Πανεπιστήμιο Πατρών
Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Διπλωματική Εργασία

Ανάκτηση Πληροφορίας σε Νέφη Υπολογιστών

Ζώης Βασίλειος
4183

Επιβλέπων: Γαροφαλάκης Ιωάννης
Εξεταστές: Γαροφολάκης Ιωάννης, Χρήστος Μακρής

Πάτρα, Σεπτέμβριος 2012

Περιεχόμενα

Σχήματα

Εισαγωγή

Η ραγδαία αύξηση της πληροφορίας σε συνδυασμό με τις συνεχόμενες απαιτήσεις των χρηστών, καθώς και των εταιριών για άμεσες λύσεις σε πληροφοριακά ζητήματα οδήγησε στην ανάπτυξη εξελιγμένων τεχνικών, για την ανάκτηση και την οργάνωση των δεδομένων. Η επικράτηση του διαδικτύου συνέβαλε στην διόγκωση της πληροφορίας που καλούμαστε να επεξεργαστούμε. Έτσι γεννήθηκε η ανάγκη για παροχή υπηρεσιών, που θα επιλύουν το πρόβλημα της επεξεργασίας και της διαθεσιμότητας των δεδομένων για την εξυπηρέτηση των αναγκών των χρηστών. Οι υπηρεσίες αυτές εμφανίστηκαν με την δημιουργία των κατανεμημένων συστημάτων, τα οποία έδωσαν την ώθηση για την κατασκευή και την λειτουργία των νεφών υπολογιστών. Τα νέφη υπολογιστών εξασφάλισαν την κάλυψη των παραπάνω αναγκών και παρείχαν λύσεις για την κατασκευή επεκτάσιμων και αξιόπιστων συστημάτων.

Στην παρούσα διπλωματική εργασία εξετάζουμε ενδελεχώς την λειτουργία και τις υπηρεσίες που παρέχουν τα κατανεμημένα συστήματα. Επικεντρωνόμαστε στα κατανεμημένα συστήματα αρχείων τα οποία είναι και η βάση των νεφών υπολογιστών, καθώς και στο πλέον διαδεδομένο μοντέλο προγραμματισμού mapreduce. Επίσης θα ασχοληθούμε και με τις κατανεμημένες βάσεις δεδομένων και πιο συγκεκριμένα με τα ευρέως διαδεδομένα key-value stores. Στην συνέχεια εστιάζουμε στην κατασκευή, διατήρηση και διαχείριση B+, B-Trees πάνω από τα συστήματα αυτά.

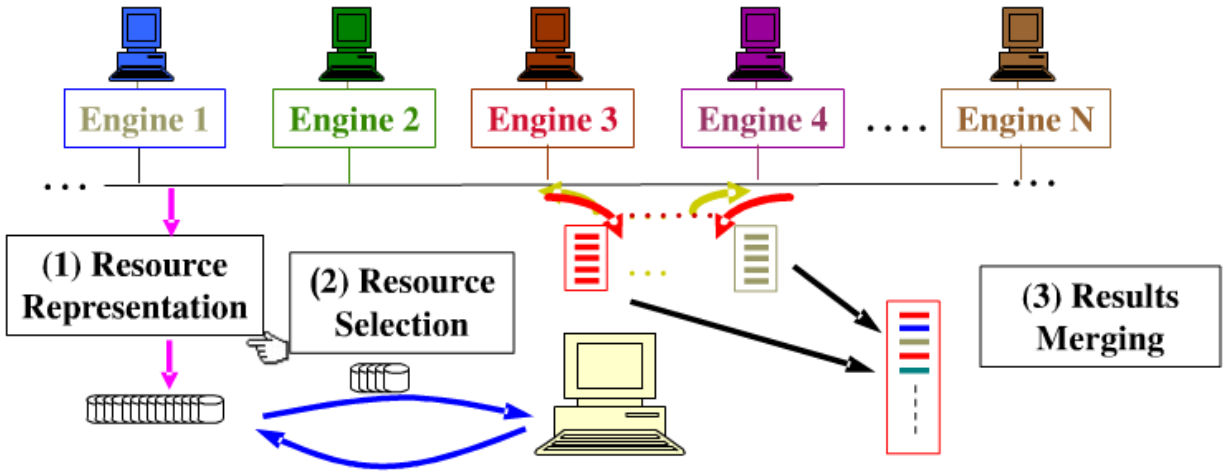
Τέλος εκτελώντας πειράματα που αντικατοπτρίζουν τις πραγματικές απαιτήσεις ενός συστήματος, εξάγουμε συγκεκριμένα συμπεράσματα για την υλοποίηση και προτείνουμε συγκεκριμένες λύσεις, με σκοπό την βελτίωση της απόδοσης του συστήματος.

1. Ανάκτηση Πληροφορίας

Η επιστήμη της ανάκτησης πληροφορίας ασχολείται με την αναπαράσταση, αποθήκευση, οργάνωση και διανομή της πληροφορίας. Η διαδικασία ανάκτησης πληροφορίας χωρίζεται σε δύο τομείς, στην αποσαφήνιση της πληροφοριακής ανάγκης και στην ανάκτηση των σχετικών, στην συγκεκριμένη πληροφοριακή, ανάγκη αντικειμένων.

Η διαδικασία της εξαγωγής της πληροφορίας ξεκινάει όταν ο χρήστης ορίσει την πληροφοριακή του ανάγκη, η οποία συνήθως έχει μορφή ερωτήματος. Το ερώτημα αυτό μπορεί να αντιστοιχίζεται σε περισσότερα από ένα αντικείμενα. Στο σημείο αυτό το σύστημα θα πρέπει να αντιστοιχίσει το ερώτημα του χρήστη σε μία συλλογή από σχετικά αντικείμενα. Το κομμάτι αυτό της επεξεργασίας σχετίζεται με την αποσαφήνιση της πληροφοριακής ανάγκης του χρήστη. Ο τομέας της πληροφορικής που ασχολείται με την διαδικασία αυτή ονομάζεται τομέας επεξεργασία φυσικής γλώσσας.

Αφού οριστεί η πληροφοριακή ανάγκη, το επόμενο βήμα είναι να ανακτηθούν επιτυχώς και πραγματικό χρόνο τα αντίστοιχα δεδομένα που ικανοποιούν την ανάγκη αυτή. Με την επικράτηση του διαδικτύου, ο τρόπος επεξεργασίας των δεδομένων για την παροχή υπηρεσιών άλλαξε δραματικά. Ο αυξημένος όγκος των δεδομένων συνέβαλλε στην αλλαγή αυτή και άνοιξε το δρόμο για την επικράτηση κατανεμημένων λύσεων. Το κλασσικό μοντέλο της ανάκτησης πληροφορίας εγκαταλείφθηκε και εμφανίστηκαν εναλλακτικές επιλογές, με την επικράτηση των πρώτων κατανεμημένων συστημάτων, τα οποία χρησιμοποιήθηκαν για την παροχή υπηρεσιών αναζήτησης, κοινωνικής δικτύωσης και αγοροπωλησιών.



ΣΧΗΜΑ 1.1 – ΚΑΤΑΝΕΜΗΜΕΝΗ ΑΝΑΚΤΗΣΗ ΠΛΗΡΟΦΟΡΙΑΣ

1.1 Ανάκτηση Πληροφορίας σε Κατανεμημένα Συστήματα

Η επίλυση του προβλήματος για την αποδοτική επεξεργασία και διανομή της πληροφορίας, επιτυγχάνεται με την χρήση κατανεμημένων συστημάτων. Ένα κατανεμημένο σύστημα προσφέρει υπηρεσίες διασύνδεσης και έλεγχου αυτόνομων υπολογιστικών πόρων, οι οποίες στοχεύουν στην υποστήριξη αξιόπιστων και επεκτάσιμων εφαρμογών κατανεμημένου υπολογισμού. Το σύστημα επίσης θα πρέπει να προσφέρει έναν μοντέλο προγραμματισμού για την διευκόλυνση του χρήστη, κατά την διαδικασία συγγραφής εφαρμογών. Ταυτόχρονα θα πρέπει να παρέχει συγκεκριμένα εργαλεία διαχείρισης και οργάνωσης δεδομένων, όπως βάσεις δεδομένων ειδικά προσαρμοσμένα στις δυνατότητες, αλλά και στους περιορισμούς ενός κατανεμημένου περιβάλλοντος. Η υλοποίηση τέτοιων συστημάτων αποτελεί ιδιαίτερη πρόκληση, με τους στόχους που πρέπει να επιτύχουμε να είναι καθορισμένοι αλλά όχι δεδομένοι. Στην πληροφορική και συγκεκριμένα στον τομέα των κατανεμημένων συστημάτων το γνωστό θεώρημα του Brewer(CAP Theorem) δηλώνει ότι είναι αδύνατο για ένα κατανεμημένο σύστημα να παρέχει ταυτόχρονα και τις τρείς παρακάτω εγγυήσεις :

- ***Συνοχή***

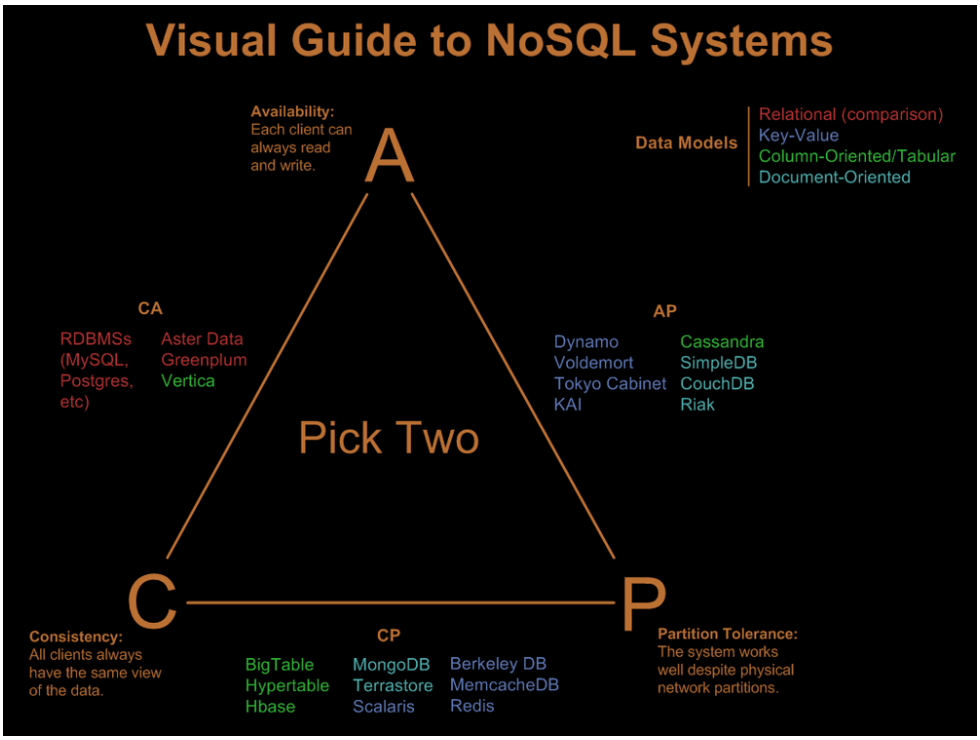
  Όλοι οι πελάτες του συστήματος, ανεξαρτήτως από τις ενημερώσεις που έχουν γίνει, θα πρέπει να βλέπουν τα ίδια δεδομένα.

- ***Διαθεσιμότητα***

  Ο πελάτης μπορεί να γράφει και να διαβάζει δεδομένα στο σύστημα οποιαδήποτε στιγμή θελήσει.

- ***Ανοχή σε Σφάλματα***

  Ανεξάρτητα από τα σφάλματα που παρουσιάζονται στο δίκτυο ή στα μηχανήματα που αποτελούν το συνολικό σύστημα, ο κάθε πελάτης θα μπορεί να έχει πρόσβαση στις υπηρεσίες που του παρέχονται.



ΣΧΗΜΑ 1.2 - CAP THEOREM

Έτσι ακολουθήθηκαν διαφορετικές προσεγγίσεις για την υλοποίηση κατανεμημένων συστημάτων, τα οποία ικανοποιούν δύο από τις τρείς εγγυήσεις που περιγράψαμε παραπάνω. Στο σχήμα 1.2 βλέπουμε μια γραφική αναπαράσταση του θεωρήματος CAP και τις διαφορετικές υλοποιήσεις κατανεμημένων συστημάτων, οι οποίες κατηγοριοποιούνται ανάλογα με τις

εγγυήσεις που προσφέρουν στον χρήστη. Η δημιουργία ενός συστήματος που να ικανοποιεί και τις τρεις εγγυήσεις παραμένει ανοιχτό πρόβλημα.

2.Νέφη Υπολογιστών

Το νέφος υπολογιστών περιγράφεται ως ένα σύνολο υπολογιστικών συστημάτων, τα οποία είναι συνδεδεμένα μεταξύ τους σε ένα ιδιωτικό ή δημόσιο δίκτυο, με τέτοιο τρόπο ώστε να προσφέρουν δυναμικές και επεκτάσιμες υποδομές, για την εκτέλεση απαιτητικών εφαρμογών και την αποθήκευση μεγάλου όγκου δεδομένων. Με την ανάπτυξη της τεχνολογίας, το κόστος των υπολογισμών, της αποθήκευσης και την διανομής δεδομένων έχει μειωθεί δραματικά. Τα νέφη υπολογιστών είναι μία πρακτική προσέγγιση με την οποία έχουμε άμεσα πλεονεκτήματα από άποψη κόστους, μετατρέποντας ένα απλό data center υψηλού κόστους σε ένα σύστημα που εκτελεί απαιτητικές εργασίες, με μεταβλητό κόστος το οποίο είναι εύκολα επεκτάσιμο(εύκολη προσθήκη νέων υπολογιστικών πόρων χωρίς απώλειες τις απόδοσης).

Η ιδέα πίσω από την υλοποίηση ενός νέφους υπολογιστών βασίζεται στην θεμελιώδη αρχή της επαναχρησιμοποίησης υπολογιστικών πόρων. Η διαφορά του cloud computing σε σχέση με τις κλασσικές αρχές του grid και distributed computing είναι, η προσπάθεια της διεύρυνσης της λειτουργίας των αντίστοιχων συστημάτων, προσφέροντας μια γενική λύση στα προβλήματα διαχείρισης υπολογιστικών πόρων, εύκολου προγραμματισμού εφαρμογών και διάθεσης αξιόπιστων υπηρεσιών σε μεγάλο αριθμό χρηστών.

2.1 Κατηγορίες

Υπάρχουν διαφορετικές υπηρεσίες που παρέχονται, όπως αναφέρθηκε και παραπάνω, από τα νέφη υπολογιστών(Cloud). Οι παροχείς υπηρεσιών Cloud προσφέρουν υπηρεσίες που μπορούν να χωριστούν στις παρακάτω κατηγορίες:

1. ***Software as a Service***

   Στην κατηγορία αυτή προσφέρεται στο χρήστη μία εφαρμογή ανάλογα με την ζήτηση που υπάρχει. Η εφαρμογή αυτή εξυπηρετεί πολλαπλές αιτήσεις από διαφορετικούς χρηστές. Παραδείγματα τέτοιων υπηρεσιών είναι το Google Docs, acrobat.com κ.τ.λ.

2. ***Platform as a Service***

Στο μοντέλο αυτό ο χρήστης χτίζει τις δικές του εφαρμογές πάνω από μία πλατφόρμα κατασκευασμένη από τον πάροχο της υπηρεσίας cloud. Συνήθως προσφέρεται ένας προκαθορισμένος συνδυασμός κάποιου λειτουργικού συστήματος με ένα προγραμματιστικό περιβάλλον, ώστε να γίνεται πιο εύκολη η διαχείριση και η ανάπτυξη των εφαρμογών του χρήστη. Παραδείγματα τέτοιων υπηρεσιών είναι τα Google App Engine , Azure Service Platform κ.τ.λ.

3. ***Infrastructure as a Service***

Στο μοντέλο αυτό παρέχονται στο χρήστη οι βασικές υποδομές αποθήκευσης και εκτέλεσης υπολογισμών πάνω από το διαδίκτυο. Ο πελάτης κατασκευάζει το δικό του λογισμικό εκμεταλλευόμενος τις υποδομές που του διατίθενται από τον αντίστοιχο πάροχο. Τέτοιες υπηρεσίες παρέχονται από το Amazon, Go Grid κ.τ.λ.

2.2 Πλεονεκτήματα

Η εκμετάλλευση του αρχιτεκτονικού μοντέλου ενός νέφους υπολογιστών προϋποθέτει την τροποποίηση των σημερινών εφαρμογών, ώστε να «ευθυγραμμίζονται» με την λειτουργίες που παρέχει το νέφος. Μερικά από τα πλεονεκτήματα που παρέχει το νέφος υπολογιστών περιγράφονται παρακάτω:

1. ***Μειωμένο Κόστος Λειτουργίας***

Οι υποδομές που χρησιμοποιούνται για την κατασκευή και την λειτουργία ενός νέφους υπολογιστών, βασίζονται στην χρήση πολλαπλών υπολογιστικών συστημάτων με φτηνό και χαμηλής κατανάλωσης υλικό.

2. ***Αυξημένος Αποθηκευτικός χώρος***

Ο μεγάλος όγκος υποδομών που παρέχονται από ένα νέφος υπολογιστών κάνει πραγματικότητα την αποθήκευση και διατήρηση μεγάλου όγκου δεδομένων. Επίσης

εξαιτίας τις επεκτασιμότητας, η οποία αποτελεί κύριο χαρακτηριστικό του νέφους υπολογιστών, έχουμε δυναμική εξισορρόπηση φορτίου στους υπολογιστικούς πόρους του συστήματος, γεγονός που το καθιστά ευέλικτο στην καθημερινή αύξηση του αριθμού και των απαιτήσεων των εκάστοτε χρηστών.

3. ***Ευελιξία***

Οι σύγχρονες απαιτήσεις για γρήγορη προσαρμογή των εταιριών σε νέες καταστάσεις καθιστά κρίσιμη την δυνατότητα άμεσης παραγωγής αποτελεσμάτων. Τα νέφη υπολογιστών έχουν την απαραίτητη υπολογιστική ισχύ ώστε να παρέχουν αποτελέσματα σε πραγματικό χρόνο.

4. ***Ανοιχτό Λογισμικό για Παροχή Υπηρεσιών***

Υπάρχουν υλοποιήσεις συστημάτων που προσφέρουν υπηρεσίες cloud οι οποίες είναι ανοιχτού λογισμικού όπως το Hadoop(Hadoop Framework).

5. ***Αξιοπιστία & Εύκολη Διαχείριση***

Εξαιτίας του πλεονασμού, δεν υπάρχουν μοναδικά σημεία αποτυχίας στο σύστημα έτσι οι υπηρεσίες που προσφέρονται είναι αξιόπιστες. Επίσης με την επαναχρησιμοποίηση των πόρων το σύστημα διαχειρίζεται εύκολα τους πόρους που έχει στην κατοχή του.

6. ***Επεκτασιμότητα***

Αποτελεί ένα από τα σημαντικότερα χαρακτηριστικά των κατανεμημένων συστημάτων, το οποίο είναι αναγκαίο για την εξυπηρέτηση των αυξανόμενων αναγκών. Η ανάγκη για την αύξηση της υπολογιστικής ισχύς οδηγεί στην εμφάνιση του προβλήματος της συμβατότητας. Εξαιτίας της ραγδαίας ανάπτυξης στην τεχνολογία των υπολογιστών, είναι αναγκαίος ο συνδυασμός διαφορετικών τεχνολογιών(διαφορετικό υλικό). Τα συστήματα που υλοποιούν τα Νέφη Υπολογιστών, επιλύουν αυτό το πρόβλημα, παρέχοντας λογισμικό το οποίο εφαρμόζεται σε ένα μεγάλο εύρος τεχνολογιών. Ταυτόχρονα παρέχουν εγγυήσεις, για την σωστή, και αποδοτική λειτουργία των συστημάτων και των υπηρεσιών τους.

2.3 Προκλήσεις

Παρόλη την αυξανόμενη επιρροή που έχουν τα νέφη υπολογιστών υπάρχουν ακόμη ζητήματα που απαιτούν επίλυση, ώστε το σύστημα που θα προκύψει να διατηρεί τα λειτουργικά του χαρακτηριστικά του. Ορισμένα από τα ζητήματα που απαιτούν επίλυση για την εξυπηρέτηση των διαφορετικών αναγκών είναι τα ακόλουθα.

1. ***Προστασία Δεδομένων***

   Η ασφαλής προσπέλαση των δεδομένων αποτελεί σημαντικό παράγοντα που πρέπει να εγγυηθεί ο πάροχος των υπηρεσιών Νέφους Υπολογιστών. Οι εταιρίες είναι απρόθυμες να χρησιμοποιήσουν τις υπηρεσίες που τους παρέχονται, χωρίς να έχουν τις απαραίτητες εγγυήσεις για την ασφάλεια των δεδομένων τους. Το ίδιο ισχύει και για τους απλούς χρήστες οι οποίοι απαιτούν να υπάρχει διαφάνεια, ώστε η πρόσβαση στα δεδομένα να είναι εύκολη αλλά και ταυτόχρονα ασφαλής .

2. ***Ανάκτηση & Διαθεσιμότητα Δεδομένων***

   Η διαθεσιμότητα καθώς και η συνοχή των δεδομένων προς ανάκτηση, αποτελούν τις βασικότερες εγγυήσεις που πρέπει να παρέχει το σύστημα στους χρήστες του. Δεν είναι πάντα δυνατόν να επιτυγχάνεται ισχυρή συνεκτικότητα και διαθεσιμότητα των δεδομένων. Συμβιβαζόμαστε πολλές φορές στην «ασθενή» υλοποίηση αυτών των σχέσεων.

