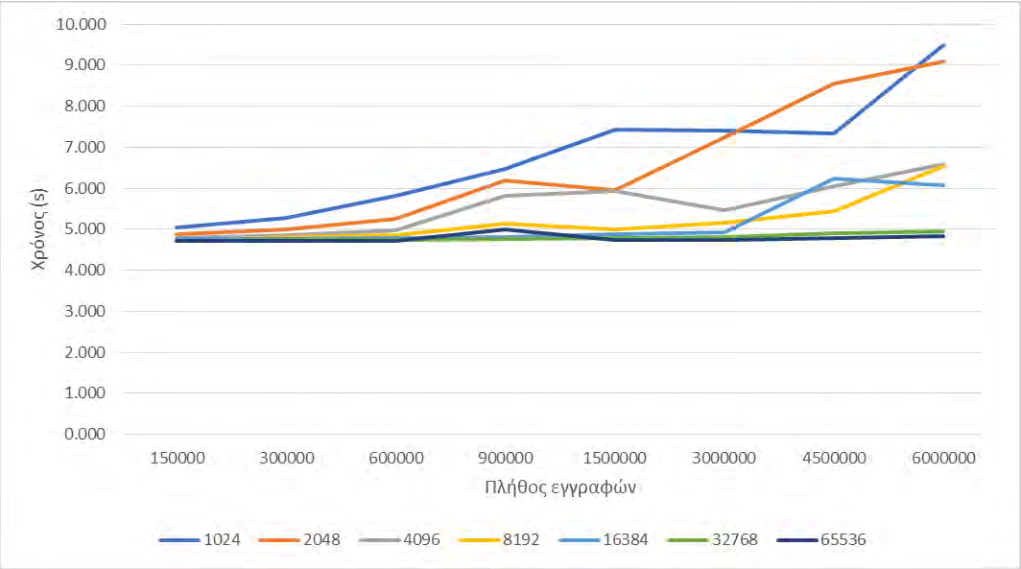
Σχήμα 5.14: Μέσος χρόνος αναζήτησης για Blocksize 65536 - Λογαριθμική κλίμακα

Blocksize / Αριθμός εγγραφών	**1024**	**2048**	**4096**	**8192**	**16384**	**32768**	**65536**
150.000	0,0683	0,0579	0,0745	0,1388	0,1873	0,3786	0,4275
300.000	0,066	0,06	0,0753	0,1558	0,2546	0,3512	0,7885
600.000	0,0691	0,0612	0,0837	0,142	0,2736	0,4362	0,7734
900.000	0,0737	0,0627	0,0929	0,1553	0,2884	0,4441	0,8359
1.500.000	0,1091	0,0627	0,1062	0,1639	0,2958	0,4927	0,9726
3.000.000	2,7855	0,1144	0,112	0,3	0,413	0,7305	1,359
4.500.000	15,122	9,3223	0,2271	0,5137	1,099	1,8475	4,219
6.000.000	29,327	25,1195	4,7692	0,735	1,3	2,8699	5,3912

Πίνακας 5.2: Συγκεντρωτικός πίνακας με τους χρόνους αναζήτησης σε second



Σχήμα 5.15: Χρόνοι δημιουργίας του πίνακα IDhbase



Σχήμα 5.16: Χρόνοι δημιουργίας του πίνακα REPhbase

Blocksize / Πλήθος εγγραφών	**1024**	**2048**	**4096**	**8192**	**16384**	**32768**	**65536**
150.000	5,150	5,092	5,079	5,052	5,044	5,055	5,026
300.000	5,213	5,128	5,085	5,134	5,088	5,092	5,091
600.000	5,552	5,201	5,079	5,087	5,095	5,031	5,008
900.000	5,606	5,256	5,305	5,095	5,076	5,069	5,081
1.500.000	5,800	5,338	5,191	5,095	5,057	5,090	5,014
3.000.000	6,540	6,069	5,077	5,061	4,896	4,843	4,823
4.500.000	6,621	6,224	5,295	5,164	5,005	4,892	4,849
6.000.000	7,347	6,667	5,236	5,097	4,931	4,964	4,885