3. ***Διαχείριση Συστήματος***

   Παρόλο που υπάρχουν αρκετοί μέχρι σήμερα προμηθευτές υπηρεσιών cloud, η διαχείριση των συστημάτων δεν έχει πετύχει σημαντικά αποτελέσματα και δεν μας έχει δώσει ολοκληρωμένες λύσεις.

3.Κατανεμημένα Συστήματα Αρχείων

Το σημαντικότερο κομμάτι των κατανεμημένων συστημάτων, από το οποίο απορρέει το μεγαλύτερο μέρος της λειτουργικότητας και της αποδοτικότητας τους, είναι τα κατανεμημένα συστήματα αρχείων. Ένα κατανεμημένο σύστημα αρχείων έχει την μορφή ενός κλασσικού συστήματος αρχείων, το οποίο είναι κατανεμημένο σε πολλαπλούς υπολογιστές πάνω από ένα ιδιωτικό ή δημόσιο δίκτυο. Στην ενότητα αυτή περιγράφονται οι υπηρεσίες που παρέχουν τα κατανεμημένα συστήματα αρχείων. Επιπλέον αναφέρονται μερικά από τα πιο ευρέως χρησιμοποιούμενα. Τέλος παρατίθεται αναλυτική περιγραφή του Hadoop Distributed File System (HDFS), το οποίο και θα χρησιμοποιήσουμε στην παρούσα διπλωματική εργασία.

3.1 Υπηρεσίες

Το κατανεμημένο σύστημα αρχείων πρέπει να προσφέρει ένα βαθμό διαφάνειας στις λειτουργίες που παρέχονται στον χρήστη. Μερικές από τις υπηρεσίες που προσφέρονται περιγράφονται παρακάτω:

- ***Διαφανής Πρόσβαση στα Αρχεία***

  Οι χρήστες δεν πρέπει να γνωρίζουν ότι τα αρχεία είναι κατανεμημένα και πρέπει να μπορούν να έχουν πρόσβαση σε αυτά, με τον ίδιο τρόπο όπως στα κανονικά συστήματα αρχείων.

- ***Διαφάνεια στην Τοποθεσία των Αρχείων***

  Ο χώρος ονομάτων(Namespace) που χρησιμοποιείται για τα αρχεία πρέπει να είναι συνεχής και να ισχύει για όλους τους κόμβους του δικτύου. Η συνοχή του Namespace είναι απαραίτητη προϋπόθεση για την σωστή λειτουργία του συστήματος αρχείων.

- ***Συνέπεια στα Δεδομένα***

  Όλοι οι πελάτες πρέπει να μπορούν να βλέπουν τα ίδια δεδομένα. Ακόμα και όταν υπάρχει μία ενημέρωση, πρέπει η ενημέρωση αυτή είτε να είναι διαθέσιμη σε όλους είτε να είναι διαθέσιμη παλαιότερη έκδοση του αντίστοιχου αντικειμένου.

- ***Ανοχή σε Σφάλματα***

  Το σύστημα θα πρέπει να λειτουργεί σωστά μετά από σφάλμα σε κάποιο node. Η ανάκτηση των δεδομένων αλλά και η διατήρηση της συνολικής απόδοσης, ανεξάρτητα από την αποτυχία ενός κόμβου, είναι απαραίτητη εγγύηση που πρέπει να παρέχει το σύστημα.

- ***Ετερογένεια***

  Το σύστημα αρχείων θα πρέπει να λειτουργεί σωστά και αποδοτικά σε υπολογιστές με διαφορετικό υλικό.

- ***Κλιμακωσιμότητα***

  Το σύστημα αρχείων θα πρέπει να είναι εύκολα επεκτάσιμο, ανεξάρτητα από τον αριθμό τον κόμβων που είναι ήδη συνδεδεμένοι. Αυτό περιλαμβάνει την πρόσθεση νέων κόμβων στο σύστημα με ευκολία, χωρίς να επηρεάζεται η εύρυθμη λειτουργία και η υψηλή του απόδοση.

- ***Τοπικότητα της Αναφοράς***

  Η κατανομή των δεδομένων πρέπει να γίνεται με τέτοιο τρόπο ώστε να μπορούμε να ανακτήσουμε προσκείμενα δεδομένα αποδοτικά.

- ***Εξισορρόπηση φόρτου***

  Η κατανομή των δεδομένων θα πρέπει να γίνεται ομοιόμορφα ώστε να μην υπάρχει υπερφόρτωση κάποιου συγκεκριμένου κόμβου, από τους πελάτες του συστήματος.

Έχουν παρουσιαστεί αρκετές εναλλακτικές υλοποιήσεις για τα κατανεμημένα συστήματα αρχείων, όπως το GFS(Google Filesystem)[4], το HDFS (Hadoop Distributed Filesystem)[5], KFS (Kosmos Filesystem) κτλ. Οι υλοποιήσεις αυτές προσφέρουν τις υπηρεσίες που περιγράφηκαν και χρησιμοποιούν το μοντέλο υπολογισμού mapreduce, για την επεξεργασία μεγάλων συνόλων δεδομένων. Εστιάζουμε στο HDFS και στο αντίστοιχο μοντέλο προγραμματισμού, τα οποία θα περιγραφούν αναλυτικά παρακάτω.

3.2 Hadoop Distributed FileSystem(HDFS)

Το Hadoop αποτελεί ένα πλαίσιο εργασίας, ανοιχτού κώδικα, το οποίο προσφέρει εργαλεία για την οργάνωση, διαχείριση και το μετασχηματισμό μεγάλου συνόλου δεδομένων. Το Hadoop αποτελείται από διαφορετικά συστατικά στοιχεία, τα οποία εκτελούν διαφορετικές διεργασίες και προσφέρουν συγκεκριμένες υπηρεσίες στο χρήστη. Αυτά τα εργαλεία αναφέρονται επιγραμματικά στο πίνακα 3.1.

Χαρακτηριστικό συστατικό του Hadoop είναι το HDFS(Hadoop Distributed FileSystem) το οποίο διατηρεί τα χαρακτηριστικά ενός κλασσικού συστήματος αρχείων Unix. Στην πράξη τα πρότυπα του Unix, για την υλοποίηση του HDFS τροποποιήθηκαν με σκοπό την βελτίωση της απόδοσης του συστήματος. Το πρότυπο λειτουργίας του HDFS δανείζεται ορισμένα χαρακτηριστικά από τα ήδη γνωστά κατανεμημένα συστήματα αρχείων, όπως το GFS, το PVFS[6] και το Lustre. Συγκεκριμένα το HDFS αποθηκεύει τα metadata για την διαχείριση των αρχείων και ολόκληρου του συστήματος, σε ορισμένο server που ονομάζεται NameNode. Τα δεδομένα τα διατηρεί ξεχωριστά σε διαφορετικούς servers οι οποίοι ονομάζονται DataNodes και είναι υπεύθυνοι για την ανάκτηση και την αντιγραφή των μπλοκ των αρχείων. Η τεχνική τις αντιγραφής των μπλοκ σε διαφορετικά DataNodes, είναι αντίστοιχη τεχνική που χρησιμοποιεί και το GFS .

Εκτός από το HDFS μαζί με το Hadoop παρέχονται επιπλέον εργαλεία για την εύκολη διαχείριση των δεδομένων και την συγγραφή εφαρμογών (HBase , MAPREDUCE) με τα οποία θα ασχοληθούμε στα επόμενα κεφάλαια.

HDFS	Κατανεμημένο Σύστημα Αρχείων
MapReduce	Μοντέλο Προγραμματισμού για κατανεμημένο υπολογισμό
HBase	Κατανεμημένη Βάση Δεδομένων
Pig	Γλώσσα Ροής Δεδομένων και Framework για παράλληλη εκτέλεση
Hive	Υποδομή Αποθήκης Δεδομένων
ZooKeeper	Υπηρεσία για Κατανεμημένο Συντονισμό
Chukwa	Σύστημα για την Συλλογή Δεδομένων Management
Avro	Σύστημα για Σειριακά Δεδομένα

ΠΙΝΑΚΑΣ 3.1 – ΕΡΓΑΛΕΙΑ HADOOP

3.3 Περιγραφή Λειτουργίας HDFS

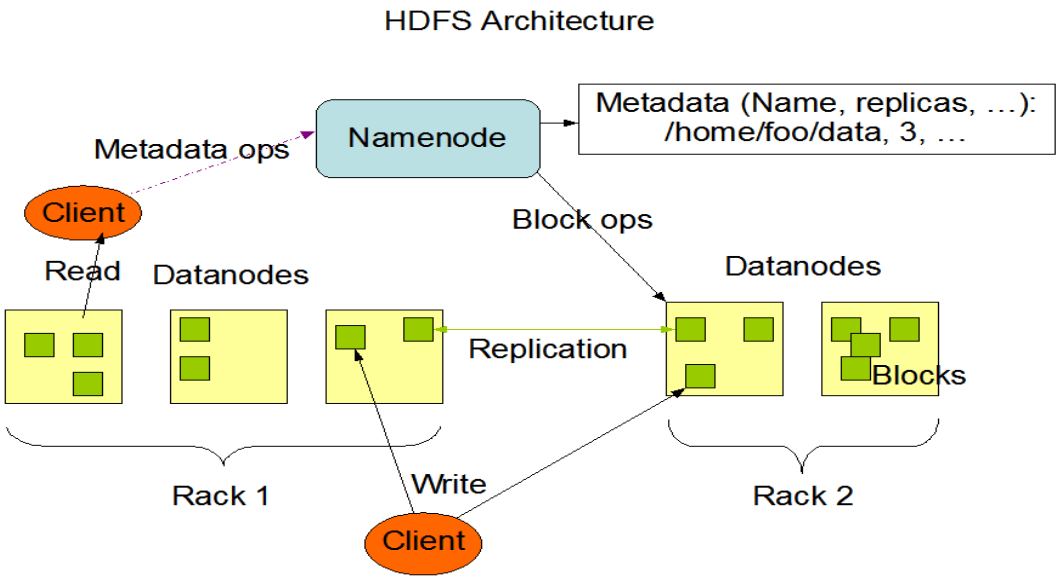
Στο σημείο αυτό, είναι απαραίτητο να παρατίθεται μία εκτενής περιγραφή της αρχιτεκτονικής του HDFS. Η περιγραφή αυτή θα βοηθήσει στην κατανόηση των δυνατοτήτων αλλά και των περιορισμών του συστήματος, ώστε να κατανοηθεί η λειτουργία του αλλά και η ευκολία προσαρμογής του σε διαφορετικές απαιτήσεις.

3.3.1 Hadoop Namenode

Το Namespace του HDFS ορίζεται από μία ιεραρχία αρχείων και φακέλων, οι οποίοι αποθηκεύονται στο NameNode με την μορφή inodes.Τα inodes αποτελούν δομές δεδομένων ευρέως χρησιμοποιούμενες σε Unix συστήματα αρχείων. Στα inodes καταγράφονται χαρακτηριστικά όπως τα permissions, ώρες τροποποίησης, πρόσβασης, μέγεθος και τα λοιπά χαρακτηριστικά ενός αρχείου. Τα περιεχόμενα ενός αρχείου χωρίζονται σε διαφορετικά blocks, (τυπικά μεγέθους 64 Mbytes) και κάθε μπλοκ ενός αρχείου αντιγράφεται σε διαφορετικά DataNodes. Το μέγεθος του μπλοκ είναι τροποποιήσιμη παράμετρος η οποία παίζει σημαντικό ρόλο στην απόδοση του συστήματος. Συγκεκριμένα η επιλογή μεγαλύτερου μεγέθους μπλοκ μειώνει το φορτίο στους DataNodes εξαιτίας των λιγότερων προσπελάσεων για την ανάγνωση τους. Παρόλα αυτά σπαταλάμε χώρο όταν δεσμεύουμε μπλοκ τα οποία δεν χρησιμοποιούνται πλήρως. Το NameNode διατηρεί το Namespace Tree και την τοποθεσία των μπλοκ ενός αρχείου στα αντίστοιχα DataNodes. Είναι υπεύθυνο για την εξυπηρέτηση αιτήσεων, τον έλεγχο της λειτουργίας των DataNodes καθώς και την συνοχή του Namespace.

Όταν ένας Client θέλει να διαβάσει ένα αρχείο, επικοινωνεί με τον NameNode και ζητάει την τοποθεσία των μπλοκ του αρχείου. Συνεχίζει διαβάζοντας τα αντίστοιχα μπλοκ από τον πιο κοντινό σε αυτόν DataNode, εάν αυτός είναι διαθέσιμος αλλιώς προχωράει στον επόμενο. Στην περίπτωση που ένας client χρειάζεται να γράψει ένα αρχείο στο HDFS επικοινωνεί με τον NameNode, ο οποίος επιλέγει κάποια DataNodes για να γράψει τα μπλοκ του αρχείου και αλλά για γράψει τα αντίγραφα των μπλοκ. Στην συνέχεια ο client γράφει με την τεχνική pipelining απευθείας στα αντίστοιχα DataNodes. Ολόκληρο το Namespace διατηρείται στην μνήμη. Τα δεδομένα των inode και η λίστα με τα μπλοκ που ανήκουν σε κάθε αρχείο αποτελούν τα μεταδεδομένα του συστήματος ή αλλιώς την «εικόνα» του συστήματος. Η ενημερωμένη εικόνα του συστήματος αποθηκεύεται στο τοπικό δίσκο του NameNode. Επίσης στον NameNode διατηρείται ένα «ημερολόγιο» (log), το οποίο ονομάζουμε journal και στο οποίο αναφέρονται οι τροποποιήσεις που γίνονται στην εικόνα του συστήματος. Για να έχουμε ανοχή σε σφάλματα δημιουργούνται αντίγραφα των παραπάνω πληροφοριών σε διαφορετικούς κόμβους.

Ο NameNode πέρα από το γεγονός ότι εξυπηρετεί αιτήσεις από clients, εναλλακτικά μπορεί να λειτουργήσει είτε ως CheckpointNode είτε ως BackupNode. Αυτή η επιλογή γίνεται στην αρχικοποίηση του NameNode. Όταν ο NameNode λάβει το ρόλο του CheckpointNode τότε αναλαμβάνει περιοδικά να συνδυάσει την ήδη υπάρχουσα εικόνα του συστήματος με τις αλλαγές που έχουν καταγραφεί στο ημερολόγιο. Συνεχίζει δημιουργώντας νέα εικόνα και νέο άδειο ημερολόγιο. Ο CheckpointNode συνήθως βρίσκεται σε διαφορετικό host καθώς έχει τις ίδιες ανάγκες για μνήμη με τον NameNode. Ο CheckpointNode αποστέλλει στον NameNode την νέα εικόνα του συστήματος που κατασκεύασε. Η χρήση CheckpointNode βοηθάει στην προστασία των μεταδεδομένων του συστήματος. Επίσης η συγχώνευση του ημερολογίου με την εικόνα του συστήματος βοηθάει, καθώς μετά από μεγάλο χρονικό διάστημα αυξάνεται το μέγεθος του ημερολογίου και δημιουργείται πρόβλημα, εξαιτίας της αύξησης της πιθανότητας απώλειας ή καταστροφής του. Επιπλέον το μέγεθος του ημερολογίου επηρεάζει το χρόνο που απαιτείται για να επανακινήσουμε τον NameNode. Όταν ο NameNode λάβει το ρόλο του BackupNode, επιτελεί την ίδια λειτουργία με τον CheckpointNode, με την διαφορά ότι διατηρεί ένα ενημερωμένο αντίγραφο της εικόνας του συστήματος στην μνήμη του, το οποίο είναι συνεπές με το NameNode. Η λειτουργία αυτή είναι πιο αποδοτική, καθώς για να αποθηκεύσει την εικόνα του συστήματος στο τοπικό του δίσκο ο BackupNode, δεν χρειάζεται να επικοινωνήσει με τον NameNode καθώς έχει στην μνήμη την συνεπή εικόνα του συστήματος.



ΣΧΗΜΑ - 3.1 ΑΡΧΙΤΕΚΤΟΝΙΚΗ HDFS

3.3.2 Hadoop DataNode

Κατά την αρχικοποίηση, ένα DataNode συνδέεται με τον NameNode εκτελώντας ένα handshake. Σκοπός είναι να επικυρώσει την έκδοση του συστήματος που χρησιμοποιείται και το namespace ID του. Εάν κάποιο από τα δύο δεν ταιριάζει με αυτά στο NameNode, τότε το αντίστοιχο DataNode απενεργοποιείται. Ένα DataNode που αρχικοποιείται για πρώτη φορά εισέρχεται στο cluster παίρνοντας ένα νέο namespaceID. Οι DataNodes έχουν ένα μοναδικό storage ID, το οποίο χρησιμοποιείται για να τους διαχωρίζουμε ακόμα και όταν έχουμε αλλαγή στην IP. Το storage ID αρχικοποιείται όταν ο DataNode συνδεθεί για πρώτη φορά με τον NameNode και δεν αλλάζει. Κάθε μπλοκ που κατέχει ένας DataNode αποτελείται από δύο αρχεία, ένα που περιέχει τα δεδομένα του μπλοκ και ένα άλλο που περιέχει μεταδεδομένα όπως το checksum του αντίστοιχου μπλοκ. Ο DataNode server αναγνωρίζει ποιά μπλοκ έχει στην κατοχή του και στέλνει μία αναφορά στον NameNode για να τον ενημερώσει. Η αναφορά αυτή περιέχει πληροφορίες όπως το block id, την σφραγίδα δημιουργίας και το μέγεθος του μπλοκ, οποίες αποστέλλονται στην αρχή της εγγραφής του DataNode και στην συνέχεια περιοδικά ανά μία ώρα. Επίσης σε κανονική λειτουργία ο DataNode στέλνει ανά διαστήματα μηνύματα ενημέρωσης στον NameNode, για να τον ενημερώσει ότι λειτουργεί κανονικά. Εάν ο NameNode

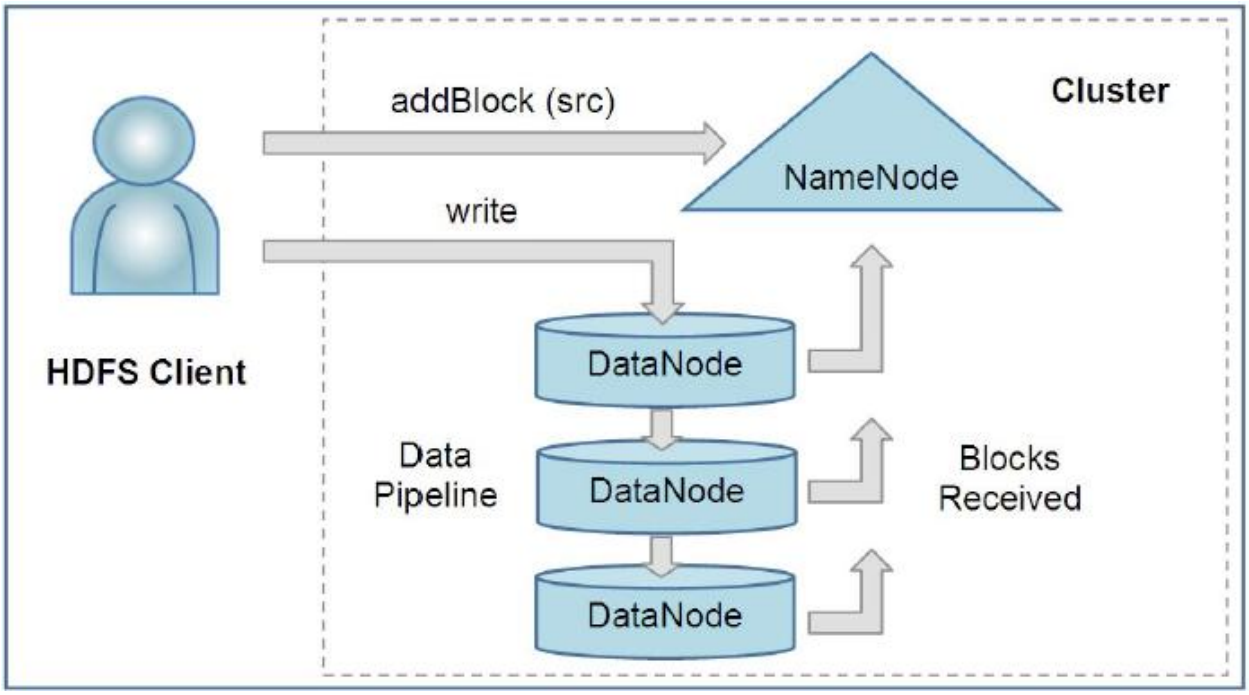
δεν έχει πάρει κάποιο μήνυμα από τον DataNode για περισσότερο από 10 λεπτά τότε θεωρεί ότι είναι εκτός υπηρεσίας, τον διαγράφει και δρομολογεί την δημιουργία νέων DataNodes που θα περιέχουν τα αντίστοιχα μπλοκ. Τα μηνύματα που χρησιμοποιούνται για την επιβεβαίωση της λειτουργίας του DataNode, περιέχουν και πληροφορίες για την χωρητικότητα του αποθηκευτικού χώρου που είναι διαθέσιμος ή χρησιμοποιείται καθώς και τις εκκρεμείς αιτήσεις για τα μπλοκ. Ο NameNode δεν καλεί άμεσα τα DataNodes, αλλά χρησιμοποιεί τις απαντήσεις στα μηνύματα επιβεβαίωσης για να στείλει οδηγίες όπως, αντιγραφή των μπλοκ σε άλλους κόμβους, διαγραφή τοπικών αντιγράφων, τερματισμό λειτουργίας του κόμβου και αποστολή αναφοράς για τα μπλοκ.

Κάθε DataNode ελέγχει περιοδικά την εγκυρότητα των μπλοκ που κατέχει χρησιμοποιώντας τον μπλοκ scanner, ο οποίος και συγκρίνει τα checksums στα αντίγραφα των άλλων κόμβων για να επικυρώσει την εγκυρότητα τους. Όταν εντοπιστεί ένα καταστραμμένο μπλοκ τότε ενημερώνεται ο NameNode, ο οποίος μαρκάρει το αντίστοιχο μπλοκ για διαγραφεί Το διαγράφει αφού δημιουργήσει ένα αντίγραφο του μπλοκ χωρίς λάθη. Έτσι ακόμα και όταν τα δεδομένα είναι λανθασμένα επιτρέπεται στο χρήστη να τα ανακτήσει άμα χρειάζεται.

3.3.3 HDFS Client

Οι εφαρμογές του χρήστη έχουν πρόσβαση στο HDFS μέσο του HDFS client ο οποίος με την σειρά του χρησιμοποιεί την βιβλιοθήκη του Hadoop. Η βιβλιοθήκη περιέχει μία διεπαφή οποία χρησιμοποιείται για την πρόσβαση σε αρχεία του συστήματος. Δεν υπάρχει ανάγκη για τον χρήστη να ξέρει πώς λειτουργεί το σύστημα, οπότε η διεπαφή αυτή υποστηρίζει διαφανή πρόσβαση στο σύστημα αρχείων. Όταν ένας client διαβάζει ένα αρχείο, ζητάει από τον NameNode μία λίστα με τους DataNodes που περιέχουν τα αντίγραφα των μπλοκ του αρχείου. Στην συνέχεια επικοινωνεί άμεσα με τους DataNodes για την ανάκτηση των αντίστοιχων μπλοκ. Αντίστοιχα όταν ένας client θέλει να γράψει, ζητάει από τον NameNode να επιλέξει τους DataNodes που θα αποθηκευτούν τα αντίγραφα του πρώτου μπλοκ. Συνεχίζει χρησιμοποιώντας την τεχνική pipelining για την εγγραφή του αντίστοιχου στο DataNode που επέλεξε. Με το τέλος της εγγραφής, ο client ζητάει ξανά από τον NameNode να επιλέξει τους DataNodes που θα φιλοξενήσουν το επόμενο μπλοκ. Η διαδικασία αυτή συνεχίζεται μέχρι όλα τα μπλοκ του

αρχείου γραφτούν στο HDFS. Το σχήμα 3.2 περιγράφει την διαδικασία που ακολουθεί ο client για την εγγραφή ενός αρχείου.



ΣΧΗΜΑ 3.2 - ΕΠΙΚΟΙΝΩΝΙΑ ΠΕΛΑΤΗ ΜΕ HDFS

Η Βιβλιοθήκη που παρέχεται για την επικοινωνία με το HDFS, δίνει επιπλέον την δυνατότητα για αναγνώριση της τοποθεσίας των μπλοκ στο HDFS. Αυτό επιτρέπει σε εφαρμογές, όπως πχ MapReduce, να εντοπίζουν την τοποθεσία των αρχείων και να χρησιμοποιούν την πληροφορία αυτή για την οργάνωση των εργασιών με σκοπό την βελτίωση της απόδοσης. Επιπλέον η βιβλιοθήκη επιτρέπει να τον ορισμό του παράγοντα αντιγραφής (replication factor) για τα αρχεία. Η αύξηση της απόδοσης του συστήματος επιτυγχάνεται με την αύξηση των αντιγράφων. Έτσι μπορούμε σε αρχεία που προσπελαύνονται συχνά να αυξήσουμε το replication factor.

3.3.4 Στρατηγική Τοποθέτησης Αντιγράφων

Ο NameNode Server, εξασφαλίζει ότι κάθε μπλοκ έχει τον απαιτούμενο αριθμό αντιγράφων. Εντοπίζει εάν ένα μπλοκ έχει λιγότερα ή περισσότερα αντίγραφα και φροντίζει για την δημιουργία ή την διαγραφή των αντίστοιχων μπλοκ. Όταν υπάρχει ανάγκη για διαγραφή

ενός μπλοκ, ο NameNode προτιμάει αρχικά να διαγράψει ένα αρχείο χωρίς να μειώσει τον αριθμό των απαιτούμενων αντιγράφων σε ένα rack, αλλιώς διαγράφει ένα αντίγραφο από το κόμβο με το λιγότερο χώρο στο δίσκο. Ο στόχος είναι να εξισορροπήσουμε την χρήση του αποθηκευτικού χώρου και ταυτόχρονα να διατηρήσουμε την διαθεσιμότητα των μπλοκ. Όταν ένα μπλοκ έχει λιγότερα από τα απαραίτητα αντίγραφα, τοποθετείτε στην ουρά προτεραιότητας αντιγράφων. Η θέση στην οποία τοποθετείτε εξαρτάται από τον αριθμό των ήδη υπαρχόντων αντιγράφων. Περιοδικά εξετάζουμε την ουρά για να αποφασίσουμε που θα τοποθετήσουμε τα μπλοκ προς αντιγραφή. Η στρατηγική τοποθέτησης που ακολουθείτε είναι ίδια με την διαδικασία για την τοποθέτηση νέων μπλοκ. Ελέγχουμε τον αριθμό των αντιγράφων που υπάρχουν και φροντίζουμε να τα τοποθετήσουμε σε διαφορετικό DataNode. Επίσης ελέγχουμε εάν τα υπάρχοντα αντίγραφα υπάρχουν σε DataNodes στο ίδιο rack. Αν ναι, τότε τοποθετούμε το νέο αντίγραφο σε διαφορετικό rack, αλλιώς το τοποθετούμε στο ίδιο rack.

Τέλος, ο NameNode φροντίζει να μην βρίσκονται όλα τα αντίγραφα των μπλοκ στο ίδιο rack. Εάν ο NameNode εντοπίσει ότι τα αντίγραφα ενός μπλοκ βρίσκονται στο ίδιο rack. τότε θεωρεί ότι το αντίστοιχο μπλοκ έχει λιγότερα αντίγραφα και το τοποθετεί στην ουρά προτεραιότητας.

3.3.5 Balancer

Ο Balancer είναι υπεύθυνος για τις στρατηγικές παραγωγής και τοποθέτησης αντιγράφων στους DataNodes. Ο Balancer δεν λαμβάνει υπόψη του το ποσοστό της χρήσης του δίσκου για τον διαμοιρασμό των αντιγράφων. Αυτό γίνεται για την αποφυγή της ανάθεση δεδομένων σε ένα μικρό υποσύνολο των κόμβων του συστήματος, εξαιτίας της μη ομοιόμορφης κατανομής των δεδομένων. Όταν χρειάζεται η ανακατανομή των δεδομένων στους DataNodes, ο Balancer τρέχει σαν εφαρμογή με δικαιώματα διαχειριστή και επαναληπτικά μεταφέρει δεδομένα από κόμβους με αυξημένο φόρτο σε κόμβους με χαμηλότερη κίνηση. Ο Balancer εγγυάται ότι με την ενέργεια αυτή δεν μειώνεται το πλήθος των αντιγράφων. Εάν τα δεδομένα προς αντιγραφή έχουν προορισμό-κόμβο σε διαφορετικό rack, ο balancer ελέγχει εάν υπάρχει κόμβος στο ίδιο rack με τα αντίστοιχα δεδομένα. Με αυτό τον τρόπο επιλέγει να κάνει την μεταφορά μεταξύ γειτονικών κόμβων, μειώνοντας την απαιτουμένη χρήση εύρους ζώνης .

3.3.6 I/O Λειτουργίες

Μία εφαρμογή προσθέτει δεδομένα στο HDFS, δημιουργώντας ένα νέο αρχείο στο οποίο γράφει. Με το κλείσιμο του νέου αρχείου τα δεδομένα που γράφονται δεν μπορούν να διαγραφούν. Νέα δεδομένα προσθέτουμε στο φάκελο με την διαδικασία append. Για την εγγραφή και το διάβασμα δεδομένων το HDFS ακολουθεί το μοντέλο single writer-multiple readers. Σύμφωνα με την τεχνική αυτή, επιτρέπεται να γράφει μόνο ένας πελάτης κάθε φορά σε ένα αρχείο ,αλλά μπορούν ταυτόχρονα παραπάνω από ένας πελάτες να διαβάζουν ένα αρχείο.

Όταν ένας client θέλει να γράψει σε ένα αρχείο, λαμβάνει μία μίσθωση διασφαλίζοντας έτσι ότι κανένας άλλος client δεν θα γράψει στο αντίστοιχο αρχείο. Ο client περιοδικά ανανεώνει την μίσθωση και όταν κλείσει το αρχείο τότε ανακαλείται. Η μίσθωση φράσσεται από ένα «ασθενές»(soft limit) και «ισχυρό»(hard limit) χρονικό όριο. Μέχρι την λήξη του «ασθενούς» ορίου ο client έχει αποκλειστική πρόσβαση στο αρχείο. Εάν λήξη η μίσθωση και ο client δεν την ανανεώσει, χωρίς να έχει κλείσει το αρχείο τότε ισχύει η «ισχυρή» μίσθωση, η οποία επιτρέπει σε οποιοδήποτε άλλο client να κάνει αίτηση για νέα μίσθωση στο ίδιο αρχείο. Όταν λήξη και η ισχυρή μίσθωση με τον client να μην έχει κάνει ανανέωση ή να μην έχει κλείσει το αρχείο, τότε θεωρείται ότι ο client έχει κλείσει το αρχείο και ακυρώνεται η μίσθωση. Με την εγγραφή των δεδομένων σε ένα αρχείο, δεν υπάρχει καμία εγγύηση ότι αυτά θα είναι διαθέσιμα άμεσα για προσπέλαση μέχρι να κλείσει το αρχείο.

Για την αντιμετώπιση των σφαλμάτων, ο client επιβεβαιώνει την εγκυρότητα του μπλοκ που διαβάζει εξετάζοντας το checksum. Όταν θέλει να γράψει ένα μπλοκ, υπολογίζει ο ίδιος το checksum και το αποστέλλει μαζί με το μπλοκ. Ο DataNode τότε επιβεβαιώνει την εγκυρότητα του μπλοκ ελέγχοντας το checksum. Όταν ο client θέλει να διαβάσει κάποια μπλοκ, παίρνει την λίστα των DataNodes που το περιέχουν από τον NameNode, και στέλνει αιτήσεις ανάγνωσης ανάλογα με την σειρά στο κοντινότερο κάθε φορά κόμβο. Αυτό γίνεται μέχρι κάποιος από τους κόμβους να απαντήσει στην αίτηση του client. Το HDFS επιτρέπει στους clients να διαβάζουν αρχεία τα οποία είναι ανοιχτά για εγγραφή. Επειδή δεν ξέρουμε πιο είναι το τελευταίο μπλοκ που θα γραφτεί στο αρχείο, ο client διαβάζει το τελευταίο μπλοκ για το οποίο έχει ενημερωθεί ο NameNode μέχρι την στιγμή της αίτησης για ανάγνωση. Ο σχεδιασμός του HDFS έχει γίνει με σκοπό την βελτιστοποίηση συστημάτων επεξεργασίας δεδομένων σε συστάδες, αναγκαία για μεθόδους κατανεμημένου υπολογισμού, όπως η μέθοδος mapreduce η οποία απαιτεί αναγνώσεις

και εγγραφές μεγάλου όγκου δεδομένων. Παρόλα αυτά έχει μελετηθεί και η δυνατότητα για την υλοποίηση συστημάτων που υποστηρίζουν πραγματικού χρόνου streaming με υψηλή ρυθμαπόδοση ή τυχαία προσπέλαση σε πραγματικό χρόνο μεγάλων πινάκων (HBase).

4. NoSQL Κίνημα

Στην επιστήμη των υπολογιστών όταν χρησιμοποιούμε τον όρο NoSQL(Not Only SQL), αναφερόμαστε σε μία μεγάλη γκάμα συστημάτων διαχείρισης βάσεων δεδομένων, τα οποία χρησιμοποιούν διαφορετική προσέγγιση από αυτή του σχεσιακού μοντέλου των κλασσικών RDBMS συστημάτων. Τα NoSQL συστήματα αποφεύγουν να χρησιμοποιούν του πίνακες ως την βασική δομή για την οργάνωση δεδομένων. Επίσης η επεξεργασία των δεδομένων δεν γίνεται με την χρήση της SQL. Τα δεδομένα μπορούν να είναι δομημένα, αλλά η προτεραιότητα των συστημάτων NoSQL είναι να μπορούν να ανακτήσουν μεγάλες ποσότητες από αυτά και όχι τις σχέσεις μεταξύ τους.

Η σημασία του όρου Not Only SQL κρύβεται πίσω από την ιδέα ότι και η δύο τεχνολογίες μπορούν να συνυπάρχουν, με την καθεμία να προσφέρει σε διαφορετικές καταστάσεις τις απαραίτητες υπηρεσίες. Εταιρίες όπως Facebook, Amazon, Google, Digg, LinkedIn προσφέρουν υπηρεσίες που χρησιμοποιούν NoSQL συστήματα. Η ανάγκη για την NoSQL εμφανίστηκε για την εξυπηρέτηση των παρακάτω αναγκών:

- ***Κλιμακωσιμότητα & Φθηνό Υλικό***

  Όπως αναφέρθηκε στις προηγούμενες ενότητες, ο όγκος των δεδομένων που καλούμαστε να επεξεργαστούμε αυξάνεται καθημερινά. Η ανάγκη για την εύρεση μίας γενικής λύσης, η οποία θα καλύπτει την συνεχόμενη αύξηση των δεδομένων, οδήγησε στο σχεδιασμό των NoSQL συστημάτων. Οι περισσότερες NoSQL βάσεις δεδομένων, σε αντίθεση με τις σχεσιακές βάσεις δεδομένων, σχεδιάστηκαν ώστε να διατηρούν την λειτουργικότητα και την αποδοτικότητα τους σε κλιμακώσιμο περιβάλλον, χωρίς να βασίζονται στην διαθεσιμότητα υλικού υψηλών επιδόσεων.

- ***Υψηλή Ρυθμαπόδοση***

  Αρκετά NoSQL συστήματα, έχουν υψηλότερη ρυθμαπόδοση από τα κλασσικά RDBMS συστήματα.

- ***Πολυπλοκότητα & Κόστος Συστημάτων***

  Οι NoSQL βάσεις δεδομένων είναι σχεδιασμένες με τέτοιο τρόπο ώστε να προσφέρουν χαμηλό κόστος σε περίπτωση που είναι αναγκαία η επέκταση του συστήματος. Επίσης προσφέρουν δυνατότητα εύκολης διαχείρισης και συντήρησης της συστοιχίας των υπολογιστικών μηχανημάτων.

- ***Διασυνδεδεμένα Δεδομένα***

  Η διασύνδεση και η συσχέτιση των δεδομένων συνεχίζει να υπάρχει. Δεδομένου του αυξημένου όγκου τους, τα κλασσικά RDBMS μοντέλα δεν μπορούν με αποδοτικό τρόπο να διατηρήσουν αυτές τις συσχετίσεις. Το NoSQL μοντέλο μπορεί να διατηρεί πολύπλοκες συσχετίσεις, μεταξύ των δεδομένων ανεξάρτητα από το συνολικό μέγεθος τους.

- ***Πολύπλοκες Δομές Δεδομένων***

  Η τεχνολογία NoSQL μπορεί να διαχειριστεί ιεραρχικά εμφωλευμένες δομές δεδομένων με ευκολία. Για να προσφέρουμε τα ίδια αποτελέσματα και υπηρεσίες με τις κλασσικές RDBMS βάσεις δεδομένων χρειαζόμαστε πολύπλοκες σχέσεις, μεταξύ διαφορετικών πινάκων. Αυτό επιδρά αρνητικά στην απόδοση των RDBMS συστημάτων ειδικά όταν αυξάνεται το μέγεθος των πινάκων αυτών.

4.1 Κατηγορίες NoSQL Συστημάτων

Οι NoSQL βάσεις δεδομένων κατηγοριοποιούνται ανάλογα με τον τρόπο που οργανώνουν και αποθηκεύουν τα δεδομένα. Μερικές από τις διαφορετικές κατηγορίες των NoSQL συστημάτων είναι οι εξής:

1. ***Key-Value Stores***

   Αποτελούν αποθήκες δεδομένων οι οποίες λειτουργούν με την χρήση ενός hashtable, στον οποίο αποθηκεύονται μοναδικά κλειδιά και δείκτες για τα δεδομένα του κάθε αντικειμένου. Τέτοιου τύπου αποθήκες δεδομένων προσφέρουν υπηρεσίες για την διαχείριση δεδομένων χωρίς τον ορισμό συγκεκριμένου σχήματος. Οι αντιστοιχίσεις των

κλειδιών σε δείκτες συνδυάζονται με μηχανισμούς caching για την μεγιστοποίηση της απόδοσης. Παραδείγματα key-value stores είναι το σύστημα Dynamo του Amazon.

2. ***Column Family Stores***

   Δημιουργήθηκαν με σκοπό την αποθήκευση και την επεξεργασία μεγάλης ποσότητας δεδομένων, τα οποία είναι κατανεμημένα σε πολλούς διαφορετικούς υπολογιστικούς κόμβους. Υπάρχουν και στην περίπτωση αυτή κλειδιά το οποία δείχνουν σε διαφορετικά σύνολα στηλών. Οι γραμμές διαχωρίζονται από κλειδιά και οι στήλες χωρίζονται σε οικογένειες. Παραδείγματα Column Family Stores είναι το BigTable της Google και το HBase, το οποίο είναι μία ανοιχτού κώδικα(open source) υλοποίηση του BigTable.

3. ***Document Databases***

   Είναι παρόμοια με τα key – value stores . Το μοντέλο ουσιαστικά διαχειρίζεται εκδόσεις αρχείων, τα οποία αποτελούν συλλογές από κλειδιά άλλων αρχείων.

4. ***Graph Databases***

   Βασίζονται στην θεωρία των γράφων για την κατασκευή συστημάτων με κόμβους, οι οποίοι τοποθετούνται ανάλογα με τις σχέσεις που προκύπτουν μεταξύ των δεδομένων. Χρησιμοποιείται ένα εύκαμπτο μοντέλο γραφημάτων, για την εύκολη κλιμάκωση του συστήματος.

4.2 Περιγραφή HBase

Το HBase ξεκίνησε σαν μία open source υλοποίηση του αντίστοιχου συστήματος Big Table της Google. Τα δεδομένα στο HBase οργανώνονται σε οικογένειες στηλών, οι οποίες αντιστοιχίζονται σε διαφορετικές γραμμές του πίνακα. Με αυτό τον τρόπο προκύπτουν «αραιοί» (δεν υπάρχουν τιμές στα χαρακτηριστικά όλων των στηλών κάθε γραμμής)(sparse) πίνακες στα πρότυπα του BigTable της Google. Η οργάνωση αυτή βοηθάει στην αποδοτική ανάκτηση και ενημέρωση των δεδομένων σε πραγματικό χρόνο. Εξαιτίας της μορφής των πινάκων μπορούμε να συμπιέσουμε τα δεδομένα κερδίζοντας σε αποθηκευτικό χώρο. Η υλοποίηση του HBase

αποτελεί μία κλιμακώσιμη λύση για την οργάνωση και την συσχέτιση των δεδομένων με τέτοιο τρόπο ώστε να προσφέρονται στους χρήστες αποδοτικές υπηρεσίες διαχείρισης βάσεων δεδομένων.

Όπως αναφέρθηκε και στις προηγούμενες ενότητες το HBase ανήκει στην κατηγορία των NoSQL συστημάτων, συγκεκριμένα στην κατηγορία των Column Family stores, επομένως δεν λειτουργεί με το κλασσικό σχεσιακό μοντέλο των βάσεων δεδομένων. Δεν υποστηρίζει πολύπλοκα ερωτήματα, join λειτουργίες, συναλλαγές και δευτερεύοντες δείκτες. Οι ανάγκες για την εκπλήρωση των υπηρεσιών αυτών μπορούν να καλυφθούν με την χρήση mapreduce εργασιών.

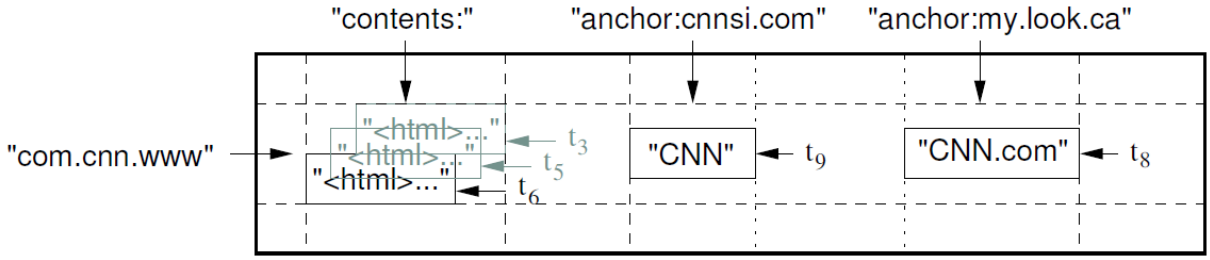
Τα κλασσικά RDBMS συστήματα, απαιτούν το ορισμό πολύπλοκων σχημάτων τα οποία επιδρούν αρνητικά στην συνολική επίδοση. Σημαντική μείωση στην απόδοση εμφανίζεται με την αύξηση των δεδομένων, τα οποία πρέπει να ανακτηθούν. Επίσης εξαιτίας των πολύπλοκων σχέσεων, είναι αδύνατη η επέκταση του συστήματος με την δημιουργία αντιγράφων. Έτσι η λύση που προσφέρουν τα κλασσικά RDBMS συστήματα μπορεί να καλύψει την ανάγκη για επεξεργασία μεγάλου πλήθους δεδομένων.