Πίνακας 5.3: Συγκεντρωτικός πίνακας με τους χρόνους δημιουργίας του πίνακα IDhbase σε second

Blocksize / Πλήθος εγγραφών	**1024**	**2048**	**4096**	**8192**	**16384**	**32768**	**65536**
150000	5,049	4,873	4,794	4,745	4,768	4,717	4,724
300000	5,269	5,007	4,859	4,804	4,755	4,734	4,716
600000	5,813	5,248	4,971	4,864	4,778	4,736	4,717
900000	6,471	6,191	5,814	5,142	4,809	4,757	4,994
1500000	7,420	5,965	5,931	4,987	4,871	4,778	4,740
3000000	7,410	7,252	5,461	5,150	4,922	4,807	4,745
4500000	7,340	8,542	6,046	5,449	6,237	4,907	4,796
6000000	9,478	9,081	6,578	6,536	6,065	4,953	4,833

Πίνακας 5.4: Συγκεντρωτικός πίνακας με τους χρόνους δημιουργίας του πίνακα REPhbase σε second

Κεφάλαιο 6

Επίλογος

Στην παρούσα πτυχιακή εργασία παρουσιάστηκε η δημιουργία ενός αλγορίθμου αναζήτησης με παρεμβολή σε δεδομένα που είναι αποθηκευμένα σε μια NoSQL βάση δεδομένων όπως η Apache Hbase. Εκτελέστηκαν διάφορα πειράματα στα οποία μετρήθηκε ο μέσος χρόνος αναζήτης με βάση το blocksize, καθώς επίσης μετρήθηκε και ο χρόνος δημιουργίας των δύο βασικών πικάνων της δομής έτσι ώστε να πραγματοποιείται η παρεμβολή στα δεδομένα της μησχεσιακής βάσης δεδομένων. Η παραπάνω εργασία προσφέρει μια πειραματική εκτίμηση των οφελών της χρήσης αναζήτησης με παρεμβολή για την αναζήτηση σχοιχείων σε NoSQL βάσεις δεδομένων. Δεν είναι όμως κατάλληλη για ευρεία χρήση. Μελλοντικά, για να επιτευχθεί αυτό θα πρέπει να γίνει η υλοποίηση σ ένα πραγματικό Hadoop cluster (fully-distributed) όπου θα υπάρχουν δεδομένα της τάξεως των πολλών gigabytes καθώς και αντίστοιχη υπολογιστική ισχύς.

Παράρτημα Α΄

Πηγαίος κωδικας

Α΄.1 Data Generator

Για την δημιουργία τυχαίων δεδομένων χρησιμοποιήθηκε ο παρακάτω κώδικας σε C++, όπου μπορούμε να ελέχξουμε το πλήθος των εγγραφών που θέλουμε να δημιουργήσουμε και ως εκ τούτου και το μέγεθος του τελικού txt αρχείου. Στην συνέχεια, το txt αρχείο που δημιουργείται γίνεται import στην HBase όπως περιγράφηκε στο κεφάλαιο 4. Για να το τρέξουμε, πληκτρολογούμε στο terminal: g++ -std=c++11 main.cpp -o table && ./table, όπου main.cpp το όνομα του αρχείου.