Το HBase λειτουργεί αποδοτικά με αδόμητα δεδομένα, σε αντίθεση με τα κλασσικά RDBMS συστήματα, γεγονός που σημαίνει ότι αποτελεί μία λύση στο πρόβλημα της συνεχόμενης αύξησης των προς επεξεργασία δεδομένων. Σημαντικό σημείο για την επιλογή του HBase είναι η δυνατότητα κατασκευής συστοιχίας υπολογιστών, η οποία συνδυάζει το χαμηλό κόστος με την υψηλή επίδοση του συστήματος. Στην συνέχεια θα αναφερθούμε αναλυτικότερα στις επιμέρους λειτουργίες αλλά και την αρχιτεκτονική του συστήματος.

4.2.1 Μοντέλο Δεδομένων & Υποστηριζόμενες Λειτουργίες

Τα δεδομένα οργανώνονται σε πίνακες οι οποίοι περιέχουν πολλαπλές οικογένειες στηλών(column families) και γραμμές με μοναδικό κλειδί. Οι γραμμές ταξινομούνται λεξικογραφικά σύμφωνα με το κλειδί, οργανώνονται σε σύνολα που ονομάζονται regions τα οποία περιέχουν πολλαπλές σειρές από συνεχόμενες ταξινομημένες γραμμές. Η ενδεδειγμένη λειτουργία του συστήματος υποστηρίζει πολλές διαφορετικές στήλες οργανωμένες σε μερικές εκατοντάδες οικογένειες στηλών. Επιλέγουμε ένα συγκεκριμένο κελί του πίνακα χρησιμοποιώντας την αντίστοιχη τριπλέτα, η οποία αποτελείται από το κλειδί της γραμμής, το

όνομα της οικογένειας μαζί με το πιστοποιητικό της συγκριμένης στήλης και την χρονική σφραγίδα του αντικειμένου ({rowkey, columnfamily:column,timestamp}⇔ cell). Η οργάνωση των δεδομένων σε μία γραμμή του πίνακα φαίνεται στο σχήμα 4.1.

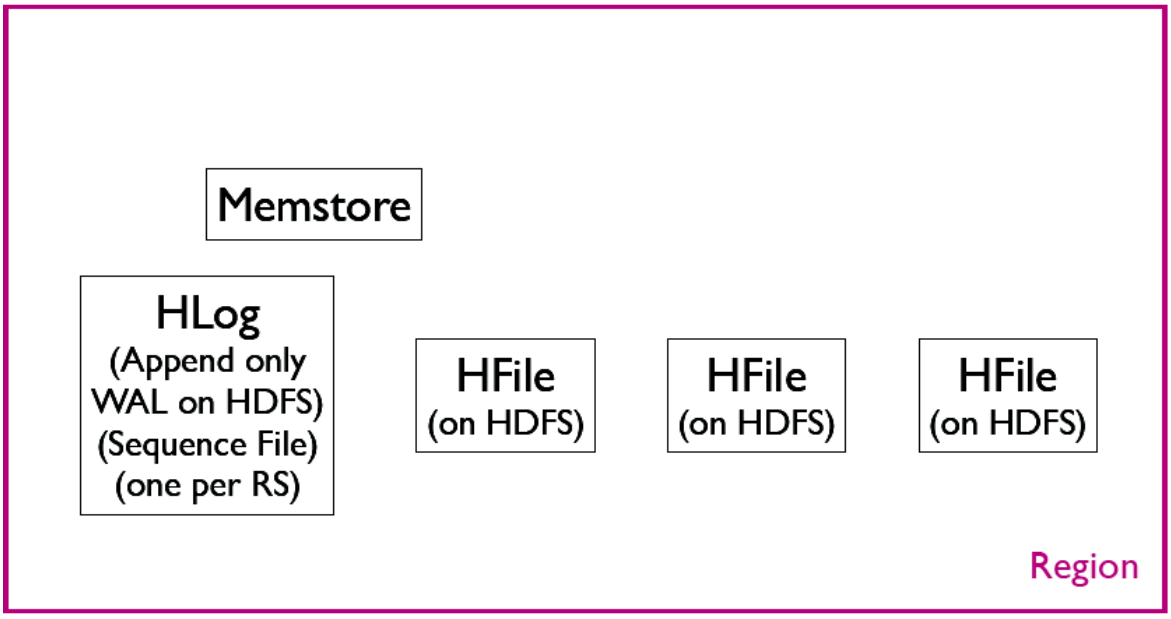


ΣΧΗΜΑ 4.1 – ΠΑΡΑΔΕΙΓΜΑ ΓΡΑΜΜΗΣ ΠΙΝΑΚΑ ΣΤΟ HBASE

Οι λειτουργίες που υποστηρίζονται βασίζονται στα κλειδιά των γραμμών του πίνακα. Υπάρχουν εντολές για την επιστροφή μοναδικής γραμμής, όπως οι εντολές put και get, αλλά και εντολές για την προσπέλαση πολλών γραμμών, όπως οι εντολές scan και multi-row. Δεν υπάρχει υποστήριξη για join, αλλά είναι δυνατή η υλοποίηση τους με map reduce εφαρμογές.

4.2.2 Φυσική Δομή Δεδομένων

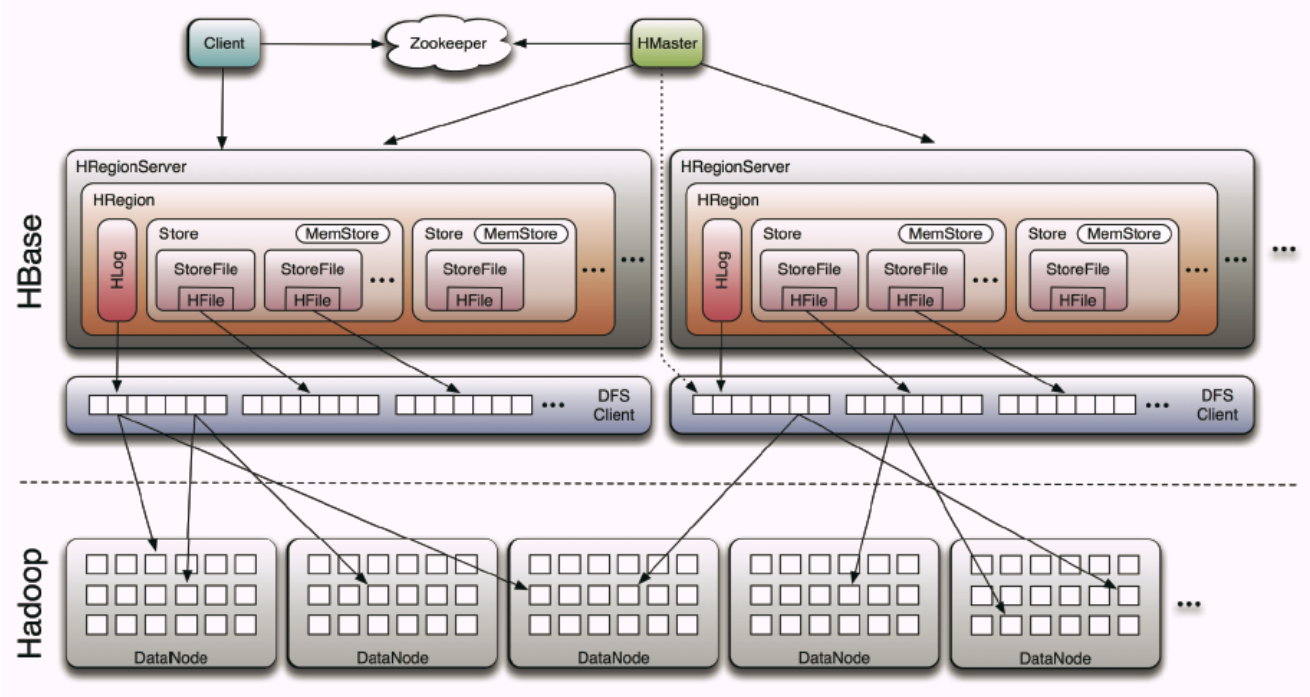
Όπως αναφέρθηκε και παραπάνω τα δεδομένα οργανώνονται σε regions. Τα regions αποτελούνται από store files. Οι στήλες που βρίσκονται στο ίδιο column family αποθηκεύονται στο ίδιο store file. Το store file αποθηκεύει αραιούς πίνακες, χωρίς να υπάρχει ανάγκη για την αναπαράσταση των κενών στηλών. Τα regions διαχωρίζονται όταν μεγαλώσουν υπερβολικά. Μπορούμε να ορίσουμε το μέγιστο μέγεθος ενός region. Αν ορίσουμε μικρό μέγεθος τότε χάνουμε τα πλεονεκτήματα που προκύπτουν, από την παραλληλοποίηση των λειτουργιών. Αντίθετα εάν επιλέξουμε μεγάλο μέγεθος για το region, τότε το σύστημα γίνεται πιο αργό. Στο σχήμα 4.2 φαίνεται η αρχιτεκτονική ενός HRegionServer του HBase.



ΣΧΗΜΑ 4.2 - ΑΡΧΙΤΕΚΤΟΝΙΚΗ HBASE HREGIONSERVER

4.2.3 Αρχιτεκτονική Συστήματος

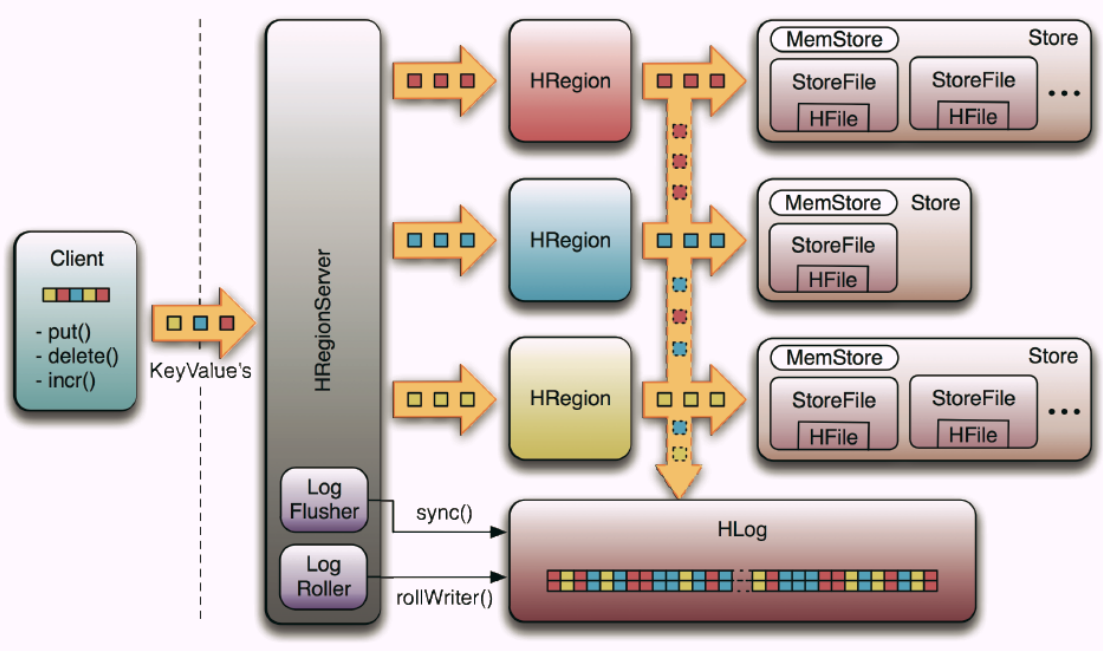
Η αρχιτεκτονική του HBase είναι παρόμοια με αυτή του HDFS. Το σύστημα αποτελείται από έναν Master και από ένα σύνολο από region servers. Παρόμοια με το HDFS η επεξεργασία των δεδομένων γίνεται με την εκτέλεση map reduce εφαρμογών. Ο Master αποφασίζει για την ανάθεση των region ενός πίνακα στους region servers. Επίσης διαχειρίζεται τις αλλαγές στο σχήμα των πινάκων. Το φορτίο που διαχειρίζεται είναι «ελαφρύ». Ο Region Server αποθηκεύει πολλαπλά regions του πίνακα και διατηρεί ημερολόγιο με τις αντίστοιχες ενημερώσεις που γίνονται στο region που κατέχει. Οι αλλαγές πρώτα γράφονται στο ημερολόγιο του Region Server και στην συνέχεια εφαρμόζονται στο region.



ΣΧΗΜΑ 4.3 - ΑΡΧΙΤΕΚΤΟΝΙΚΗ HBASE & ΑΛΛΗΛΕΠΙΔΡΑΣΗ ΜΕ ΤΟ HADOOP

Ο client για να προσπελάσει μία γραμμή του πίνακα κάνει αίτηση στον master. Ο Master επικοινωνεί με τον client επιστρέφοντας την τοποθεσία του HRegionServer που κατέχει την αντίστοιχη γραμμή του πίνακα. Στην συνέχεια ο client επικοινωνεί απευθείας με τον Region Server από τον οποίο ζητάει την αντίστοιχη γραμμή. Εάν δημιουργηθεί πρόβλημα με τους Region Servers, τότε ο client επικοινωνεί ξανά με τον Master. Το HBase χρησιμοποιεί τα παρακάτω κομμάτια από το Hadoop.

- Ειδικές RPC λειτουργίες του Hadoop
- MapFile για τα HStoreFiles
- Sequence File για τα logs



ΣΧΗΜΑ 4.4 - ΕΠΙΚΟΙΝΩΝΙΑ ΠΕΛΑΤΗ ΜΕ ΤΟ HBASE

Στην εικόνα 4.4 βλέπουμε πώς ένας client επικοινωνεί με το HBase για να εκτελέσει λειτουργίες όπως put, get, delete. Ο Client επικοινωνεί με τον Master ο οποίος θα αναζητήσει τον αντίστοιχο HRegionServer, που περιέχει το αντίστοιχο Region του πίνακα και θα επιστρέψει την διεύθυνση του στον Client. Στην συνέχεια ο Client συνδέεται αποκλειστικά με τον Region Server και εκτελεί τις ενέργειες που επιθυμεί στο αντίστοιχο κομμάτι του πίνακα.

5.Κατανεμημένος Υπολογισμός

Ο κατανεμημένος υπολογισμός είναι ο τομέας της πληροφορικής που μελετάει τα κατανεμημένα συστήματα. Ένα κατανεμημένο σύστημα αποτελείται από αυτόνομα υπολογιστικά συστήματα, τα οποία επικοινωνούν μεταξύ τους με μηνύματα μέσο δικτύου. Η συνεργασία των υπολογιστικών συστημάτων είναι καθοριστικής σημασίας για την επίτευξη συγκεκριμένων εργασιών. Η διαφορά του κατανεμημένου υπολογισμού σε σχέση με τον παράλληλο υπολογισμό έγκειται στο γεγονός, ότι η επικοινωνία τον διεργασιών γίνεται μέσο δικτύου και όχι μέσο κοινής μνήμης που είναι προσπελάσιμη από όλους του υπολογιστικούς πόρους.

Η επιλογή για κατανεμημένο υπολογισμό επιβάλλεται σε δύο περιπτώσεις. Εάν από την φύση της η εφαρμογή που θέλουμε να εκτελεστεί χρειάζεται δεδομένα από κάποια άλλη φυσική τοποθεσία πχ έναν υπολογιστή σε διαφορετική γεωγραφική περιοχή. Σε άλλη περίπτωση, υπάρχουν εφαρμογές, όπου ο κατανεμημένος υπολογισμός είναι πιο αποδοτικός χρονικά για την παραγωγή αποτελεσμάτων. Σημαντικό σημείο στο κατανεμημένο υπολογισμό είναι η αξιοπιστία του καθώς δεν υπάρχουν μοναδικά σημεία αποτυχίας (single points of failure).

5.1 Το Μοντέλο Προγραμματισμού MapReduce

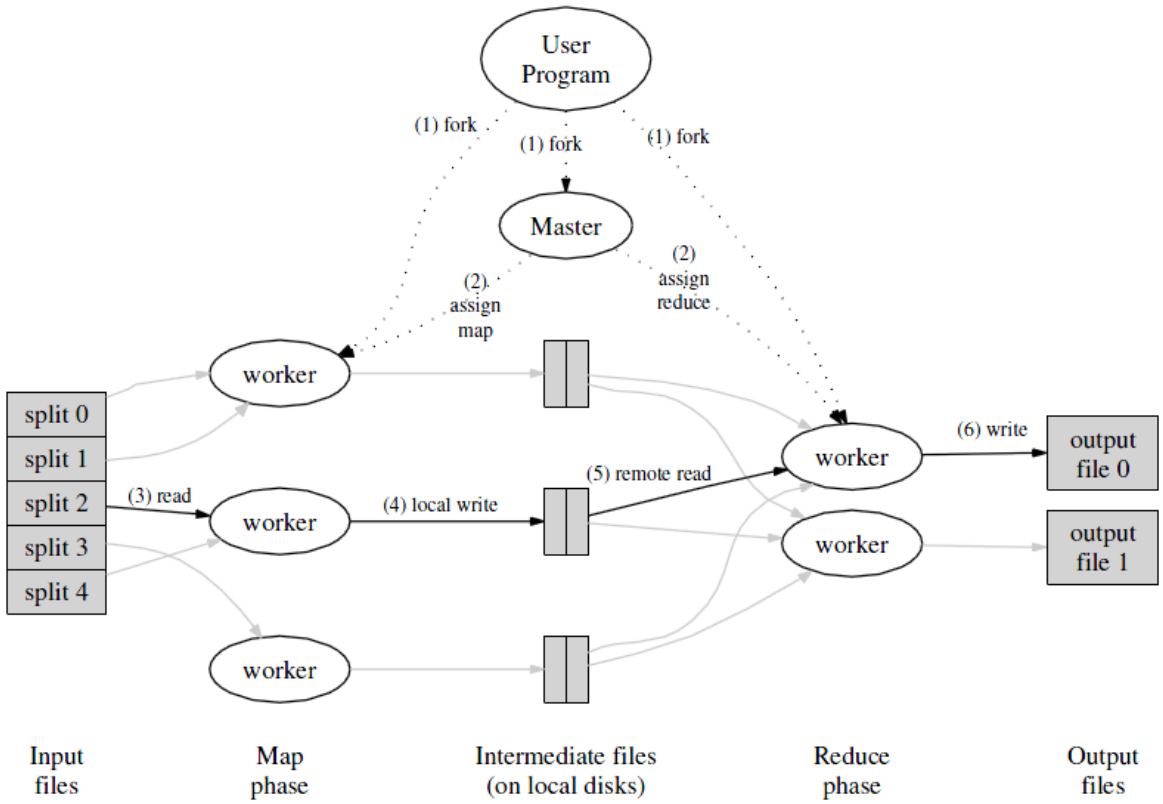
Τα τελευταία χρόνια , η μέθοδος προγραμματισμού mapreduce έχει αναδειχθεί, σαν μία από τις πιο ευρέως χρησιμοποιούμενες μεθόδους για την επεξεργασία δεδομένων της τάξεως των TERABYTE και PETABYTE. Επιτρέπει εύκολη παραλληλοποίηση πάνω από πολλαπλά διαφορετικά μηχανήματα. Επίσης έχουν γίνει αρκετές επιτυχείς προσπάθειες για να χρησιμοποιηθεί για γενικότερους υπολογισμούς πέρα από την επεξεργασία δεδομένων.

Ο χρήστης στο μοντέλο αυτό, προσδιορίζει μία συνάρτηση map και μία συνάρτηση reduce οι οποίες θα εκτελεστούν στα μηχανήματα που είναι διαθέσιμα στο cluster. Το σύστημα, πάνω από το οποίο εκτελούνται τα προγράμματα, αναλαμβάνει το διαμοιρασμό των δεδομένων, χρονοπρογραμματίζει την εκτέλεση του προγράμματος πάνω από το cluster, διαχειρίζεται τις αποτυχίες των μηχανημάτων και την επικοινωνία μεταξύ τους. Το mapreduce υλοποιήθηκε ώστε να τρέχει πάνω από εκατοντάδες μηχανήματα κάθε φορά και είναι ειδικά σχεδιασμένο

διατηρείται η απόδοση του με την επέκταση σε χιλιάδες μηχανήματα. Το μοντέλο προγραμματισμού mapreduce είναι εμπνευσμένο από τις συναρτήσεις map και reduce, οι οποίες χρησιμοποιούνται από το functional programming μοντέλο. Η χρήση τους παρόλα αυτά έχει διαφορετικό σκοπό από αυτόν του functional programming.

5.2 Διαδικασία Εκτέλεσης MapReduce

Όταν ξεκινήσει η διαδικασία Map τα δεδομένα διαμοιράζονται σε διαφορετικά μηχανήματα (workers), τα οποία αναλαμβάνουν να επεξεργαστούν κομμάτια(splits) από τα δεδομένα. Τα splits μπορούμε να τα επεξεργάζονται παράλληλα από διαφορετικά μηχανήματα. Η διαδικασία reduce ξεκινάει λαμβάνοντας τα δεδομένα που παρήγαγαν οι mappers σύμφωνα με το ενδιάμεσο κλειδί. Ο αριθμός των partitions που θα χρησιμοποιηθούν και η συνάρτηση που δημιουργεί τα αντίστοιχα partitions είναι παράμετρος, η οποία καθορίζεται από τον εκάστοτε χρηστή. Στο σχήμα 5.1 φαίνεται η γραφική αναπαράσταση της εκτέλεσης μίας mapreduce εργασίας.



ΣΧΗΜΑ 5.1 - ΓΡΑΦΙΚΗ ΑΝΑΠΑΡΑΣΤΑΣΗ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΟΥ ΜΟΝΤΕΛΟΥ MAREDUCE

Όταν ο χρήστης εκτελέσει την εφαρμογή του εκτελούνται τα παρακάτω βήματα:

1. Η βιβλιοθήκη(MAPREDUCE Library) που είναι υπεύθυνη για την εκτέλεση της mapreduce εργασίας διαμοιράζει τα δεδομένα εισόδου σε M splits. Στην συνέχεια ξεκινάει πολλαπλά αντίγραφα της του προγράμματος σε διαφορετικά μηχανήματα.

2. Ο Master είναι υπεύθυνος για τον έλεγχο της σωστής εκτέλεσης της εφαρμογής. Οι κόμβοι που εκτελούν την mapreduce εργασία ονομάζονται εργάτες (workers). Ο master επιλέγει αδρανής workers οι οποίοι θα εκτελέσουν μία map ή reduce εργασία.

3. Ένας worker που αναλαμβάνει μία map εργασία, παίρνει ένα split διαβάζει από τα δεδομένα ζευγάρια τιμών (key,value) τα οποία και επεξεργάζεται. Στην συνέχεια «εκπέμπει»(emit) στην έξοδο ένα ζευγάρι τιμών.

4. Περιοδικά τα ζευγάρια αυτά γράφονται στο δίσκο μοιρασμένα σε partitions ανάλογα με την συνάρτηση για partitioning που έχει ορίσει ο χρήστης. Η περιοχή που γράφονται τα partitions αποστέλλονται στον master ο οποίος χρησιμοποιεί την πληροφορία αυτή για να προωθήσει τα partitions στους αντίστοιχους workers που θα εκτελέσουν την reduce εργασία.

5. Όταν ένας reducer ενημερωθεί για την τοποθεσία του δικού του partition χρησιμοποιεί remote procedure calls για να το ανακτήσει. Όταν ο reducer διαβάσει όλα τα ενδιάμεσα δεδομένα χρησιμοποιεί τα ενδιάμεσα κλειδιά για να τα ταξινομήσει. Η ταξινόμηση γίνεται για να μοιραστούν εύκολα τα δεδομένα στους αντίστοιχους reducers.

6. Ο worker που έχει αναλάβει να εκτελέσει την συνάρτηση reduce επεξεργάζεται κάθε ενδιάμεσο κλειδί που ανήκει στο partition του και στην συνέχεια εκπέμπει στην έξοδο ένα ζευγάρι (key,value). Τα ζευγάρια αυτά γράφονται σε διαφορετικά αρχεία τα οποία δημιουργεί ο κάθε reducer.

7. Όταν όλες οι εργασίες της εφαρμογής έχουν ολοκληρωθεί ο Master στέλνει μήνυμα στην εφαρμογή του χρήστη για να την ενημερώσει. Στο σημείο αυτό ολοκληρώνεται η διαδικασία MapReduce.

Ένα παράδειγμα MapReduce ψευδό-κώδικα εφαρμογής εμφανίζεται στην εικόνα 5.2. Στο παράδειγμα αυτό υλοποιούμε την wordcount εφαρμογή για η οποία βρίσκει το πλήθος των εμφανίσεων κάθε λέξης σε κάποια κείμενα που παίρνουμε ως είσοδο. Ο mapper παίρνει σαν είσοδο ένα split από τα αντίστοιχα κείμενα. Το key είναι ο αριθμός της γραμμής του κειμένου και το value τα δεδομένα της γραμμής. Σπάμε τις γραμμές σε λέξεις και για έξοδο δίνουμε ένα ζευγάρι (key,value) που είναι η λέξη και η μονάδα αντίστοιχα. Τα ζευγάρια αυτά μοιράζονται στους reducers σύμφωνα με τον default partitioner με την χρήση της τεχνικής hashing. Έτσι κάθε reducer παίρνει όλα τα ζευγάρια, συγκεκριμένων λέξεων. Στην συνέχεια για κάθε λέξη χρησιμοποιείται ένας counter για να υπολογιστεί το πλήθος εμφανίσεων. Τέλος στην έξοδο γράφεται το ζευγάρι (key,value) το οποίο αντιστοιχεί στην λέξη και την συχνότητα εμφάνισης της.

```
Mapper{
	map(key , value):
		words = value.split()
		for word in words:
			output(word,one)
}

Reducer{
	reduce(key,list<value>):
		sum=0
		for v in list<value>:
			sum+=v
		output(key,sum)
}
```

ΣΧΗΜΑ 5.2 - ΠΑΡΑΔΕΙΓΜΑ ΨΕΥΔΟΚΩΔΙΚΑ ΕΦΑΡΜΟΓΗΣ MAPREDUCE

5.3 Ανοχή σε Σφάλματα

Επειδή η βιβλιοθήκη για την εκτέλεση του MapReduce έχει σχεδιαστεί για την επεξεργασία μεγάλου όγκου δεδομένων, παράλληλα με το να χειρίζεται χιλιάδες μηχανήματα πρέπει να μπορεί να διαχειρίζεται τις αποτυχίες των μηχανημάτων αυτών.

- ***Worker Failure***

  Ο Master αποστέλλει περιοδικά μηνύματα στους workers για να εξετάσει την κατάσταση λειτουργίας στην οποία βρίσκονται. Εάν κάποιος worker δεν απαντήσει στο μήνυμα για συγκεκριμένο χρονικό διάστημα, τότε ο master θεωρεί ότι έχει αποτύχει και ακυρώνει την εργασία που του είχε αναθέσει. Στην περίπτωση που ο worker έχει αναλάβει το ρόλο του mapper πρέπει να ενημερώσει και τους reducer που θα χρειαστούν να προσπελάσουν δεδομένα από αυτόν. Αφού ακυρώσει την εργασία στον αντίστοιχο worker, συνεχίζει αναθέτοντας την σε νέους workers. Η τεχνική MapReduce είναι ανθεκτική απέναντι στην αποτυχία μεγάλου πλήθους worker.

- ***Master Failure***

  Ο Master διατηρεί δεδομένα που περιγράφουν σε ποία κατάσταση βρίσκεται η mapreduce εργασία που εκτελείτε. Εάν έχουμε αποτυχία του Master τότε μπορούμε να ξεκινήσουμε καινούργιο, εξετάζοντας τις πληροφορίες που έχουμε για την κατάσταση της εφαρμογής που εκτελείτο.

5.4 Τοπικότητα

Το εύρος ζώνης που χρησιμοποιείται για την επικοινωνία των διεργασιών, θεωρείται πεπερασμένος πόρος. Μειώνουμε την σπατάλη στο εύρος ζώνης εκμεταλλευόμενοι το γεγονός, ότι τα δεδομένα που είναι απαραίτητα για την εκτέλεση της διεργασίας βρίσκονται στους τοπικούς δίσκους των workers. Ο Master κατά την αναζήτηση worker για την ανάθεση της αντίστοιχης εργασίας, εντοπίζει μηχανήματα που περιέχουν αντίγραφα των δεδομένων εισόδου. Εάν αποτύχει στην ανάθεση αυτή, τότε προσπαθεί να αναθέσει την mapreduce εργασία σε

μηχανήματα που είναι πιο κοντά στα δεδομένα. Τα περισσότερα δεδομένα βρίσκονται τοπικά στους δίσκους όταν εκτελείτε μία MapReduce εφαρμογή, έτσι καταναλώνουμε σχεδόν μηδενικό εύρος ζώνης κατά την εκτέλεση μίας mapreduce εργασίας.

6. B+, B Δέντρα

Όλες οι εφαρμογές λογισμικού λειτουργούν με δεδομένα τα οποία πρέπει να είναι αποθηκευμένα σε κάποιο επίπεδο μνήμης. Υπάρχουν διαφορετικά επίπεδα αποθήκευσης στους υπολογιστές , η κύρια μνήμη(συμπεριλαμβανομένου της κρυφής μνήμης) και ο σκληρός δίσκος. Τα δεδομένα που είναι αποθηκευμένα στο πρώτο επίπεδο μπορούν να προσπελαστούν άμεσα από τον επεξεργαστή του συστήματος. Δεν μπορούμε όμως να διατηρούμε όλα τα δεδομένα στην κύρια μνήμη του συστήματος, είτε εξαιτίας του περιορισμένου αποθηκευτικού χώρου και σημαντικού κόστους της είτε γιατί τα δεδομένα δεν μπορούν να διατηρηθούν όταν δεν υπάρχει ρεύμα. Ο σκληρός δίσκος είναι συνήθως φτηνότερος, έχει μεγαλύτερο αποθηκευτικό χώρο και τα αποθηκευμένα δεδομένα μπορούν να διατηρηθούν και με την απουσία ρεύματος. Παρόλα αυτά, το χρονικό κόστος για την προσπέλαση των δεδομένων στο δίσκο είναι δεκαπλάσιο από το αντίστοιχο στην κύρια μνήμη του υπολογιστή. Η οργάνωση των δεδομένων στο δίσκο θα πρέπει να γίνεται με τέτοιο τρόπο ώστε να επιτρέπεται η αποδοτική ανάκτηση τους. Η θεμελιώδης ερώτηση που προσπαθεί να απαντήσει η επιστήμη των βάσεων δεδομένων είναι πως θα γεφυρώσουμε το χάσμα που υπάρχει μεταξύ κύριας μνήμης και σκληρού δίσκου. Μία λύση είναι να χρησιμοποιήσουμε ένα είδος ιεραρχικής δομής δεδομένων με την οποία θα μπορούμε σε λίγα βήματα να εντοπίσουμε το αντίστοιχο μπλοκ που είναι αποθηκευμένη η ζητούμενη πληροφορία.

Τα δέντρα αποτελούν τις ποιο διαδεδομένες δομές δεδομένων που προσομοιώνουν ιεραρχική δομή χρησιμοποιώντας ένα σύνολο από διασυνδεδεμένους κόμβους. Ένα δέντρο μπορεί να οριστεί αναδρομικά σαν μία συλλογή από κόμβους ξεκινώντας από την ρίζα, όπου κάθε κόμβο έχει μία τιμή και ένα σύνολο από παιδιά (κόμβους) με τον περιορισμό ότι κανένας κόμβος δεν είναι αντίγραφο κάποιου άλλου. Υπάρχουν διαφορετικά είδη δέντρων για την κάλυψη συγκεκριμένων αναγκών.

Στην επιστήμη των υπολογιστών, τα B & B+ Trees χαρακτηρίζονται ως γενίκευση των δυαδικών δέντρων, καθώς ένας κόμβος μπορεί να έχει παραπάνω από δύο παιδιά. Τα δέντρα αυτά χρησιμοποιούνται για την αναπαράσταση ταξινομημένων δεδομένων στο σκληρό δίσκο, ώστε να επιτρέπεται η αποδοτική εισαγωγή, ανάκτηση και ενημέρωση τους. Συνήθως τα δέντρα

αυτά χρησιμοποιούνται σε βάσεις δεδομένων και συστήματα αρχείων. Επίσης χρησιμοποιούνται ευρέως και σε κατανεμημένα συστήματα αρχείων.

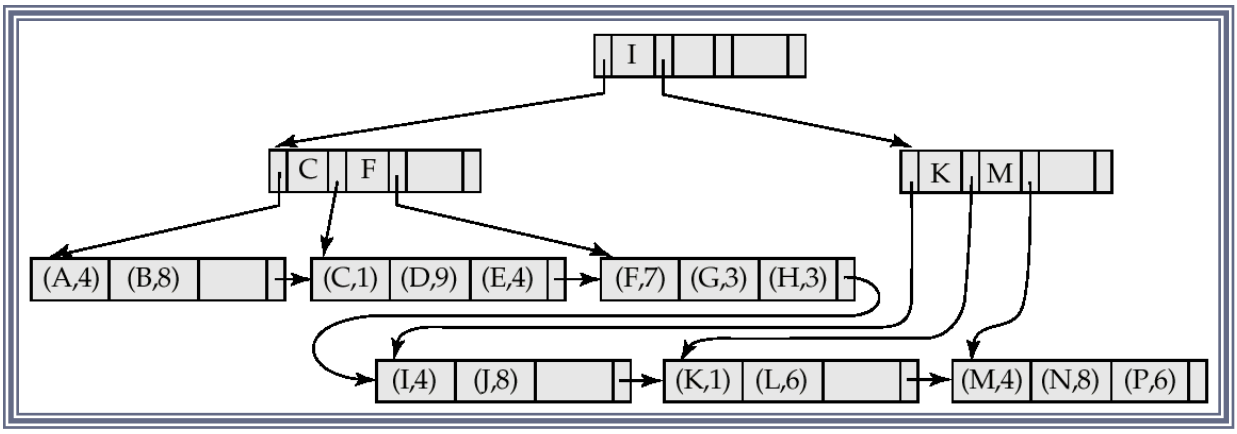
Τα B & B+Trees προσφέρουν τις εξής δυνατότητες: διατηρούν τα δεδομένα ταξινομημένα ώστε να μπορούμε να τα προσπελάσουμε σειριακά, χρησιμοποιούν ιεραρχικό ευρετήριο για την ελαχιστοποίηση των προσπελάσεων στο δίσκο, ο κάθε κόμβος διατηρεί χώρο για την αποδοτική εισαγωγή και διαγραφή δεδομένων, το ευρετήριο τους ενημερώνεται σε λογαριθμικό χρόνο με την χρήση αναδρομικού αλγορίθμου.

6.1 Χαρακτηριστικά B+Tree

Τα B+ Trees αποτελούν μία μορφή ζυγισμένων δέντρων όπου κάθε μονοπάτι από την ρίζα ενός δέντρου έως τα φύλλα του δέντρου έχουν το ίδιο μήκος. Κάθε κόμβος που δεν είναι φύλλο έχει μεταξύ n/2 και n παιδιά, όπου n είναι η τάξη του δέντρου. Τα B+Tree έχουν πολύ καλή απόδοση στο search αλλά δημιουργούν σημαντικό overhead γιατί σπαταλάνε χώρο. Ένας τυπικός κόμβος περιέχει μέχρι το πολύ n-1 κλειδιά και το πολύ n δείκτες(n παιδιά). Τα κλειδιά διατηρούνται στο κόμβο σε σειρά.

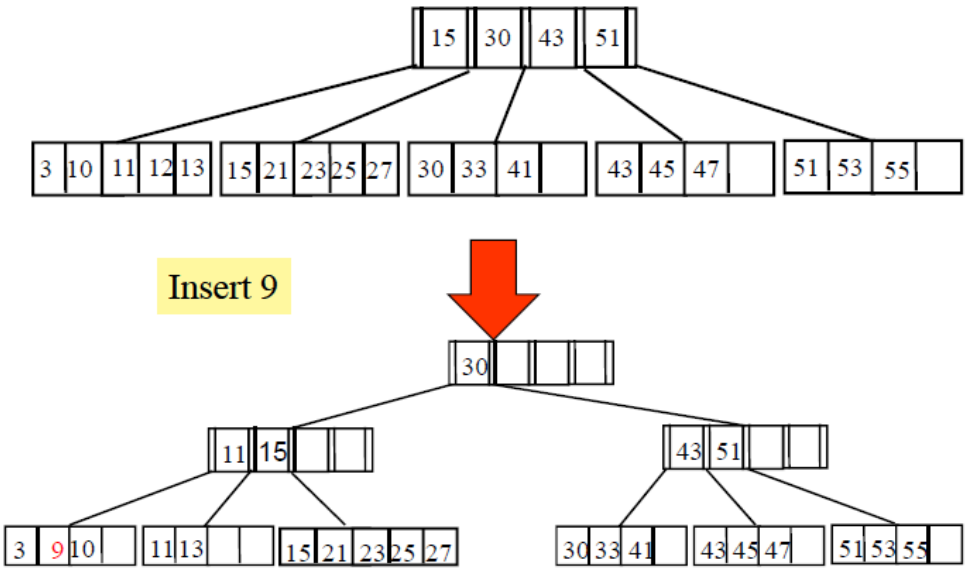
ΣΧΗΜΑ 6.1 – ΠΑΡΑΔΕΙΓΜΑ ΚΟΜΒΟΥ B,B+ TREE

Τα φύλλα στο B+Tree δέντρο διατηρούν pointers στο επόμενο φύλλο. Επίσης στα φύλλα για κάθε κλειδί διατηρείται η τιμή του αντίστοιχου κλειδιού. Οι τιμές στα φύλλα καθορίζονται από δείκτες που δείχνουν είτε στις αντίστοιχες εγγραφές ή σε ένα αρχείο στο οποίο διατηρούνται οι εγγραφές.



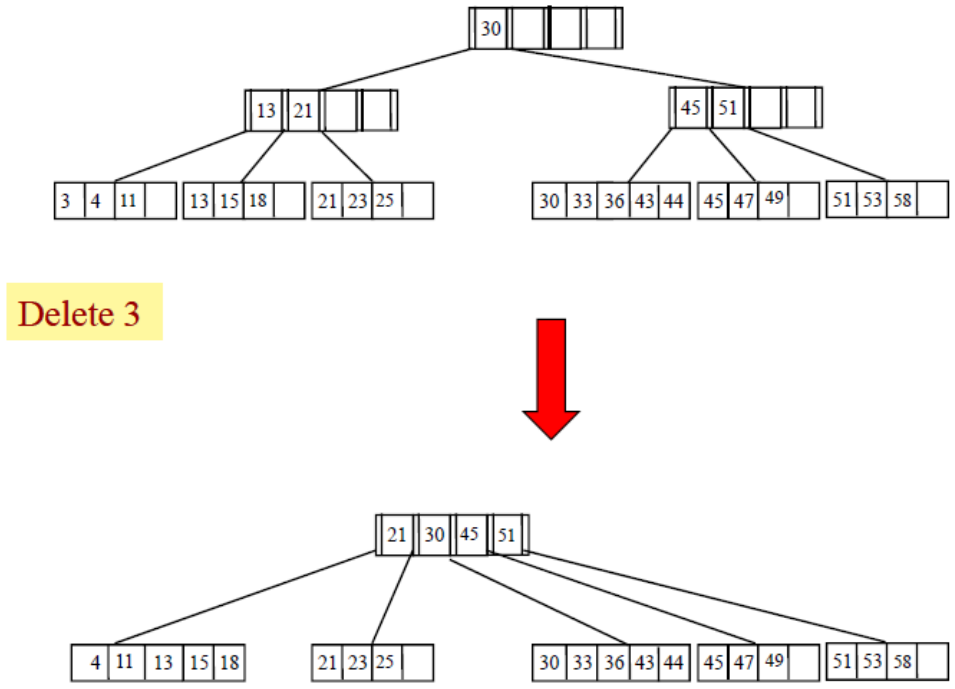
ΣΧΗΜΑ - 6.2 B+TREE

Το B+Tree υποστηρίζει ενέργειες όπως insert, delete, search, update. Όταν εισάγουμε στο δέντρο νέο κλειδί τότε αρχικά βρίσκουμε το φύλλο στο οποίο θα πρέπει να το εισάγουμε. Εάν με την εισαγωγή του νέου κλειδιού ο αριθμός των κλειδιών στο φύλλο είναι μεγαλύτερος της τάξης του δέντρου (δηλαδή > n), τότε επιβάλλεται η διάσπαση του κόμβου σε δύο νέους και η εγγραφή του μεσαίου κλειδιού στον κόμβο πατέρα του κόμβου που διασπάστηκε. Αναδρομικά ελέγχουμε τους κόμβους μέχρι την ρίζα και εάν χρειαστεί τους διασπάμε με αντίστοιχο τρόπο.



ΣΧΗΜΑ 6.3 – ΕΙΣΑΓΩΓΗ ΣΕ B+TREE

Όταν θέλουμε να διαγράψουμε ένα κλειδί από το δέντρο βρίσκουμε το αντίστοιχο φύλλο που περιέχει το κλειδί και το διαγράφουμε . Εάν ο κόμβος μετά την διαγραφή περιέχει λιγότερα από n/2 κλειδιά τότε πρέπει να συμπληρώσουμε το φύλλο με κλειδιά από τον πατέρα ή από το διπλανό φύλλο. Αντίστοιχο παράδειγμα λειτουργίας delete σε δέντρο B+Tree φαίνεται στο παρακάτω σχήμα.

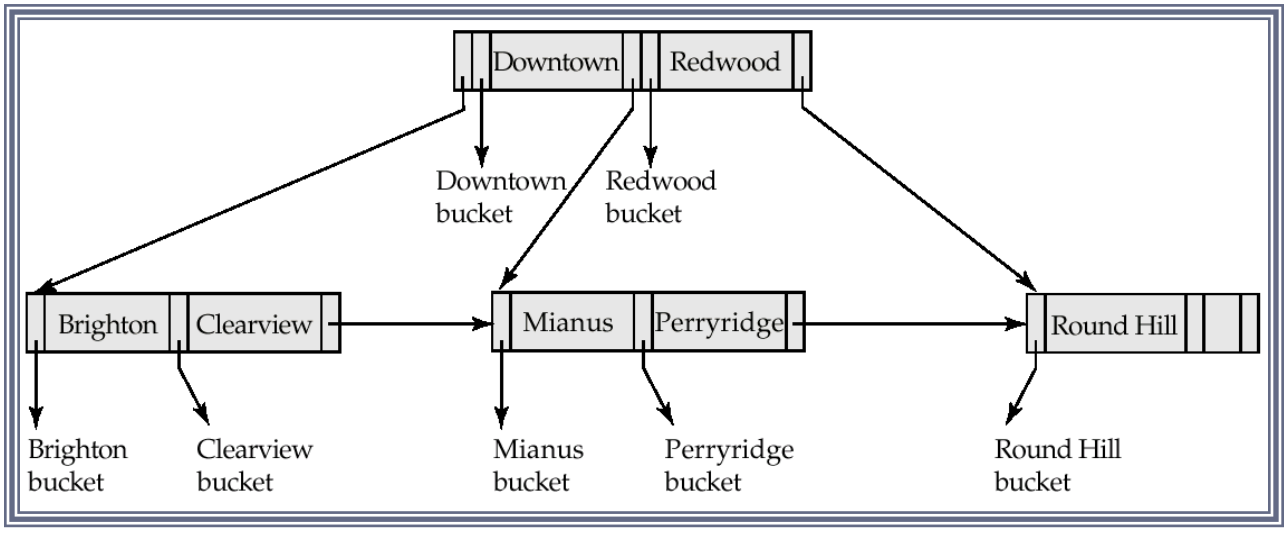


ΣΧΗΜΑ - 6.4 ΔΙΑΓΡΑΦΗ ΑΠΟ B+TREE

6.2 Χαρακτηριστικά B-Tree

Το B-Tree είναι ακριβώς ίδιο με το B+Tree με την διαφορά ότι δεν αποθηκεύουμε τιμές μόνο στα φύλλα και επιπλέον δεν έχουμε δείκτες μεταξύ των φύλλων. Σε κάθε κόμβο αποθηκεύονται n-1 κλειδιά μαζί με τις αντίστοιχες τιμές τους. Τα φύλλα του δέντρου δεν ενώνονται μεταξύ τους με δείκτες. Το B-Tree δέντρο απαιτεί μικρότερο αποθηκευτικό χώρο στο δίσκο από το B+Tree. Κάποιες φορές είναι δυνατόν να βρούμε ένα κλειδί χωρίς να χρειαστεί να φτάσουμε μέχρι τα φύλλα. Τα μειονεκτήματα του B-Tree είναι ότι σε αντίθεση με το B+Tree δεν μπορούμε να εκτελέσουμε αποδοτικά ερωτήματα εύρους, . Επίσης επειδή τα φύλλα είναι

μικρότερα σε μέγεθος δεν έχουμε καλό fan-out, έτσι το βάθος του δέντρου είναι αυξημένο άρα και οι ενέργειες πάνω σε αυτό ολοκληρώνονται σε περισσότερο χρόνο.