Listing A.1: Δημιουργία τυχαίων δεδομένων

```
#include <iostream>
#include <fstream>

using namespace std;

 static const char alphabet[] =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
"abcdefghijklmnopqrstuvwxyz";

static const char nums[] =
"0123456789";

int stringLength = sizeof(alphabet) - 1;

char genRandom()  // Random string generator function.
{
    return alphabet[rand() % stringLength];
}

int main (int argc, char *argv[])
```

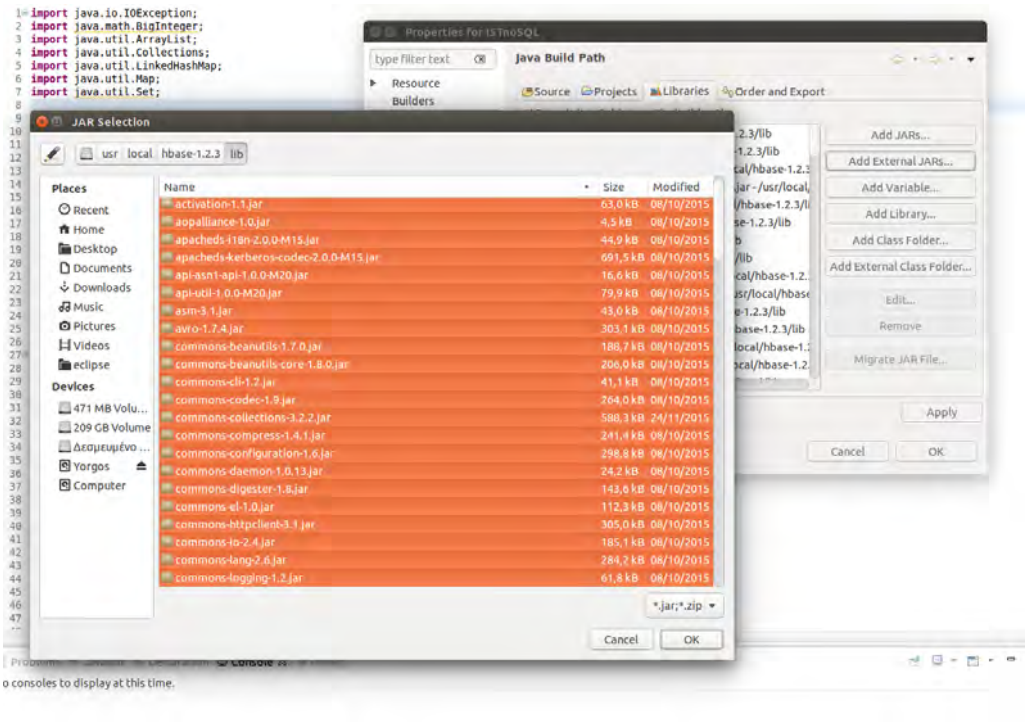
```
{
  ofstream file;
  file.open("data.txt");

  for (int i = 0; i < 1; i++){
    file << 500000000 + i+1 << "," << "George" << "," << "
       Sikotidis" << "," << "Dimitriados" << "," << "7" << ",
       " << "Volos" << "," << "Greece" << endl;;
  }
  // create 50000 records
  for (int i = 1; i < 50000; i++){
    string name = "";
    string surname = "";
    string street = "";
    string city = "";
    string country = "";
    for(int z=0; z < 10; z++){
        name += genRandom();
        surname += genRandom();
        street += genRandom();
        city += genRandom();
        country += genRandom();
    }
    file << 500000000 + i+1 << "," << name << "," << surname
       << "," << street << "," << (rand() % 50) + 1 << "," <<
        city << "," << country << endl;;
  }
  file.close();
  return 0;
}
```

Α΄.2 Interpolation Search Tree

Παρακάτω παρατίθεται ο πηγαίος κώδικας για την υλοποίηση του IST. Η ανάπτυξη του κώδικα έγινε στο ολοκληρωμένο περιβάλλον ανάπτυξης (Integrated Development Environment - IDE) Eclipse 4.5.1. Αρχικά, θα πρέπει να ξεκινήσουμε το Hadoop και την HBase πηγαίνοντας στους αντίστοιχους καταλόγους εγκατάστασης, όπως περιγράφηκε προηγουμένως στις οδηγίες εγκατάστασης, και να τρέξουμε τις εντολές start-all.sh και start-hbase.sh αντίστοιχα. Αφού ξεκινήσουμε το Eclipse και δημιουργήσουμε το project, θα πρέπει να κάνουμε import τις βιβλιοθήκες του Hadoop και της HBase έτσι ώστε να μπορέσουμε χρήση των