ΣΧΗΜΑ - 6.5 B-TREE

Οι ενέργειες που μπορούν να εκτελεστούν πάνω σε ένα B+ ή B- Tree έχουν όλες πολυπλοκότητα $O(\log_B N)$ όπου B = τάξη του δέντρου και N= το σύνολο των κλειδιών στο δέντρο. Για να την επίτευξη καλύτερης πολυπλοκότητας, αρκεί να μειωθεί το ύψος του δέντρου. Αυτό το επιτυγχάνεται αυξάνοντας την τάξη του, δηλαδή τον αριθμό των κλειδιών που μπορεί να αποθηκεύσει ο κάθε κόμβος του δέντρου.

6.3 Παρατηρήσεις & Γενικά Χαρακτηριστικά

Τα B+ και B- Trees χρησιμοποιούνται για την κατασκευή ευρετηρίου δεδομένων που δεν χωράνε στη κύρια μνήμη. Η ιδιαίτερη μορφή τους βοηθάει στην αποθήκευση τους στο δίσκο και στην αποδοτική τους ανάκτηση από αυτόν. Για ένα δέντρο τάξης 31 με ένα εκατομμύριο εγγραφές θα χρειαστούμε το πολύ 5 προσπελάσεις στο δίσκο για να βρούμε ένα κλειδί στο ευρετήριο. Ο αριθμός αυτός μειώνεται καθώς αυξάνουμε την τάξη του δέντρου. Έτσι γλυτώνουμε τις πολλές προσπελάσεις στο δίσκο άρα βελτιώνουμε και το συνολικό χρόνο προσπέλασης των δεδομένων. Παρόλα αυτά καθώς αυξάνουμε την τάξη του δέντρου το μέγεθος του κάθε κόμβου αυξάνεται. Η αύξηση αυτή έχει επίπτωση στο χρόνο ανάκτησης του κάθε

κόμβου καθώς όσο αυξάνεται το μέγεθος του τόσο περισσότερα μπλοκ θα χρειάζεται να ανακτήσουμε για τον αντίστοιχο κόμβο. Επομένως πρέπει να ορίσουμε προσεκτικά την τάξη του δέντρου ώστε να πετύχουμε το καλύτερο δυνατό χρόνο προσπέλασης.

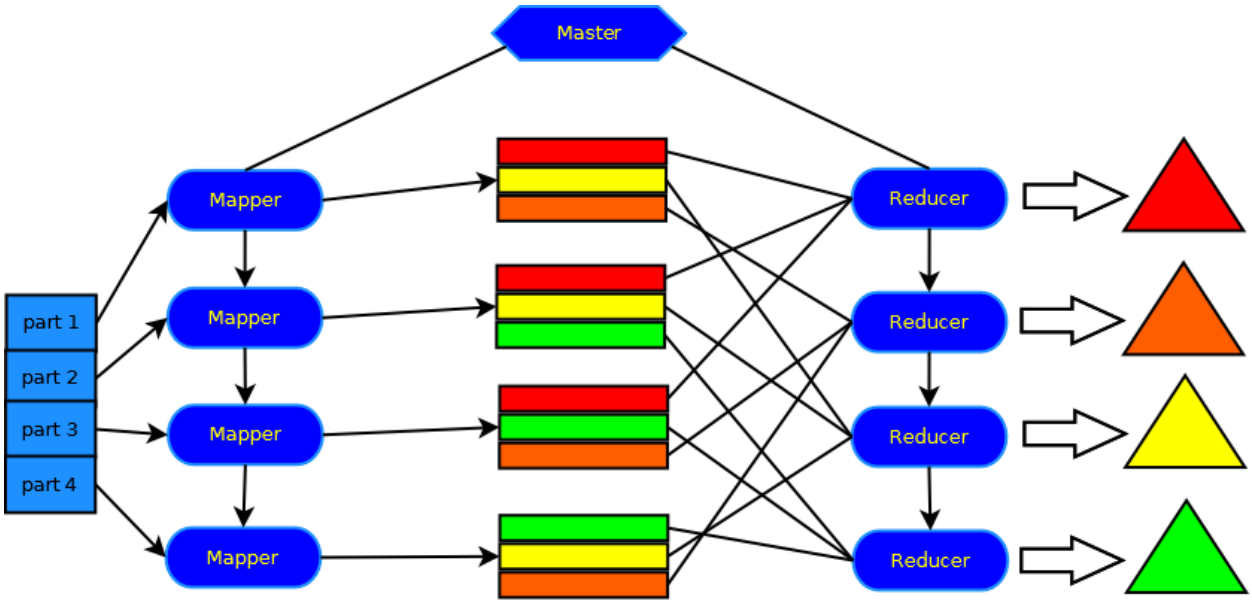
Όταν έχουμε μία μεγάλη συλλογή από δεδομένα συνηθίζεται να κατασκευάζουμε το δέντρο χρησιμοποιώντας την τεχνική bulk loading και στην συνέχεια να χρησιμοποιούμε πράξεις εισαγωγής στο δέντρο για να προσθέσουμε νέες εγγραφές σε αυτό. Σύμφωνα με την τεχνική αυτή, απαιτείται ένα σύνολο από κλειδιά τα οποία θα είναι ταξινομημένα. Επιλέγεται η τάξη του δέντρου και χωρίζονται τα ταξινομημένα κλειδιά σε ομάδες οι οποίες αποτελούν τα φύλλα. Στην συνέχεια αφαιρείται από κάθε φύλλο το τελευταίο κλειδί το οποίο θα τοποθετηθεί σε υψηλότερο επίπεδο του δέντρου. Η διαδικασία συνεχίζεται αναδρομικά μέχρι να κατασκευαστεί ένας κόμβο του οποίου το πλήθος των κλειδιών δεν θα ξεπερνάει την τάξη που ορίστηκε για το δέντρο. Ο κόμβος αυτός θα είναι η ρίζα του δέντρου με την κατασκευή του οποίου, ολοκληρώνεται και η διαδικασία. Εάν το πλήθος των κλειδιών είναι n και η επιθυμητή τάξη του δέντρου είναι B τότε ο αλγόριθμος bulk loading ολοκληρώνεται σε $O(\frac{n}{B})$ βήματα.

7.Περιγραφή Υλοποίησης

Στην ενότητα αυτή περιγράφονται οι αλγόριθμοι για την κατασκευή των δέντρων αλλά και την εκτέλεση ερωτημάτων εύρους οι οποίοι σχεδιάστηκαν στα πλαίσια της παρούσας διπλωματικής εργασίας. Οι αλγόριθμοι που χρησιμοποιήθηκαν για την κατασκευή των δέντρων είναι ο αλγόριθμος BulkInsert και ο αλγόριθμος BulkLoading. Στην ενότητα 7.2 περιγράφουμε τους αλγορίθμους που χρησιμοποιήθηκαν, για την εκτέλεση ερωτημάτων εύρους στο δέντρο που κατασκευάστηκε.

7.1 Κατασκευή Δέντρων στο HBase

Χρησιμοποιούμε σαν είσοδο για την κατασκευή των δέντρων, που περιγράφτηκαν στις προηγούμενες ενότητες δεδομένα αποθηκευμένα στο HDFS. Με την μέθοδο mapreduce η επεξεργασία των δεδομένων γίνεται στην μορφή (key,value), ανάλογα με την ενδιαφερόμενη τιμή. Ο διαχωρισμός των δεδομένων γίνεται με ειδικό partitioner, ο οποίος παίρνει σαν είσοδο το μικρότερο και το μεγαλύτερο κλειδί και υποθέτοντας ομοιόμορφη κατανομή, μοιράζει ανάλογα τα δεδομένα στους αντίστοιχους reducers. Η επιλογή αυτή γίνεται αφενός λόγο της ευκολίας που παρουσιάζει στην υλοποίηση, αφετέρου για να αποκτήσει κάθε reducer συνεχόμενα δεδομένα που θα του χρησιμεύσουν για την σωστή κατασκευή του δέντρου. Το δέντρο που θα προκύψει θα περιέχει δεδομένα από ένα συνεχόμενο εύρος τιμών, το οποίο ορίστηκε από τον partitioner. Τα δεδομένα αποθηκεύονται στον ίδιο πίνακα στο HBase ακολουθώντας μία συγκεκριμένη τεχνική που θα περιγράφει παρακάτω αναλυτικά.



ΣΧΗΜΑ 7.1 - ΑΝΑΠΑΡΑΣΤΑΣΗ ΚΑΤΑΣΚΕΥΗΣ ΔΕΝΤΡΟΥ ΜΕ ΤΗΝ ΜΕΘΟΔΟ MAPREDUCE

Το σχήμα 7.1 παρουσιάζει την γραφική αναπαράσταση της μεθόδου BulkInsert για την κατασκευή του δέντρου. Επίσης στο σχήμα 7.2 φαίνεται ο ψευδοκώδικας του αλγορίθμου BulkInsert.

```
Mapper{
        map(line,data):
                tokens=line.split(".")
                key=tokens.next()
                value=tokens.next()
                output(key,value)
}

Reducer{
        setup():
                root= new node()

        reducer(key,list):
                root=insert(root,key,list)

        cleanup():
                parseTree(root)

        parseTree(node):
                if(node!=null):
                        writeTable(node)
                        for kid in node:
                                parseTree(kid)
}
```

ΣΧΗΜΑ 7.2 – ΨΕΥΔΟΚΩΔΙΚΑΣ BULKINSERT

Υποθέτοντας ότι έχουμε N διαφορετικά κλειδιά από τα δεδομένα προς επεξεργασία και ότι χρησιμοποιούμε R reducers, τότε θα προκύψουν R διαφορετικά δέντρα με την ολοκλήρωση της mapreduce εργασίας. Επιπλέον υποθέτοντας ότι ο partitioner μοιράζει ομοιόμορφα τα δεδομένα στους reducer, πετυχαίνεται η καλύτερη δυνατή εξισορρόπηση φορτίου. Κάθε reducer θα επεξεργαστεί, στην ιδανική περίπτωση N/R διαφορετικά κλειδιά και θα χρειαστεί για την κατασκευή του δέντρου $\frac{N}{R}\log_B \frac{N}{R}$. Η βάση του λογάριθμου καθορίζεται από την τάξη που επιλέχθηκε για το δέντρο. Όπως αναφέρθηκε και παραπάνω η τάξη καθορίζει το ύψος / πλάτος του δέντρου και μπορεί να επηρεάσει την απόδοση του συστήματος. Επίσης είναι σημαντικό να αναφερθεί ότι η επιλογή της τάξης του δέντρου, σχετίζεται άμεσα με το μέγεθος του κάθε region. Οι κόμβοι που προκύπτουν από το δέντρο πρέπει να ομαδοποιούνται σε πολλά διαφορετικά region, για να αποφθεχθεί η συγκέντρωση του φορτίου σε συγκεκριμένους εξυπηρέτες του συστήματος. Αντίθετα για την αποφυγή συχνής επικοινωνίας με τους Region Servers θα πρέπει τα region, να περιέχουν περισσότερες γραμμές του πίνακα άρα και μεγαλύτερο μέγεθος. Η προσεκτική επιλογή της τάξης του δέντρου μπορεί να βελτιώσει σημαντικά την απόδοση του συνολικού συστήματος.

Τα δεδομένα που προκύπτουν από την κατασκευή του δέντρου καταγράφονται σε πίνακα στο HBase, όπως αναφέρθηκε και στις προηγούμενες ενότητες. Ορίζουμε ένα Column Family που περιέχει όλες τις πληροφορίες για το κάθε κόμβο του δέντρου (κλειδιά, τιμές ,παιδιά ,δείκτες στα φύλλα). Καθώς το κάθε κλειδί εμφανίζεται το πολύ μία φορά στο δέντρο δίνουμε σε κάθε κόμβο, κλειδί-γραμμής που αντιστοιχίζεται στο τελευταίο στην σειρά κλειδί που φιλοξενεί. Για το B+Tree προστίθεται ένα ιδιαίτερο αναγνωριστικό στους κόμβους-φύλλα ώστε να είναι εύκολη η ανάκτηση του. Η μέθοδος επιλογής των ονομάτων των κόμβων γίνεται για διευκόλυνση στην προσπέλαση των δεδομένων, ιδιαίτερα για την περίπτωση του B+Tree όπως θα περιγραφεί παρακάτω. Βέβαια η επιλογή αυτή δεν ιδιαίτερη εύχρηστη, στην περίπτωση που θελήσουμε να κάνουμε αλλαγές στο δέντρο προσθέτοντας νέα δεδομένα. Επειδή το χαρακτηριστικό αυτό είναι απαραίτητο για το B+Tree θα μπορούσε να χρησιμοποιηθεί μόνο για αυτό και συγκεκριμένα για τα φύλλα διευκολύνοντας έτσι την επέκταση του δέντρου.

ΣΧΗΜΑ 7.3 - ΓΡΑΜΜΗ ΣΤΟ HBASE ΠΟΥ ΑΝΑΠΑΡΙΣΤΑ ΕΝΑ ΚΟΜΒΟ ΤΟΥ ΔΕΝΤΡΟΥ

7.1 Ερωτήματα Εύρους σε B,B+ Trees

Η οργάνωση που περιγράφηκε παραπάνω διευκολύνει σημαντικά την εκτέλεση ερωτημάτων εύρους. Το HBase υποστηρίζει και αυτό ερωτήματα εύρους στα κλειδιά των γραμμών του πίνακα με την μέθοδο scan. Επομένως με την οργάνωση των δεδομένων με την αντίστοιχη λογική, η ανάκτηση τους με ερωτήματα εύρους είναι εύκολη υπόθεση. Η μορφολογία του δέντρου στην περίπτωση του B+Tree βοηθάει σημαντικά στην αποδοτική εκτέλεση ερωτημάτων εύρους. Αντίθετα για το B-Tree η εκτέλεση ερωτημάτων εύρους είναι πολυπλοκότερη διαδικασία συνεπώς και λιγότερο αποδοτική. Παρακάτω περιγράφεται πιο αναλυτικά ο αντίστοιχος αλγόριθμος που υλοποιήθηκε σε κάθε περίπτωση και στην συνέχεια τους αναλύεται η πολυπλοκότητα, αναδεικνύοντας τα πλεονεκτήματα και τα μειονεκτήματα τους.

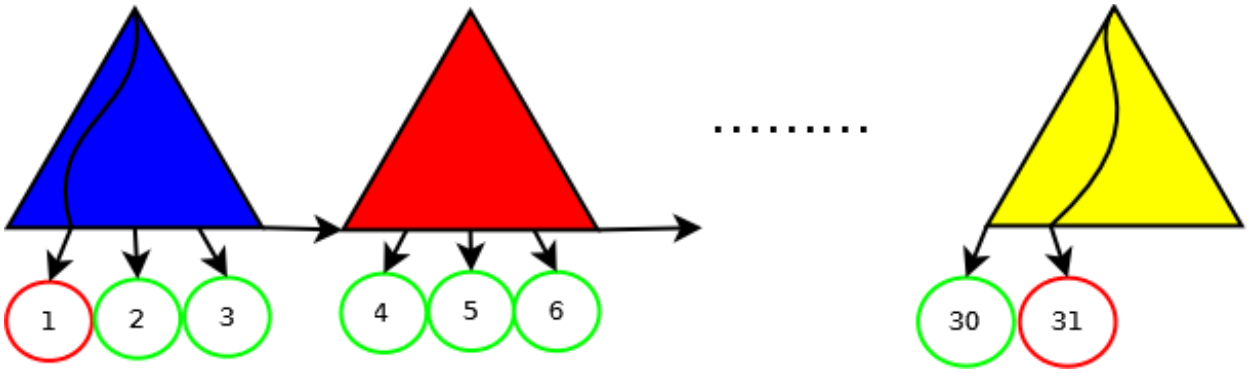
Το B+Tree όπως περιγράφηκε παραπάνω διαθέτει δείκτες που ενώνουν τα φύλλα μεταξύ τους. Επιπλέον οι απαραίτητες πληροφορίες αποθηκεύονται στα φύλλα κατά την κατασκευή του δέντρου. Το συμπέρασμα λοιπόν είναι πώς αρκεί να οργανωθούν τα φύλλα του δέντρου με τέτοιο τρόπο, ώστε όταν εκτελεστεί ένα ερώτημα εύρους στο πίνακα του HBase, να επιστραφούν τα δεδομένα που βρίσκονται στα φύλλα με την σωστή σειρά. Για αυτό το λόγο επιλέγεται να ονομαστούν τα φύλλα του δέντρου όπως περιγράφηκε στην ενότητα 7.1.Αφού ο χρήστης δώσει το εύρος που επιθυμεί, τότε αναζητείται το αντίστοιχο φύλο που περιέχει το αρχικό κλειδί ψάχνοντας τα δέντρα που έχουν προκύψει. Με αντίστοιχο τρόπο ανακτάται το όνομα του φύλλου που περιέχει το τελικό κλειδί που αναζητείται.

```
BpTreeRangeQuery{
      ##NUMBER OF TREES###
      global MIN,MAX
      main(start,end):
            startroot=findRoot(start)
            endroot=findRoot(end)
            startLeaf=findLeaf(startroot,start)
            endLeaf=findLeaf(endroot,end)
```

```
            scanTable(startLeaf,endLeaft)
      ############################
      findRoot(element):
            list=scanTable("root"+MIN,"root"+MAX).getRange()
            for range in list:
                  root=check(element,range)
            return range
      ############################
      findLeaf(root,element):
            leafname=parseTree(root,element)
            return leafname
      ############################
}
```

ΣΧΗΜΑ 7.3 – ΨΕΥΔΟΚΩΔΙΚΑΣ ΕΡΩΤΗΜΑΤΩΝ ΕΥΡΟΥΣ ΣΕ B+TREE

Επειδή τα φύλλα έχουν συγκεκριμένη ονομασία και ταξινομούνται με την σωστή σειρά, αρκεί να εκτελεστεί scan στον πίνακα με ορίσματα το κλειδί των αντίστοιχων φύλλων. Εάν υποθέσουμε ότι έχουμε T δέντρα με τάξη B και σε κάθε δέντρο έχουμε E στοιχεία τότε το ερώτημα εύρους ολοκληρώνεται σε $2\,(T + \log_B E)$ βήματα. Αυτό συμβαίνει γιατί η εύρεση του πρώτου φύλλου ολοκληρώνεται σε $(T + \log_B E)$ βήματα με τον εντοπισμό του δέντρου και την αναζήτηση σε αυτό. Η διαδικασία αυτή εκτελείται δύο φορές, για την εύρεση του καθενός από τα δύο φύλλα. Έτσι ο συνολικός αριθμός βημάτων θα είναι $2\,(T + \log_B E)$. Γραφική αναπαράσταση της μεθόδου που περιγράφηκε φαίνεται στο σχήμα 7.5.



ΣΧΗΜΑ 7.4 - ΕΚΤΕΛΕΣΗ ΕΡΩΤΗΜΑΤΟΣ ΕΥΡΟΥΣ

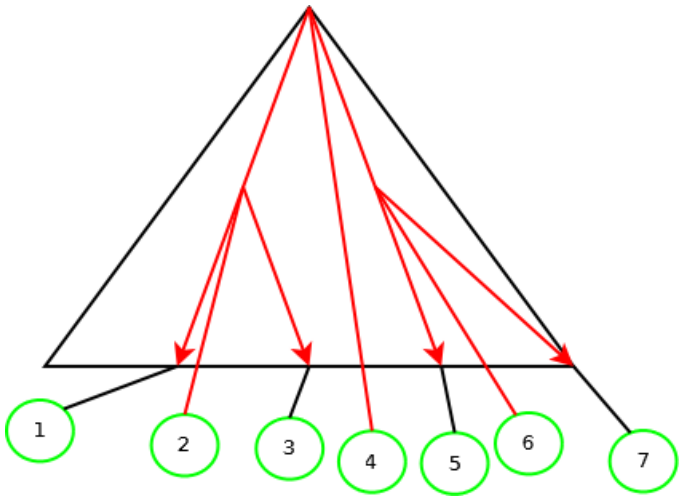
Στην περίπτωση του B-Tree η εκτέλεση ερωτημάτων εύρους είναι πιο πολύπλοκη. Επειδή η ζητούμενη πληροφορία δεν βρίσκεται μόνο στα φύλλα του δέντρου, αλλά και σε εσωτερικούς κόμβους θα πρέπει να προσπελαστεί όλο το δέντρο κάθε φορά για να βρεθεί η απαραίτητη πληροφορία. Αυτό το επιτυγχάνεται εκτελώντας αναζήτηση κατά βάθος(Depth First

Search), σε κάθε δέντρο που περιέχει την ζητούμενη πληροφορία. Αντίστοιχα με την μέθοδο πού χρησιμοποιήθηκε στο B+Tree, εντοπίζονται τα δέντρα που περιέχουν την πληροφορία. Στην συνέχεια εκτελείτε ο αλγόριθμος Depth First Search σε καθένα από τα δέντρα αυτά.

```
BTreeRangeQuery{
	##NUMBER OF TREES BUILT IN HBASE###
	global MIN,MAX
	main(start,end):
		startroot=findRoot(start)
		stoproot=findRoot(stop)
		list=retrieveData(startroot,stoproot);
	############################
	findRoot(element):
		list=scanTable("root"+MIN,"root"+MAX).getRange()
		for range in list:
			root=check(element,range)
		return range
	############################
	retrieveData(start,stop)
		for root in range(start,stop):
			list.append(parseTree(root))
		return list
}
```

ΣΧΗΜΑ 7.5 – ΨΕΥΔΟΚΩΔΙΚΑΣ ΕΡΩΤΗΜΑΤΩΝ ΕΥΡΟΥΣ ΣΕ B-TREE

Το ερώτημα εύρους σε κάθε B-Tree, ολοκληρώνεται ύστερα από ExT Βήματα, καθώς πρέπει να ελεγχθούν T δέντρα τα οποία περιέχουν E διαφορετικά κλειδιά.