αντίστοιχων κλάσεων και συναρτήσεων τους. Για να γίνει αυτό, κάνουμε δεξί κλικ στην δεξιά στήλη στο όνομα του πρότζεκτ και επιλέγουμε ιδιότητες. Στην συνέχεια στο πεδίο Java Build Path επιλέγουμε αρχικά την καρτέλα Libraries και στην συνέχεια την επιλογή Add External JARs. Τέλος, βρίσκουμε την τοποθεσία στον δίσκο που έχουμε αποθηκευμένη την HBase (και το Hadoop) και στον κατάλογο libs επιλέγουμε όλα τα jar αρχεία που βρίσκονται εκεί όπως φαίνεται στο σχήμα Α΄.1.



Σχήμα Α΄.1: Προσθήκη εξωτερικών jar αρχείων

Στην αρχή του αλγορίθμου φορτώνεται ο πίνακας των δεδομένων έτσι ώστε να αρχικοποιηθούν οι παράμετροι που χρησιμοποιούνται στην συνέχεια για την κατασκευή των πινάκων για την αναζήτηση με παρεμβολή. Έπειτα, κατασκευάζονται οι δύο πίνακες IDhbase και REPhbase και τέλος γίνεται η διαδικασία της αναζήτησης σύμφωνα με όσα περιγράφηκαν στο κεφάλαιο 4.

Listing A.2: Πηγαίος κώδικας του IST

```
1 import java.io.IOException;
2 import java.math.BigInteger;
3 import java.util.ArrayList;
4 import java.util.Collections;
5 import java.util.LinkedHashMap;
6 import java.util.Map;
7 import java.util.Set;
8 import org.apache.hadoop.conf.Configuration;
9 import org.apache.hadoop.hbase.HBaseConfiguration;
```

```
10 import org.apache.hadoop.hbase.HColumnDescriptor;
11 import org.apache.hadoop.hbase.HTableDescriptor;
12 import org.apache.hadoop.hbase.KeyValue;
13 import org.apache.hadoop.hbase.client.Get;
14 import org.apache.hadoop.hbase.client.HBaseAdmin;
15 import org.apache.hadoop.hbase.client.HTable;
16 import org.apache.hadoop.hbase.client.Put;
17 import org.apache.hadoop.hbase.client.Result;
18 import org.apache.hadoop.hbase.client.ResultScanner;
19 import org.apache.hadoop.hbase.client.Scan;
20 import org.apache.hadoop.hbase.filter.Filter;
21 import org.apache.hadoop.hbase.filter.PageFilter;
22 import org.apache.hadoop.hbase.util.Bytes;
23
24 public class isthbase {
25
26   public static void main(String[] args) throws IOException {
27
28     // ****** Read HBase table data
29
30     // Declare the table from which the data will be loaded,
          here named 'data10'
31     HTable table = new HTable(HBaseConfiguration.create(), "
          data10");
32     Scan scan = new Scan();
33
34     scan.addFamily(Bytes.toBytes("name"));
35     scan.addFamily(Bytes.toBytes("surname"));
36     scan.addFamily(Bytes.toBytes("street"));
37     scan.addFamily(Bytes.toBytes("streetNumber"));
38     scan.addFamily(Bytes.toBytes("city"));
39     scan.addFamily(Bytes.toBytes("country"));
40
41     // Fisrt step is to read the data and make the
          appropriate computations on ram
42
43     // Declaration of a Map for the key-value information
44     Map<String, String> keyValues = new LinkedHashMap<String,
           String>();
45     // Declaration of an ArrayList for the IDs-keys
46     ArrayList<String> keys = new ArrayList<String>();
```

```
47      // Declaration of an ArrayList for the range search
48      ArrayList<Integer> keysForRangeSearch = new ArrayList<
           Integer>();
49      //initialize first ID
50      int firstElement = 500000001;
51      keysForRangeSearch.add(firstElement);
52
53      // Create a scanner for the table
54      ResultScanner scanner = table.getScanner(scan);
55      // Add the IDs
56      for (Result r : scanner) {
57        keys.add(new String(r.getRow()));
58      }
59
60      double numberOfStoredElements = 0;
61      // Count the number of the stored elements
62      for (int i = 0; i < keys.size(); i++) {
63        numberOfStoredElements += 1;
64        keysForRangeSearch.add(firstElement + i + 1);
65      }
66
67      // Call for a shuffle function for the range search later
68      Collections.shuffle(keysForRangeSearch);
69
70      // ****** Read data to ram - Prerequisites
71
72      // Declare the blocksize
73      double blockSize = 1024;
74      // Compute the number of the buckets
75      double numberOfBuckets = Math.ceil(numberOfStoredElements
            / blockSize);
76      // Declare the e parameter
77      double e = 0.1;
78      // Declare and compute the I(n), which equals to the
           total number of elements of ID array
79      double yiota;
80      yiota = Math.floor(
81          numberOfBuckets / Math.pow(Math.log(Math.log(
               numberOfBuckets) / Math.log(2)) / Math.log(2), 1 +
                e));
82      // Print the bumber of the buckets
```

```
83      System.out.println("Number of buckets R(n): " +
            numberOfBuckets);
84      // Declare an array for the stored elements
85      int[] V = new int[(int) numberOfStoredElements];
86      // Declare the REP table
87      int[] REP = new int[(int) numberOfBuckets];
88      // Declare the ID table
89      int[] ID = new int[(int) yiota];
90      // Put values to V[] array
91      for (int i = 1; i <= numberOfStoredElements; i++) {
92        V[i - 1] = i;
93      }
94
95      // The REP array has the minimum value of each bucket of
            the V array
96      for (int j = 0; j < numberOfBuckets; j++) {
97        REP[j] = Integer.parseInt(keys.get(j * (int) blockSize)
              );
98      }
99      // Compute the lower value
100     int lowerValue = REP[0];
101     // Print the lower value, mainly for debugging
102     System.out.println("lowerValue " + lowerValue);
103     // Compute the upper value
104     int upperValue = REP[((int) numberOfBuckets - 1)];
105     // Print the upper value, mainly for debugging
106     System.out.println("upperValue " + upperValue);
107
108     // Compute the values of the ID table
109     for (int k = 0; k < (int) yiota; k++) {
110       for (int z = 0; z < numberOfBuckets; z++) {
111         if (((lowerValue + k * (upperValue - lowerValue) /
                yiota) > REP[z])
112             && ((lowerValue + k * (upperValue - lowerValue) /
                    yiota) <= REP[z + 1])) {
113           ID[k] = z;
114           break;
115         }
116       }
117     }
118
```

```
119     // ****** Build the tree - ID and REP tables for HBase
120
121     // Create IDhbase table
122
123     HBaseConfiguration hconfig = new HBaseConfiguration(new
           Configuration());
124     HBaseAdmin hbase_admin = new HBaseAdmin(hconfig);
125     // Run disable and delete only if there is already a
           REPhbase table, otherwise commented
126     hbase_admin.disableTable("IDhbase");
127     hbase_admin.deleteTable("IDhbase");