ΣΧΗΜΑ 7.6 - ΑΝΑΖΗΤΗΣΗ ΚΑΤΑ ΒΑΘΟΣ

Είναι φανερό ότι ο χρόνος ολοκλήρωσης του ερωτήματος εύρους είναι πολύ χειρότερος από αυτόν στην περίπτωση του B+Tree. Επειδή τα δέντρα δεν έχουν εξάρτηση μεταξύ τους και επειδή τα δεδομένα ανακτώνται από το πίνακα στον HBase, μπορεί να βελτιωθεί ο χρόνος εκτέλεσης των ερωτημάτων με την παραλληλοποίηση της αναζήτησης κατά βάθος. Με τον τρόπο αυτό, πετυχαίνεται σημαντική επιτάχυνση στην ολοκλήρωση της διαδικασίας.

8. Πειράματα & Συμπεράσματα

Στην ενότητα αυτή επιβεβαιώνεται η σωστή λειτουργία των τεχνικών κατασκευής δέντρων, που περιγράφηκαν προηγουμένως μέσο της εκτέλεσης πειραμάτων. Αρχικά στην ενότητα 8.1 περιγράφονται οι προδιαγραφές τους συστήματος, το οποίο χρησιμοποιήθηκε για την εκτέλεση των πειραμάτων. Στην συνέχεια στην ενότητα 8.2 παρατίθενται τα αποτελέσματα των μετρήσεων για την μέθοδο BulkInsert, στα οποία φανερώνονται τα μειονεκτήματα της μεθόδου αυτής. Τέλος στην ενότητα 8.3 περιγράφεται μία πιο αποδοτική μέθοδος για την κατασκευή των δέντρων στο HBase, που βασίζεται στον αλγόριθμο BulkLoading για τα δέντρα. Εκτελούνται αντίστοιχα πειράματα για την μέθοδο αυτή και αναλύονται τα αποτελέσματα.

8.1 Συστήματα & Εργαλεία

Η προδιαγραφές του συστήματος που χρησιμοποιήθηκε για την εκτέλεση των πειραμάτων φαίνονται στο πίνακα 7.1. Η παραγωγή των δεδομένων έγινε με το εργαλείο tpc-H. Το εργαλείο αυτό έχει κατασκευαστεί για την αξιολόγηση, των επιδόσεων συστημάτων διαχείρισης βάσεων δεδομένων. Με χρήση του tpc-H κατασκευάστηκε ο πίνακας Orders από τον οποίο χρησιμοποιήθηκαν, τα χαρακτηριστικά order_key , cust_key για την κατασκευή του δέντρου. Τα χαρακτηριστικά αυτά είναι ακέραιοι αριθμοί, οι οποίοι αντιπροσωπεύουν μοναδικά κλειδιά παραγγελίας και πελάτη αντίστοιχα. Ο κώδικας που υλοποιεί την κατασκευή των δέντρων έγινε με χρήση των Generic της Java. Επομένως ο χρήστης μπορεί να διαλέξει τους δικούς του τύπους δεδομένων, για την κατασκευή του δέντρου που ο ίδιος επιθυμεί.

Hadoop	Έκδοση 1.0.1
HBase	Έκδοση 0.94.1
OS	Debian Base 6.0.5
CPU	4 CPUs per Machine
RAM	2048 MB per Machine
HDD	40 GB per Machine
Number Of Machines	4 Nodes

ΠΙΝΑΚΑΣ 8.1 –ΠΡΟΔΙΑΓΡΑΦΕΣ ΣΥΣΤΗΜΑΤΟΣ

8.2 Πειράματα Μεθόδου BulkInsert

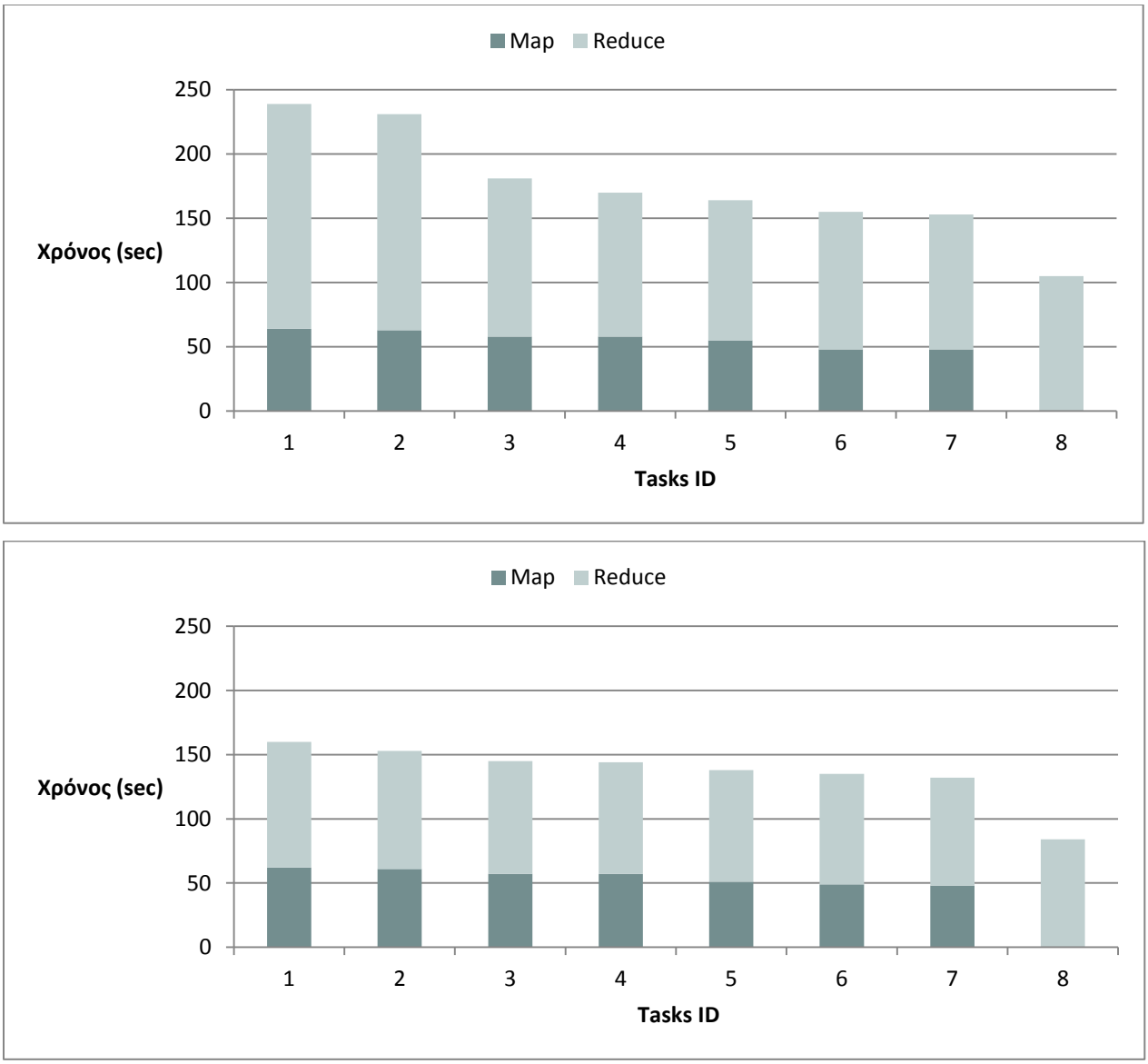
Ο σκοπός των πειραμάτων είναι η επιβεβαίωση της σωστής λειτουργίας των αλγορίθμων που υλοποιήθηκαν άλλα και η αξιολόγηση της αποδοτικότητας τους. Στην ενότητα αυτή θα εκτελεστούν πειράματα κατασκευής δέντρων με την χρήση της μεθόδου BulkInsert. Αρχικά ερευνείται η επιρροή της τάξης του δέντρου στον χρόνο κατασκευής του, αλλά και στον απαιτούμενο αποθηκευτικό χώρο για την διατήρηση του. Στην συνέχεια συγκρίνονται τα δύο δέντρα μεταξύ τους.

	B+Tree	**B-Tree**
Μέγεθος Δεδομένων Εισόδους	230MB	230MB
Μέγεθος Δέντρου στην Έξοδο	2,2 GB	1,4 GB
Χρόνος Εκτέλεσης (sec)	900	451
Μέσος Χρόνος Εκτέλεσης Map (sec)	56,29	55
Μέσος Χρόνος Εκτέλεσης Shuffle (sec)	28	28,75
Μέσος Χρόνος Εκτέλεσης Reduce (sec)	125,5	88,25
Αριθμός Reducer	8	8
Μέγεθος Συνολικής Φυσικής Μνήμης	19525 MB	15222 MB
Τάξη Δέντρου	5	5

ΠΙΝΑΚΑΣ 8.2 – ΜΕΤΡΗΣΕΙΣ ΓΙΑ ΤΗΝ ΚΑΤΑΣΚΕΥΗ ΔΕΝΤΡΩΝ ΤΑΞΗΣ 5

Επιλέχθηκε να κατασκευαστούν τα δέντρα τάξης 5 και 101 αντίστοιχα. Έτσι φανερώνεται η διαφορά τόσο σε απαιτούμενο αποθηκευτικό χώρο όσο και στο συνολικό χρόνο ολοκλήρωσης της διαδικασίας. Στον πίνακα 8.2 φαίνονται τα αποτελέσματα των μετρήσεων για την κατασκευή των δέντρων τάξης 5. Στο σχήμα 8.1 φαίνεται η γραφική παράσταση που περιγράφει την κατανομή του συνολικού χρόνου εκτέλεσης στις επιμέρους διεργασίες map, reduce.

Ο αριθμός των mapper που χρησιμοποιήθηκαν από την διεργασία εξαρτώνται από το μέγεθος της εισόδους σε συνδυασμό με τον μέγεθος του block που ορίστηκε για το HDFS. Το μέγεθος του block που χρησιμοποιήθηκε για τα πειράματα είναι 256 MB.



ΣΧΗΜΑ 8.1 – ΚΑΤΑΝΟΜΗ ΧΡΟΝΟΥ ΚΑΤΑΣΚΕΥΗΣ ΔΕΝΤΡΟΥ ΤΑΞΗΣ 5(B+ & B ΔΕΝΤΡΑ)

Στον πίνακα 8.3 φαίνονται τα αποτελέσματα των μετρήσεων για την περίπτωση του δέντρου τάξης 101.

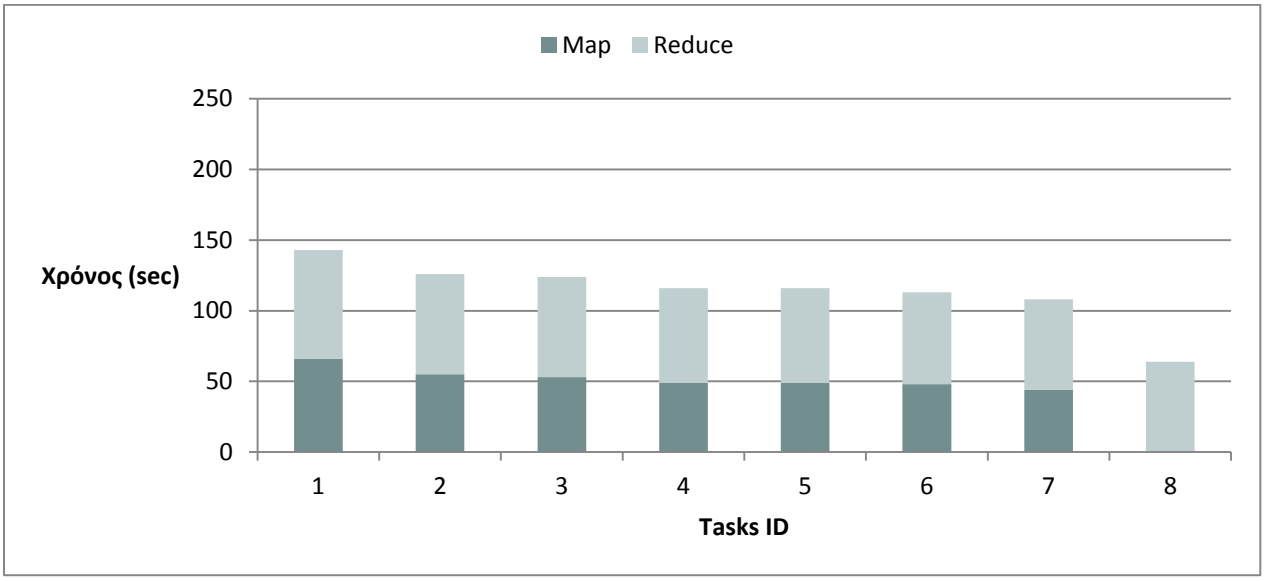
Οι μετρήσεις στους πίνακες 8.2 και 8.3 φανερώνουν την αναμενόμενη διαφορά στον χρόνο ολοκλήρωσης, αλλά και στο απαιτούμενο αποθηκευτικό χώρο για την κατασκευή του δέντρου με τάξη 5 και 101 αντίστοιχα. Όταν η τάξη του δέντρου είναι μικρή, εξαιτίας των συχνών εξισορροπήσεων (rebalance) για την διατήρηση της δομής του, ο χρόνος ολοκλήρωσης της διαδικασίας είναι μεγαλύτερος σε σχέση με τον αντίστοιχο χρόνο για το δέντρο μεγαλύτερης τάξης.

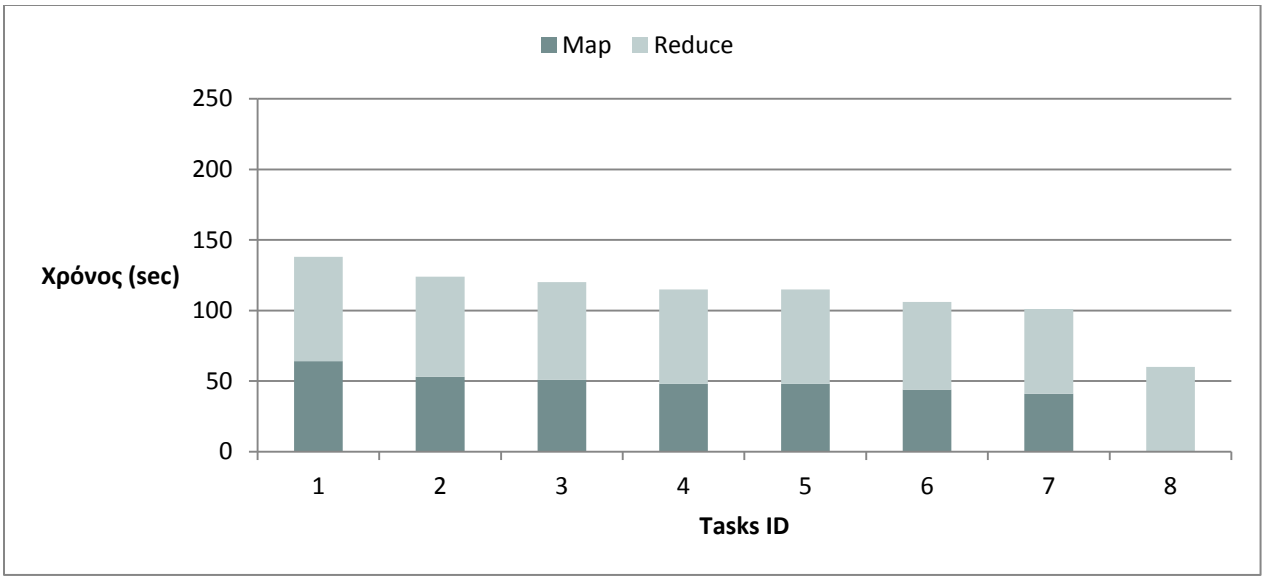
	B+Tree	B-Tree
Μέγεθος Δεδομένων Εισόδους	230MB	230MB
Μέγεθος Δέντρου στην Έξοδο	598,2MB	256MB
Χρόνος Εκτέλεσης (sec)	263	246
Μέσος Χρόνος Εκτέλεσης Map (sec)	52	49,86
Μέσος Χρόνος Εκτέλεσης Shuffle (sec)	28,63	29,75
Μέσος Χρόνος Εκτέλεσης Reduce (sec)	68,25	66,25
Αριθμός Reducer	8	8
Μέγεθος Συνολικής Φυσικής Μνήμης	9501 MB	9286 MB
Τάξη Δέντρου	101	101

ΠΙΝΑΚΑΣ 8.3 – ΜΕΤΡΗΣΕΙΣ ΓΙΑ ΤΗΝ ΚΑΤΑΣΚΕΥΗ ΔΕΝΤΡΩΝ ΤΑΞΗΣ 101

Η διαφορά που υπάρχει μεταξύ του B+ και B-Tree στο τελικό μέγεθος είναι αναμενόμενη, καθώς λόγο των διπλότυπων εγγραφών αλλά και των δεικτών στα φύλλα ο απαιτούμενος αποθηκευτικός χώρος αυξάνεται.

Τέλος η μέθοδος του BulkInsert είναι ιδιαίτερα μνημοβόρα, καθώς είναι αναγκαίο να διατηρηθεί ολόκληρο το δέντρο στην μνήμη πριν να αρχίσει να γράφεται στον πίνακα του HBase.





ΣΧΗΜΑ 8.2 – ΚΑΤΑΝΟΜΗ ΧΡΟΝΟΥ ΚΑΤΑΣΚΕΥΗΣ ΔΕΝΤΡΟΥ ΤΑΞΗΣ 101(B+ & B ΔΕΝΤΡΑ)

Τα συμπεράσματα που προέκυψαν από τα προηγούμενα πειράματα, φανερώνουν την μη αποδοτική φύση της μεθόδου BulkInsert. Ταυτόχρονα η επιλογή τάξης 101 για το δέντρο μας δίνει την απαραίτητη εξισορρόπηση μεταξύ απαιτούμενου αποθηκευτικού χώρου και χρόνου ανάκτησης των δεδομένων στο δέντρο. Επομένως στην επόμενη ενότητα η τάξη των δέντρων που θα κατασκευαστούν με την χρήση του αλγορίθμου BulkLoading, θα είναι 101.

8.3 Προτάσεις & Βελτιώσεις

Στην ενότητα αυτή, περιγράφεται ο αλγόριθμος bulk loading για την κατασκευή του δέντρου και πώς υλοποιείται με την μέθοδο mapreduce. Όπως αναφέρθηκε σε προηγούμενες ενότητες για την εφαρμογή του αλγόριθμου BulkLoading πρέπει να έχουμε ταξινομημένα δεδομένα. Ο κάθε reducer δέχεται τα δεδομένα στην είσοδο ταξινομημένα ανάλογα με το αντίστοιχο κλειδί που παρήγαγε ο αντίστοιχος mapper.

Αφού τα δεδομένα προς επεξεργασία ταξινομούνται πριν τα επεξεργαστεί ο αντίστοιχος reducer, τα διασπάμε σε ομάδες κατά την διαδικασία reduce. Οι ομάδες αυτές αποτελούν τα φύλλα του δέντρου. Όταν ένα ζευγάρι (key,value) μπει στο reducer, τότε ελέγχεται εάν το φύλλο που κατασκευάζεται είναι γεμάτο. Εάν ναι τότε δημιουργείται νέο φύλλο στο οποίο προστίθεται το αντίστοιχο ζευγάρι, αλλιώς το ζευγάρι προστίθεται στο ήδη υπάρχον φύλλο. Όταν ένα φύλλο

γεμίσει, τότε αποθηκεύεται σε ένα buffer και είναι έτοιμο για εγγραφή στο πίνακα του HBase. Όταν ο buffer γεμίσει τότε ξεκινάει η διαδικασία εγγραφής στο πίνακα του HBase. Ανάλογα με τον τύπο του δέντρου, για κάθε φύλλο που δημιουργείται διατηρείται στην μνήμη η αντίστοιχη τιμή για το επόμενο επίπεδο του δέντρου.

Στο cleanup επαναληπτικά, ακολουθείται η ίδια διαδικασία κατά την οποία διασπούνται τα κλειδιά που έχουν απομείνει, μέχρι προκύψει κόμβος ο οποίος δεν θα ξεπερνάει την τάξη του δέντρου. Ο κόμβος αυτός θα αποτελεί την αντίστοιχη ρίζα του δέντρου.