128
129     HBaseConfiguration hc = new HBaseConfiguration(new
           Configuration());
130     HTableDescriptor ht = new HTableDescriptor("IDhbase");
131     // Create column id for the interpolation
132     ht.addFamily(new HColumnDescriptor("id"));
133     System.out.println("connecting");
134     HBaseAdmin hba = new HBaseAdmin(hc);
135
136     // Create the table
137     System.out.println("Creating IDhbase table...");
138     hba.createTable(ht);
139     org.apache.hadoop.conf.Configuration config =
           HBaseConfiguration.create();
140     HTable table2 = new HTable(config, "IDhbase");
141
142     // Put the values inside the IDhbase table
143     for (int i = 0; i < yiota; i++) {
144       String iStr = String.valueOf(i);
145       String idStr = String.valueOf(ID[i]);
146       // Transform from string to bytes and put the value
147       Put p = new Put(Bytes.toBytes(iStr));
148       p.add(Bytes.toBytes("id"), Bytes.toBytes("id"), Bytes.
             toBytes(idStr));
149       table2.put(p);
150     }
151     System.out.println("IDhbase table created");
152
153     // Create REPhbase table
154
```

```
155     // Run disable and delete only if there is already a
           REPhbase table, otherwise commented
156     hbase_admin.disableTable("REPhbase");
157     hbase_admin.deleteTable("REPhbase");
158
159     HBaseConfiguration hcRep = new HBaseConfiguration(new
           Configuration());
160     HTableDescriptor htRep = new HTableDescriptor("REPhbase")
           ;
161     // create column REPid for the interpolation
162     htRep.addFamily(new HColumnDescriptor("REPid"));
163     System.out.println("connecting");
164     HBaseAdmin hbaRep = new HBaseAdmin(hcRep);
165
166     // Create the table
167     System.out.println("Creating REP Table...");
168     hbaRep.createTable(htRep);
169     org.apache.hadoop.conf.Configuration configRep =
           HBaseConfiguration.create();
170     HTable tableRep = new HTable(configRep, "REPhbase");
171
172     for (int i = 0; i < numberOfBuckets; i++) {
173       String iStr2 = String.valueOf(i);
174       String repStr = String.valueOf(REP[i]);
175       // Transform from string to bytes and put the value
176       Put p = new Put(Bytes.toBytes(iStr2));
177       p.add(Bytes.toBytes("REPid"), Bytes.toBytes("REPid"),
             Bytes.toBytes(repStr));
178       tableRep.put(p);
179     }
180     System.out.println("REPhbase table created");
181
182
183     // ****** Interpolation Search
184
185     ArrayList<Double> times = new ArrayList<Double>();
186     // iterate the process for the proper mean search time,
           here 500 times
187     int totalIterations = 500;
188     for (int i = 0; i < totalIterations; i++) {
189       System.out.println("completed: " + i*100/
```

```
            totalIterations + "%");
190         // Start count the time
191         double start = System.nanoTime();
192         // Initialize a random element for searching from the
               keysForRangeSearch array
193         double element = keysForRangeSearch.get(i); // 50000000
               1;
194         int idIndex = 0;
195         // find the idIndex
196         idIndex = (int) Math.floor((yiota * (element -
               lowerValue) / (upperValue - lowerValue)));
197         double repIndex;
198         String idIndexIST;
199         // find the repIndex
200         if (idIndex == yiota) {
201           // repIndex = ID[(int)idIndex-1];
202           idIndexIST = String.valueOf(idIndex - 1);
203           repIndex = ID[(int) idIndex - 1];
204         } else {
205           idIndexIST = String.valueOf(idIndex);
206           repIndex = ID[(int) idIndex];
207           // repIndex = ID[(int)idIndex];
208         }
209         // Instantiating Get class
210         Get g = new Get(Bytes.toBytes(idIndexIST));
211         // Reading the data
212         Result result = table2.get(g);
213         // Reading values from Result class object
214         byte[] valuexx = result.getValue(Bytes.toBytes("id"),
               Bytes.toBytes("id"));
215
216         String name = Bytes.toString(valuexx);
217         String repIndex2 = name;
218
219         // Instantiating Get class
220         Get g2 = new Get(Bytes.toBytes(repIndex2));
221         // Reading the data
222         Result result2 = tableRep.get(g2);
223         // Reading values from Result class object
224         byte[] valuexx2 = result2.getValue(Bytes.toBytes("REPid
               "), Bytes.toBytes("REPid"));
```

```
225       String name2 = Bytes.toString(valuexx2);
226       Scan scan2 = new Scan(Bytes.toBytes(name2), Bytes.
             toBytes(keys.get(keys.size() - 1) + 1));
227       // Start the search procedure from the found id,
             interpolation done
228       ResultScanner scanner2 = table.getScanner(scan2);
229       for (Result r : scanner2) {
230         String fo = new String(r.getRow());
231         int found = Integer.parseInt(fo);
232         if (found == element) {
233           double elapsedTime = (System.nanoTime() - start) /
                 Math.pow(10, 9);
234           times.add(elapsedTime);
235           break;
236         }
237       }
238     }
239     System.out.println("completed: 100%");
240     double totalTime = 0;
241     for (int i = 0; i < times.size(); i++) {
242       totalTime += times.get(i);
243     }
244     totalTime /= times.size();
245     System.out.println("Total search time: " + totalTime + "
          seconds");
246   }
247 }
```