Η τεχνική αυτή πέρα από το χαμηλό αριθμό βημάτων που πετυχαίνει είναι ιδιαίτερα αποδοτική, καθώς περιοδικά γράφει τα δεδομένα του buffer στον πίνακα. Έτσι η απαιτούμενη μνήμη για την επεξεργασία των δεδομένων μπορεί να προσαρμοστεί από τον ίδιο τον χρήστη που εκτελεί την mapreduce εργασία. Στο σχήμα 8.3, φαίνεται ο ψευδοκώδικας για τον που περιγράφει τον αλγόριθμο BulkLoading.

```
Mapper{
	map(line,data):
		tokens=line.split("|")
		key=tokens.next()
		value=tokens.next()
		output(key,value)
}

Reducer{
	global next,curr,buffer,pl
	setup():
		buffer=128
		next= new node()
		curr= new node()
		pl= new PutList(buffer)
	##############################
	reducer(key,list):
		if(curr.order<MAX.ORDER):
			curr.add(key,list)
		else:
			flushNode()
			curr=new node()
			curr.add(key,list)
			next.add(key,list)
	##############################
	cleanup():
		flushNode()
		curr=new node()
		tmp= new node()
		while(next.order>MAX.ORDER)
			count= next.order/ORDER
			for i in range(count):
				count.add(next.getK(0),next.getC(0))
				next.removeK(0) next.removeC(0)
				tmp.add(next.getK(0),next.getC(0))
				next.removeK(0) next.removeC(0)
```

```
                        flushNode()
                        curr=new node()
                    next=tmp
                    tmp=new node()
                curr=new node()
                curr=next
                flushNode()
        ##############################
        flushNode():
                if(pl.size()<=BUFFER):
                    pl.add(curr)
                else:
                    flushToTable(pl)
}
```

ΣΧΗΜΑ 8.3 – ΨΕΥΔΟΚΩΔΙΚΑΣ BULKLOADING

8.4 Πειράματα Μεθόδου BulkLoading

Στην ενότητα αυτή παρουσιάζονται τα πειράματα που εκτελέστηκαν για την αξιολόγηση της μεθόδου BulkLoading. Η τάξη των δέντρων που χρησιμοποιήθηκε είναι 101, για του λόγους που αναφέρθηκαν στην ενότητα 8.1. Αρχικά επιβεβαιώνεται από τα πειράματα που εκτελέστηκαν, η σαφής υπεροχή του αλγορίθμου BulkLoading απέναντι στον αλγόριθμο BulkInsert. Στην συνέχεια αναλύεται η επιρροή της αύξησης του buffer στην συνολική επίδοση του BulkLoading.

Στο πίνακα 8.4 καταγράφονται οι μετρήσεις που προέκυψαν από την εκτέλεση του αλγορίθμου BulkLoading για την κατασκευή δέντρων τάξεως 101. Ο Buffer που χρησιμοποιήθηκε είναι 128 (διατηρούμε 128 Put αντικείμενα στην μνήμη πριν κάνουμε flush στον πίνακα του HBase).

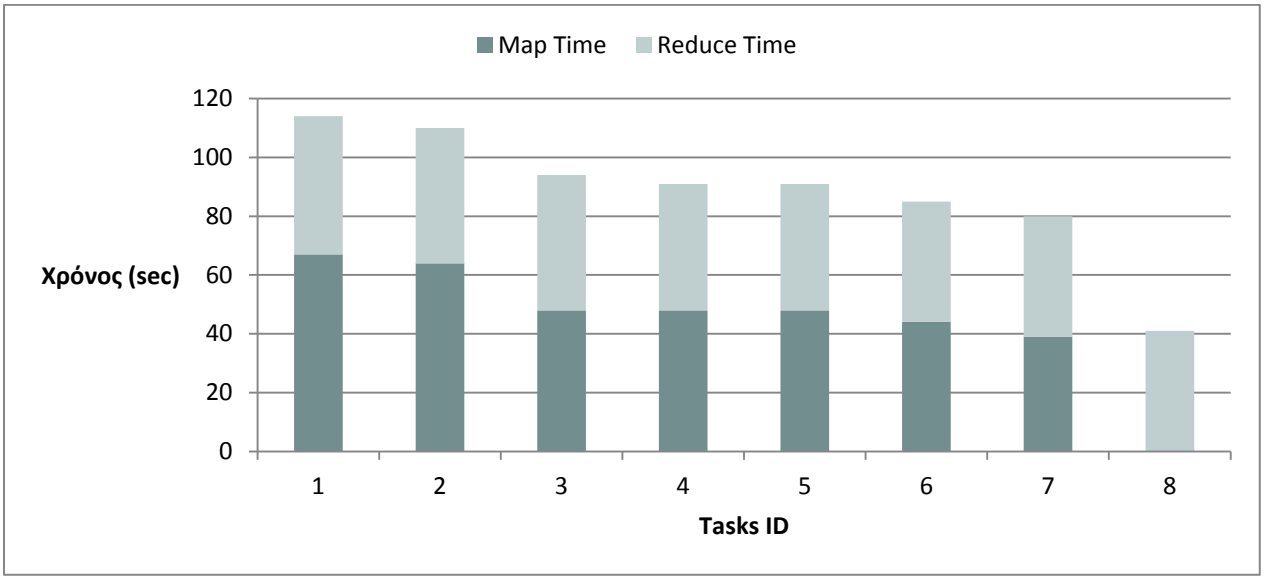
Η αποδοτικότητα του αλγορίθμου επιβεβαιώνεται τόσο από την μείωση του μέσου χρόνου εκτέλεσης του κάθε reducer, όσο και από την μείωση του συνολικού χρόνου. Επίσης διαπιστώνεται από τις μετρήσεις ότι το μέγεθος της φυσικής μνήμης που απαιτείται για την εκτέλεση του αλγορίθμου είναι πολύ μικρότερο σε σχέση με την υλοποίηση του BulkInsert

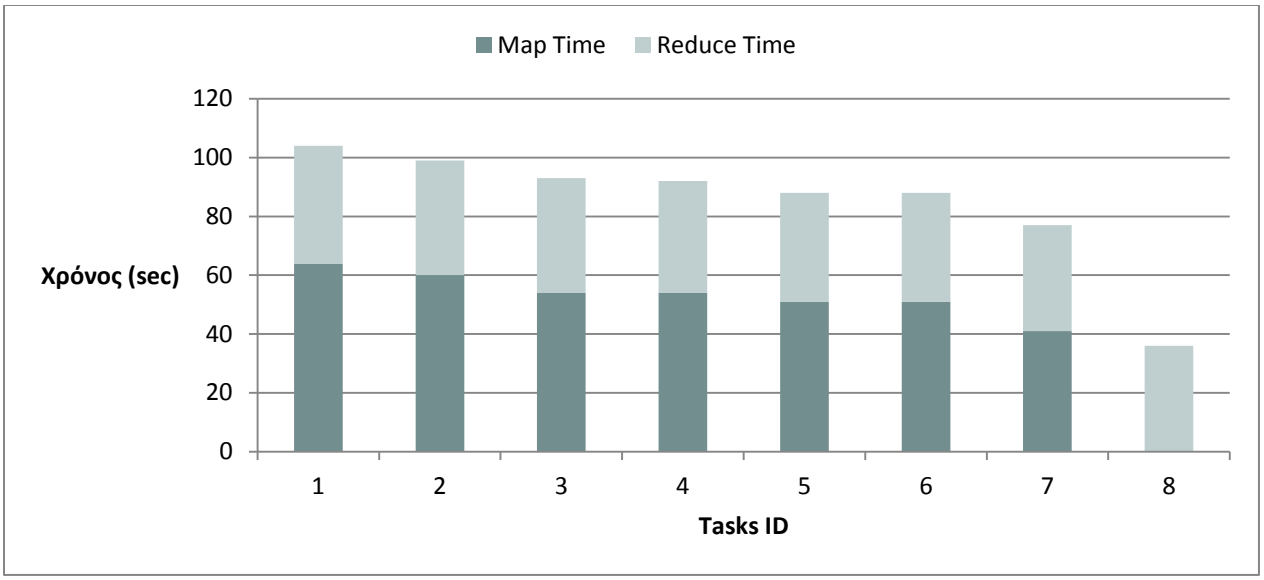
	B+Tree	**B-Tree**
Μέγεθος Δεδομένων Εισόδους	230MB	230MB
Μέγεθος Δέντρου στην Έξοδο	267,1 MB	256MB
Χρόνος Εκτέλεσης (sec)	132	125
Μέσος Χρόνος Εκτέλεσης Map (sec)	51,14	53,57

Μέσος Χρόνος Εκτέλεσης Reduce (sec)	43,5	37,75
Αριθμός Reducer	8	8
Μέγεθος Buffer (Put Objects)	128	128
Μέγεθος Συνολικής Φυσικής Μνήμης	6517 MB	6165 MB

ΠΙΝΑΚΑΣ 8.4 - ΜΕΤΡΗΣΕΙΣ ΓΙΑ ΤΗΝ ΚΑΤΑΣΚΕΥΗ ΔΕΝΤΡΩΝ ΜΕ BULKLOADING (BUFFER 128)

Στο σχήμα 8.3, απεικονίζεται η γραφική παράσταση της κατανομής του συνολικού χρόνου εκτέλεσης στις επιμέρους διεργασίες Map,Reduce. Όπως ήταν και αναμενόμενο το μεγαλύτερο ποσοστό του συνολικού χρόνου εκτέλεσης καταναλώνεται από την reduce διεργασία.





ΣΧΗΜΑ 8.3 - ΚΑΤΑΝΟΜΗ ΧΡΟΝΟΥ ΚΑΤΑΣΚΕΥΗΣ ΜΕ BUFFER 128 (B+ & B ΔΕΝΤΡΑ)

Ο μέσος χρόνος εκτέλεσης της διαδικασίας Map παραμένει ίδιος καθώς δεν υπάρχει αλλαγή στα δεδομένα ούτε στην μέθοδο map. Επίσης ο χρόνος για την κατασκευή του B-Tree είναι μικρότερος από αυτόν του B+Tree, όπως ήταν και αναμενόμενο καθώς χρειάζεται σχεδόν την διπλάσια πληροφορία για την κατασκευή του.

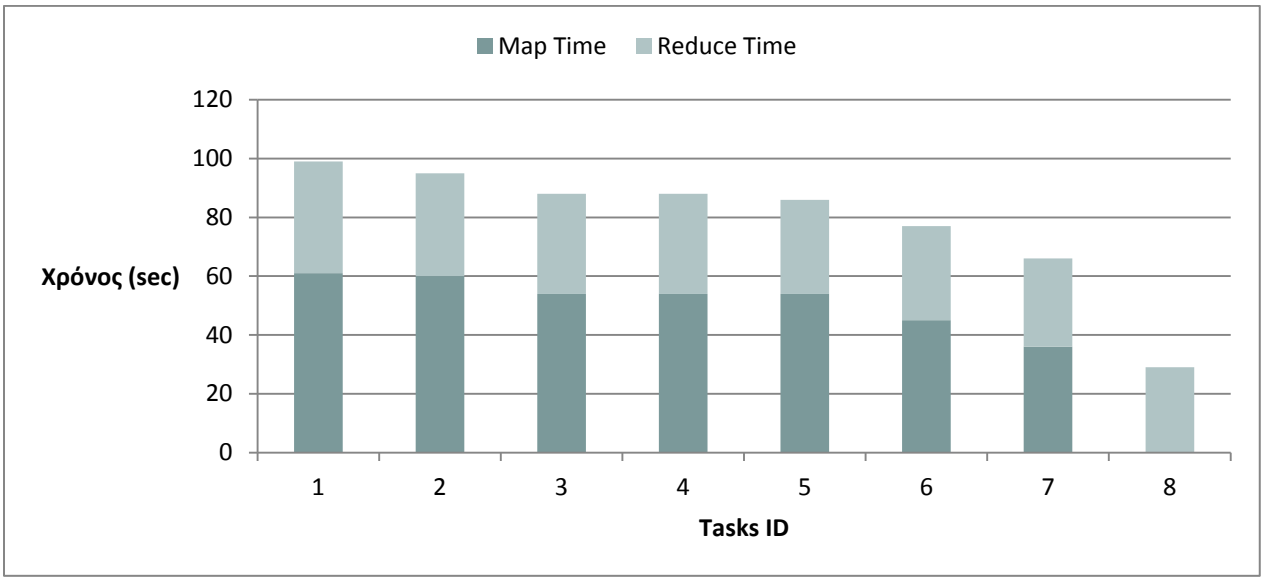
Παρόλο που το μέγεθος του δέντρου παραμένει ίδιο, η απαιτούμενη μνήμη που για την εκτέλεση της διαδικασίας μειώνεται δραματικά, καθώς η διαχείριση των δεδομένων γίνεται με χρήση του buffer, ο οποίος περιοδικά γράφει τα δεδομένα στο πίνακα του HBase. Το μέγεθος όμως του Buffer επηρεάζει και αυτό με την σειρά του την απόδοση του συστήματος. Εάν επιλεχθεί μικρός σε μέγεθος buffer, τότε θα εκτελούνται συχνά εγγραφές με αποτέλεσμα να χάνεται χρόνος από την κατασκευή του δέντρου. Επίσης με μικρό Buffer, δημιουργείτε η ανάγκη για συχνή δέσμευση και αποδέσμευση μνήμης, διαδικασία που είναι ιδιαίτερα χρονοβόρα.

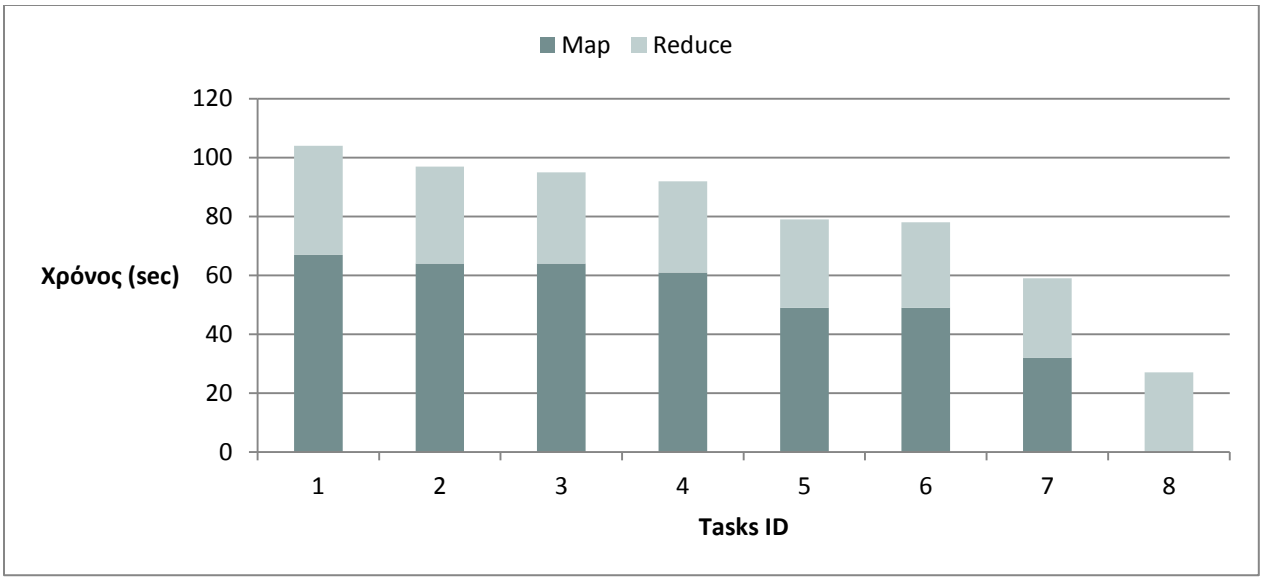
Στο σημείο αυτό εκτελέστηκαν τα ίδια πειράματα με προηγουμένως, αφού πρώτα αυξήθηκε το μέγεθος του buffer. Τα πειράματα που εκτελέστηκαν με το αυξημένο buffer μας έδωσαν τα αποτελέσματα του πίνακα 8.5.

	B+Tree	B-Tree
Μέγεθος Δεδομένων Εισόδους	230MB	230MB
Μέγεθος Δέντρου στην Έξοδο	267,1MB	256MB
Χρόνος Εκτέλεσης(sec)	114	108
Μέσος Χρόνος Εκτέλεσης Map (sec)	52	55,14
Μέσος Χρόνος Εκτέλεσης Reduce (sec)	33	30,63
Αριθμός Reducer	8	8
Μέγεθος Buffer (Put Objects)	512	512
Μέγεθος Συνολικής Φυσικής Μνήμης	6613 MB	6678 MB

ΠΙΝΑΚΑΣ 8.5 - ΜΕΤΡΗΣΕΙΣ ΓΙΑ ΤΗΝ ΚΑΤΑΣΚΕΥΗ ΔΕΝΤΡΩΝ ΜΕ BULKLOADING (BUFFER 512)

Η φυσική μνήμη που δεσμεύτηκε για την ολοκλήρωση της κατασκευής των δέντρων δεν αυξήθηκε σημαντικά. Παρόλα αυτά ο χρόνος ολοκλήρωσης της διαδικασίας μειώθηκε σημαντικά. Το γεγονός αυτό δηλώνει ότι υπάρχει περιθώριο για περαιτέρω βελτίωση του χρόνου εκτέλεσης, χωρίς να απαιτείται αρκετά μεγαλύτερή φυσική μνήμη. Στο σχήμα 8.4 φαίνεται η κατανομή του συνολικού χρόνου εκτέλεσης, στις αντίστοιχες διεργασίες της mapreduce εργασίας. Είναι φανερή η μείωση στο συνολικό χρόνο εκτέλεσης της εργασίας, τόσο σε σχέση με την μέθοδο BulkInsert αλλά και με την αύξηση του buffer της μεθόδου BulkLoading.





ΣΧΗΜΑ 8.4 – ΚΑΤΑΝΟΜΗ ΧΡΟΝΟΥ ΚΑΤΑΣΚΕΥΗΣ ΜΕ BUFFER 512 (B+ & B ΔΕΝΤΡΑ)

8.5 Συμπεράσματα

Με την σύγκριση των δύο υλοποιήσεων κατασκευής B, B+ δέντρων στο HBase είναι φανερό ότι η τεχνική BulkLoading υπερτερεί τις τεχνικής BulkInsert. Το γεγονός αυτό φαίνεται από τι μείωση του συνολικού χρόνου εκτέλεσης αλλά και από την μείωση του μέσου χρόνου εκτέλεσης ανά Reducer. Επίσης παρατηρούμε ραγδαία μείωση στην απαιτούμενη φυσική μνήμη που χρειάζεται η εργασία mapreduce για να ολοκληρωθεί.

Τα πειράματα που εκτελέστηκαν δεν δίνουν σαφή διάκριση μεταξύ των διαφορετικών τύπων δέντρων που υλοποιήθηκαν. Βασιζόμενοι στην ανάλυση των ενοτήτων 6,7 & 8 μπορούμε να συμπεράνουμε τα εξής. Το B+ Tree απαιτεί σαφώς μεγαλύτερο αποθηκευτικό χώρο αλλά ταυτόχρονα είναι πιο αποδοτικό σε ερωτήματα εύρους. Αντίθετα το B-Tree απαιτεί μικρότερο αποθηκευτικό χώρο και είναι ιδιαίτερα αποδοτικό στην αναζήτηση μοναδικών κλειδιών στο δέντρο, δεν υποστηρίζει όμως αποδοτικά ερωτήματα εύρους. Η χρήση των B+ δέντρων είναι ευρέος διαδεδομένη στις βάσεις δεδομένων. Επομένως ανάλογα με τις ανάγκες του χρήστη μπορούμε να επιλεχθεί ο κατάλληλος τύπος δέντρου για υλοποίηση.

Τέλος φαίνεται ότι η κατασκευή των δέντρων στο HBase μπορεί να επισπεύσει σημαντικά την προσπέλαση των δεδομένων. Χρησιμοποιώντας τις κατάλληλες ρυθμίσεις, μπορούν να μοιραστούν οι κόμβοι στους Region Servers του συστήματος, ώστε να

επιτυγχάνεται η καλύτερη δυνατή εξισορρόπηση του φορτίου. Τα δέντρα που κατασκευάστηκαν μπορούν να χρησιμοποιηθούν σε συνδυασμό με την τεχνική prewarm, σύμφωνα με την οποία, όταν εκτελείται ένα ερώτημα εύρους χρησιμοποιώντας τα άκρα και εντοπίζοντας τους αντίστοιχους κόμβους του δέντρου, αποστέλλεται μήνυμα στους αντίστοιχους Region Servers, ώστε να ξεκινήσουν την διαδικασία της ανάκτησης πριν επικοινωνήσει ο client άμεσα με αυτούς.

Βιβλιογραφία – Αναφορές

[1]. Information Retrieval - http://en.wikipedia.org/wiki/Information_retrieval

[2]. Cap Theorem- http://en.wikipedia.org/wiki/CAP_theorem

[3]. Web Crawler & Distributed IR – Ron Jin

[4]. The Google File System - Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung

[5].The Hadoop Distributed FileSystem - Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler

[6]. PVFS: A Parallel File System for Linux Clusters

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

[8].HBasics: An Introduction to Hadoop HBase - Michael Stack

[9].HBase:A Comprehensive Introduction - James Chin, ZikaiWang

[10].HBase @ Meetup - Gary Helmling

[11]. HBase Page- http://hbase.apache.org/

[12].Distributed File System Review – Schuber Zhang

[13]. Distributed File Systems Design -Paul Krzyzanowski

[14]. Distributed Computing- http://en.wikipedia.org/wiki/Distributed_computing

[15]. MapReduce: Simplified Data Processing on Large Clusters - Jeffrey Dean & Sanjay Ghemawat

[16]. Chapter 15. BTrees -Donghui Zhang Northeastern University

[17]. B+ - Tree & B – Tree - Phi Thong Ho

[18]. B-Tree - http://en.wikipedia.org/wiki/B-tree

[19]. B+Tre - http://en.wikipedia.org/wiki/B%2B_tree