Βιβλιογραφία

[1] Apache hadoop. `http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html`, Accessed: June-2018.

[2] Arron fu, cto uniprint.net, 7 different types of cloud computing structures. `https://www.uniprint.net/en/7-types-cloud-computing-structures`, Accessed: June-2018.

[3] Jeffrey Dean και Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[4] Lars George. *HBase: The Definitive Guide*. O'Reilly Media, 1η έκδοση, 2011.

[5] Sanjay Ghemawat, Howard Gobioff και Shun Tak Leung. The google file system. Στο *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, σελίδες 20–43, Bolton Landing, NY, 2003.

[6] Alexis Kaporis, Christos Makris, George Mavritsakis, Spyros Sioutas, Athanasios Tsakalidis, Kostas Tsichlas και Christos Zaroliagis. Isb-tree: A new indexing scheme with efficient expected behaviour. *Journal of Discrete Algorithms*, 8(4):373 – 387, 2010.

[7] Alexis Kaporis, Christos Makris, Spyros Sioutas, Athanasios Tsakalidis, Kostas Tsichlas και Christos Zaroliagis. Dynamic interpolation search revisited, 2006.

[8] Amandeep Khurana. Introduction to hbase schema design. `https://www.usenix.org/publications/login/october-2012-volume-37-number-5/introduction-hbase-schema-design`, 2012.

[9] Richard McCreadie, Craig Macdonald και Iadh Ounis. Mapreduce indexing strategies: Studying scalability and efficiency. *Information Processing & Management*, 48(5):873 – 888, 2012.

[10] Spyros Sioutas, Peter Triantafillou, George Papaloukopoulos, Evangelos Sakkopoulos, Kostas Tsichlas και Yannis Manolopoulos. ART : Sub-logarithmic decentralized range query processing with probabilistic guarantees. *CoRR*, αβς/1201.2766, 2012.

[11] Gerth Stølting Brodal, Alexis Kaporis, Apostolos Papadopoulos, Spyros Sioutas, Konstantinos Tsakalidis και Kostas Tsichlas. Dynamic 3-sided planar range queries with expected doubly logarithmic time, 2012.

[12] Γιάννης Κοκοτίνης. Μελέτη και Υλοποίηση Αποδοτικών Ευρετηρίων (Indexing Methods) σε NoSQL Database Systems. Πάτρα, 2015.

[13] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 4τηη έκδοση, 2015.

[14] Wikipedia. Big data. `https://en.wikipedia.org/wiki/Big_data`, [Accessed: May-2018].

[15] Wikipedia. Cloud computing. `https://en.wikipedia.org/wiki/Cloud_computing`, [Accessed: May-2018].